Projekt TIN - Dokumentacja

Wiktor Michalski
Przemysław Stawczyk
Maciej Szulik
Kamil Zacharczuk

January 28, 2020

Contents

Part I

Dokumentacja logiczna

Chapter 1

Zadanie

1.1 Tresć Zadania

Napisać program obsługujący prosty protokół P2P

1. Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach i takiej samej wielkości pliku w bajtach.
2. Początkowo dany zasób znajduje się w jednym węźle sieci, następnie może być propagowany do innych węzłów w ramach inicjowanego przez użytkownika ręcznie transferu - raz pobrany zasób zostaje zachowany jako kopia.
3. Po pewnym czasie działania systemu ten sam zasób może się znajdować w kilku węzłach sieci (na kilku maszynach).
4. System ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
5. Program powinien umożliwiać współbieżne:
    - wprowadzanie przez użytkownika (poprzez interfejs tekstowy):
        - nowych zasobów - z lokalnego systemu plików
        - poleceń pobrania nazwanego zasobu ze zdalnego węzła
    - pobieranie zasobów (także kilku jednocześnie)
    - rozgłaszanie informacji o posiadanych lokalne zasobach
6. W przypadku pobierania zdalnego zasobu system sam (nie użytkownik) decyduje skąd zostanie on pobrany.
7. Powinno być możliwe pobranie zasobu z kilku węzłów na raz (tj. "w kawałkach").
8. Zasób pobrany do lokalnego węzła jest kopią oryginału, kopia jest traktowana tak samo jak oryginał (są nierozróżnialne). Istnienie kopii jest rozgłaszane tak samo jak oryginału.
9. Właściciel zasobu może go unieważnić wysyłając odpowiedni komunikat rozgłaszany. Wszystkie kopie zasobu powinny przestać być rozgłaszane. W przypadku trwających transferów zasobów powinny się one poprawnie zakończyć, dopiero wtedy informacja o zasobie może zostać usunięta.

1.2 Wariant zadania dla zespołu

4. Opóźnienia dla wybranego węzła - węzeł reaguje, ale (czasami) z dużym opóźnieniem.

1.3 Interpretacja Zadania

Doprecyzowanie treści i dodatkowe założenia

- W związku z tym, że kopia i oryginał są nierozróżnialne, zasób może być unieważniony przez dowolnego użytkownika, który go posiada.
- Każdy węzeł okresowo rozgłasza informację o posiadanych zasobach. Unieważnienie pliku oznacza, że żaden z węzłów nie będzie już rozgłaszał faktu posiadania tego pliku.
- Unieważnienie wysyłane jest asynchronicznie poprzez broadcast UDP.
- W przypadku unieważnienia pliku w trakcie trwającego przesyłu tego pliku przesyłanie kończy się sukcesem, o ile nie wystąpią inne błędy. Nowy posiadacz pliku nie będzie jednak nigdy rozgłaszał o nim informacji.
- Każdy węzeł przechowuje listy dostępnych zasobów każdego innego węzła. Po odebraniu rozgłoszenia listy zasobów od innego węzła lista ta jest nadpisywana w pamięci węzła odbierającego. Informacje o węźle, w tym lista jego zasobów, są usuwane w przypadku braku, przez ustalony czas, nadchodzącego rozgłoszenia jego listy zasobów.
- W przypadku połączenia z innym węzłem w celu pobrania od niego pliku oczekiwanie na odpowiedź tego węzła ma pewien timeout. Ponadto, jeżeli węzeł przekracza pewien ustalony czas odpowiedzi (nawet jeżeli nie dochodzi do timeout'u), to inkrementujemy licznik "opóźnień" tego węzła (każdy węzeł przechowuje takie liczniki dla każdego innego węzła). Po osiągnięciu ustalonej wartości licznik ten jest zerowany, a węzeł zliczający nie będzie próbował łączyć się z "opóźnionym" węzłem przez pewien określony czas.
- Jeśli węzeł niespodziewanie zakończy połączenie TCP i przestanie rozgłaszać swoją tablicę, to po pewnym czasie pozostałe węzły uznają to za opuszczenie przez niego sieci.
- W przypadku gdy pojawi się błąd w trakcie transferu TCP, usuwamy pobrane dane (segmenty) i kończymy wątek pobierający. Ponowne pobieranie od tego węzła będzie odbywać się po ponownym połączeniu z węzłem.

Chapter 2

Opis Funkcjonalny Projektu

Użytkownik może wprowadzać tekstowe komendy, aby za ich pomocą

- *wyświetlić listę zasobów obecnych w systemie,*
- *wyświetlić listę dostępnych komend,*
- *opuścić system,*

a także wykonywać operacje na plikach, wśród których rozróżniamy:

- *dodanie pliku,*
  można dodać do zasobów plik, którego nazwa nie wystąpiła jeszcze wśród plików w lokalnym rejestrze zasobów i reszcie sieci.
  Zakładamy, że nie wystąpi sytuacja, gdy więcej niż jeden użytkownik doda plik o tej samej nazwie "jednocześnie" - to znaczy przed "zauważeniem" przez całą sieć dodania pliku o takiej nazwie przez któregokolwiek z nich.
- *usunięcie pliku,*
  można usunąć plik z własnego rejestru zasobów.
- *unieważnienie pliku,*
  można unieważnić plik, który mamy we własnym rejestrze zasobów. Oznacza to, że zasób nie będzie już widoczny na liście plików dostępnych w systemie, ale dotychczasowi posiadacze nadal będą go posiadali w swoim lokalnym systemie plików.
- *pobranie pliku,*
  można pobrać plik, którego nie mamy jeszcze w rejestrze zasobów, a który jest obecny w systemie.

Chapter 3

Protokół

3.1 Składnia :

```
<Command 1 octet> ::= <REVOKE> | <FILE_LIST> | <REQ_CONN> | <REQ_SEGMENT>

<ResourceHeader> ::=
        <nazwa pliku: 256 octets, NULL terminated> <rozmiar pliku: 64bit>

<Message> ::= <Command> | <Command><ResourceHeader> |
        <Command><ResourceHeader><Options> |
        <Command><No_Of_Files 16bit><ResourceHeader><ResourceHeader>...

<Resource> ::= <1 KB of file>
```

3.2 Komendy :

- *Unieważnienie pliku:*
  Broadcast po UDP:
  `<Command = REVOKE><ResourceHeader = Revoked File>`
- *Rozgłaszanie dostępnych plików:*
  Broadcast po UDP:
  `<Command = FILE_LIST><No_Of_Files = liczba dostępnych plików>`
  `<ResourceHeader = plik1>....`
- *Żądanie utworzenia połączenia TCP:*
  Wysyłane do węzła po TCP: `<Command = REQ_CONN>`
- *Żądanie pobrania segmentu:*
  Wysyłane do węzła po TCP:
  `<Command = REQ_SEGMENT><ResourceHeader = plik><Options = segment number>`

Chapter 4

Organizacja Projektu

4.1 Współbieżność

Ogólna koncepcja zakłada istnienie wspólbierznie działań:

1. *Obsługa przychodzących żądań przesłania posiadanego zasobu.*
   Nasłuchuje na porcie TCP. W przypadku nawiązania połączenia na tym porcie tworzony jest watek Wątek ten odbiera żądanie przesłania lokalnie posiadanego pliku i nadzoruje to przesłanie.
2. *Odbiór komunikatów broadcast UDP.*
   Komunikaty te obejmują okresowe rozgłaszanie przez każdego użytkownika listy lokalnie posiadanych zasobów, rozgłaszanie unieważnienia zasobu. Po odebraniu komunikatu wątek ten przekazuje otrzymane informacje do bazy danych programu.
3. *Okresowe rozgłaszanie lokalnej listy zasobów.*
   Co pewien czas rozgłaszana jest przez UDP lista zasobów, które host udostępnia do pobrania.
4. *Zachowywanie spójności danych o plikach w sieci*
   Struktura danych przechowująca informacje o plikach umożliwiająca współbieżny dostęp. Działanie przez jego interfejs zachowuje spójność danych
5. *Obsługa interfejsu użytkownika.*
   Interakcja z użytkownikiem przez CLI. Odbieranie komend od użytkownika i odpowiednie reagowanie - powoływanie nowych wątków, które mają zająć się realizacją komendy, między innymi:
   - W przypadku chęci pobrania pliku tworzony jest wątek nadzorujący to pobieranie. Na potrzeby połączenia z wieloma węzłami może on stworzyć kilka kolejnych wątków przypisanych do połączeń z węzłami.
     W zależności od wyniku pobierania przekaże odpowiednie informacje do wątku synchronizującego dane.
   - W przypadku chęci rozgłoszenia faktu posiadania pliku można dodać informację o nim do systemu
   - Wyświetlanie informacji o plikach przechowywanej w programie *tj* w sieci i w rejestrze hosta

   Wyniki działań powyższych wątków przekazywane są z powrotem do CLI, który wyświetla je użytkownikowi.

4.2 Wykorzystanie Protokołów

- **Protokół UDP** - użyty do rozgłaszania informacji o lokalnie posiadanych zasobach oraz o unieważnieniu pliku. Wybraliśmy go do realizacji powyższych celów, ponieważ nie wymaga zestawiania połączenia i umożliwia łatwe rozgłaszanie informacji do wszystkich węzłów sieci. Zestawianie połączenia w tym wypadku byłoby skrajnie nieefektywne.
- **Protokół TCP** - użyty do przesyłania plików pomiędzy węzłami, a także do wysyłania żądań pobrania pliku, bezpośredniego poprzedzających transfer zasobu. Ponieważ protokół ten umożliwia zestawienie trwałego połączenia pomiędzy hostami i zapewnia abstrakcję strumienia danych, jest wygodny, gdy chcemy realizować wysyłanie żądań do konkretnego węzła i odbieranie od niego segmentów plików w odpowiedzi - zazwyczaj kilku w ramach jednej sesji.

Chapter 5

Implementacja

5.1 Spis użytych narzędzi

- Jezyki : *C++*
- Biblioteki : *Boost, std*
- System budowania : *CMake*
- Dokumentacja funkcji : *Doxygen*
- Tworzenie dokumentacji : *LaTeX*
- Testy na jednej maszynie : *Docker*
- Analiza ruchu siecowego : *Wireshark*

5.1.1 CMake

Użyliśmy go do łatwiejszego zarządzania dużym projektem. Dodatkowe zewnętrzne zalezności są wyszukiwane przez narzedzie, tak samo zmiany w ilości i nazwach plików

5.1.2 Boost::asio

Użyliśmy jej do realizacji serwera i klienta TCP oraz serwera i klienta UDP. Jest to biblioteka udostępniająca wiele użytecznych narzędzi do programowania sieciowego, a jedną z możliwości, które wprowadza, jest tworzenie asynchronicznych serwerów i klientów TCP i UDP. Napisane przez nas serwery TCP i UDP, korzystając z tego narzędzia, realizują całą komunikację sieciową w jednym wątku. W przypadku nadejścia żądania połączenia TCP jest ono asynchronicznie akceptowane, przy czym tworzony jest nowy obiekt sesji, który następnie asynchronicznie realizuje komunikację przez sieć. W tym samym czasie może działać wiele takich obiektów.

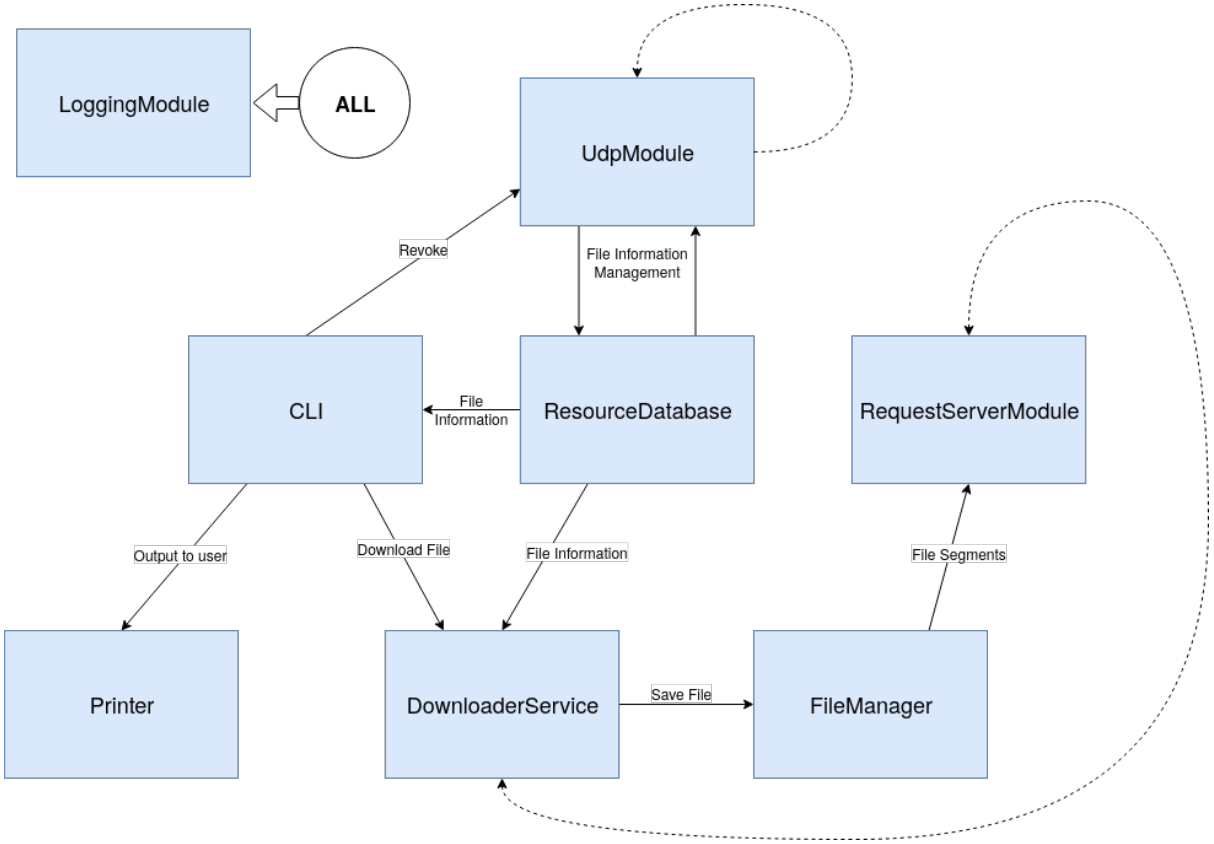
5.1.3 Docker

Użyliśmy go do realizacji testów na jednej maszynie. Dzięki zwirtualizowanemu środowisku możliwe jest tworzenie sieci o różnej ilości hostów.

5.2 Moduły

1. **CLI** - moduł odpowiedzialny za interpretowanie żądań użytkownika, przekazywanie poleceń innym modułom w reakcji na te żądania, a także przygotowywanie informacji do wyświetlenia użytkownikowi. Oparty jest na działającym w pętli wątku.
2. **UdpModule** - moduł serwera i klienta UDP odpowiedzialny za rozgłaszanie i odbieranie informacji o posiadanych przez węzeł zasobach i o unieważnieniu zasobu. Okresowo wysyła do wszystkich węzłów w sieci informację o lokalnych zasobach, a na polecenie CLI rozgłasza unieważnienie zasobu.Po odebraniu komunikatu od innego węzła przekazuje ją do *ResourceDatabase*, aby ten uaktualnił swoje informacje o zasobach innych węzłów.
3. **RequestServerModule** - moduł odbierający żądania pobrania zasobów oraz wysyłający dany zasób w odpowiedzi. Oparty jest na asynchronicznym, działającym stale w jednym wątku serwerze TCP. W momencie nadejścia połączenia zestawiana jest sesja, podczas której asynchronicznie odbierany jest komunikat zawierający nazwę zasobu i numer segmentu, korzystając z obiektu *FileManager* buforowany jest odpowiedni fragment pliku, po czym następuje asynchroniczne wysłanie go po TCP. Sesja jest kończona w momencie odebrania komunikatu z prośbą o zakończenie połączenia. Wiele takich sesji może działać "jednocześnie" ze względu na wspomnianą asynchroniczność realizowanej komunikacji.
4. **DownloaderService** - moduł wysyłający żądania pobrania zasobów i odbierający dany zasób w odpowiedzi. Oparty jest na synchronicznej wersji biblioteki booost::asio. Przez TCP łączy się z modułem RequestServerModule. W ramach tego modułu tworzonych jest kilka wątków - klientów TCP łączących się z różnymi węzłami, tak, aby można było zasób pobierać równolegle od wielu hostów. Jednorazowo przez sieć pobierany jest segment o rozmiarze 1 KB. Po pobraniu wszystkich segmentów w tym module odbywa się kompletowanie ich w ciągły plik w odpowiedniej kolejności. Tak przygotowane dane zapisywane są na dysku przy pomocy obiektu *FileManager*. Informacja o nowym lokalnym zasobie przekazywana jest również do *ResourceDatabase*. Nadzoruje naliczanie timeoutów poszczególnym hostom.
5. **FileManager** - API służące do odczytywania zasobów z systemu plików i buforowania ich, a także do zapisywania na dysku zbuforowanych danych.
6. **ResourceDatabase** - moduł przechowujący informacje o plikach i hostach w sieci. Udostępnia synchronizowany interfejs dla pozostałych modułów programu.
7. **Printer** - moduł zbierający informacje do wyświetlenia użytkownikowi
8. **LoggingModule** - Logi są wypisywane na standardowe wyjście błędów *stderr* co umożliwia ich łatwe przekierowanie do pliku. Moduł logujący zbiera informacje od wszystkich wątków grupując je w kolejkę i wypisując na wyjście. Zapewnia on synchronizację przekazywania logów.

   Format logów jest następujacy :
   <logline> := [<data i godzina wystąpienia zdarzenia>]<zdarzenie z informacją o wysyłającym oraz jego szczegółach>



5.3 Ważniejsze struktury danych

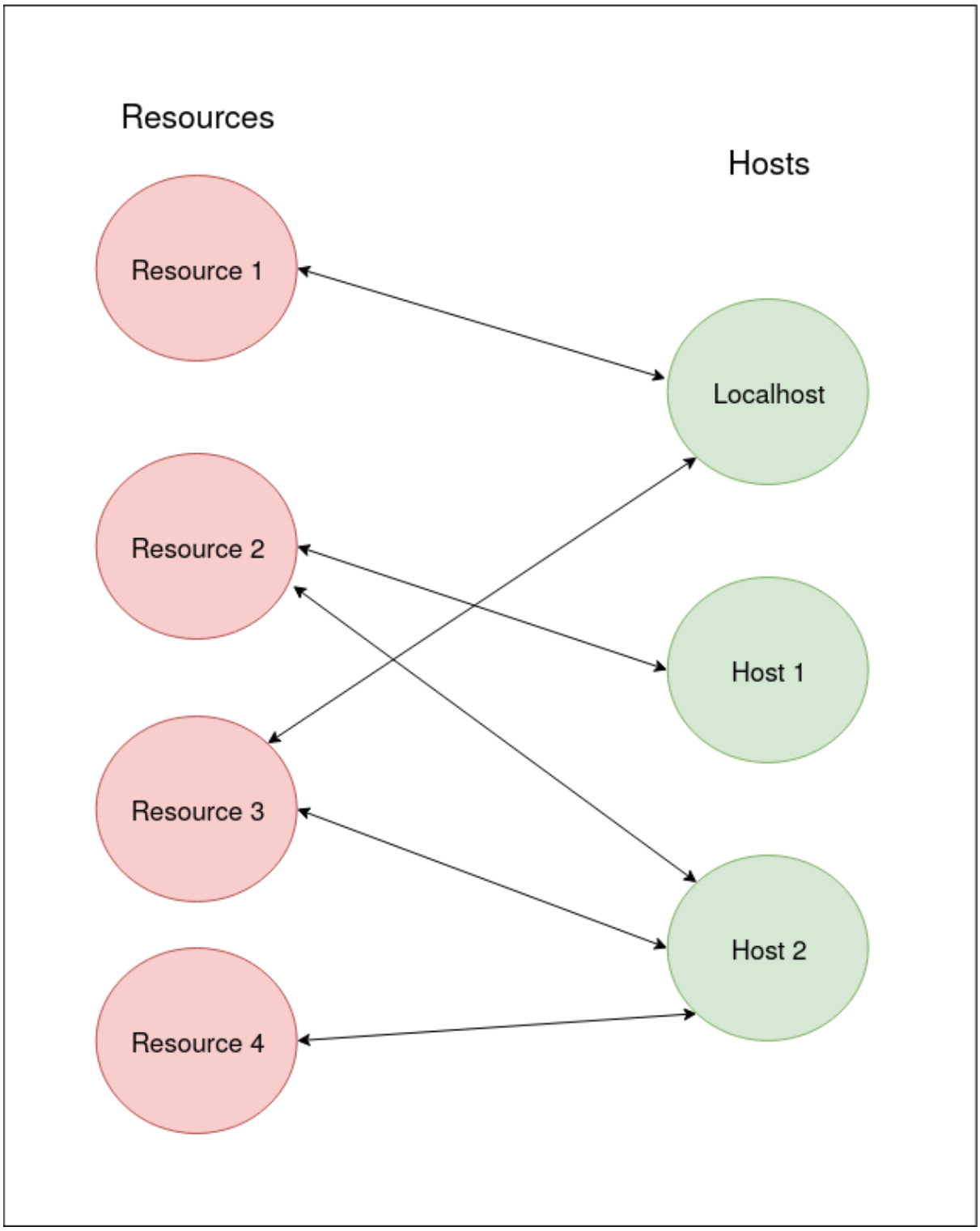
5.3.1 CompleteResource

Klasa reprezentująca kompletny zasób, przechowująca zarówno wskaźnik do zbuforowanej zawartości pliku, jak i informacje o nim, takie jak lista hostów będących w jego posiadaniu. Ponadto klasa ta jest przystosowana do pobierania pliku w postaci jedno-kilobajtowych segmentów i do kompletowania ich w różnej kolejności.
W momencie pobierania zasobu przez sieć wiele wątków pobiera współbieżnie różne segmenty zasobu. Każdy z wątków ma dostęp do obiektu klasy *CompleteResource*. Klasa ta przechowuje bitsety niosące informacje o tym, które segmenty są aktualnie pobierane, które są już pobrane, a których jeszcze nie pobrano. Wątki pobierające modyfikują te bitsety poprzez odpowiednie metody klasa zapewnia synchronizację dostępu do bitsetów.

5.3.2 ResourceDatabase

ResourceDatabase można przybliżyć jako graf dwudzielny którego jedna z klas dwudzielności to hosty a druga pliki posiadane przez odpowiednie hosty. Szczególnym przypadkiem hosta jest host lokalny czyli komputer na którym działa program.

ResourceDatabase



5.4 Obsługa Sytuacji Wyjatkowych

1. *Opuszczenie przez węzeł sieci :*
   po nie otrzymaniu od któregoś z węzłów sieci rozgłoszenia przez kilka okresów rozgłoszeń informacja o posiadaniu przez niego plików jest usuwana z systemu. Ze względu na fakt istnienia weak_ptr w kopiach hostów posiadających plik, informacja o jego istnieniu nie jest usuwana z bazy danych
2. *Opóźnione działanie węzła sieci :*
   Dla każdego z węzłów prowadzone jest zliczanie prekroczeń ustalonego czasu przy odpowiedziach na zapytania. Po odczekaniu pewnego czasu [ban time] próbujemy ponownie, jesli węzeł znów odpowiada z opóźnieniem nie pobieramy od niego segmentów. jeśli wszystkie węzły są opóźnione kończymy pobieranie.
3. *Błąd przy odbiorze danych przez TCP :*
   wątek pobierający od danego Hosta zwalnia segment oraz kończy pracę
4. *Gubienie rozgłaszanych pakietów:*
   Udp rozgłasza pakiety fragmentami. Fragmenty są mniejsze niż 2*576 oktetów co oznacza że w przypadku działania w sieci przewodowej ryzyko zgubienia pakiety będzie minimalne [mtu = 1500 oktetów], a w przypadku sieci bezprzewodowej przynajmniej co któryś pakiet dotrze w całości. W przypadku notorycznego gubienia pakietów węzeł będzie niewidoczny dla pozostałych.

5.5 Testy

Aplikacja została sprawdzona pod kątek działania w sieci wifi na2 osobnych maszynach

1. Sprawdzono poprawność rozgłoszeń, obserwując ruch wiresharkiem.
2. Sprawdzono poprawność rozgłoszeń w programie.
3. Sprawdzono działanie unieważnień w programie.
4. Sprawdzono doświadczalnie na przykładzie kilkumegabajtowego pliku zdolność programu do wypełniania głównego zadania

Part II

Dokumentacja kodu źródłowego - *English*

Chapter 6

Class Index

6.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 7

Class Documentation

7.1 simpleP2P::CLI Class Reference

Public Member Functions

- CLI (ResourceDatabase &res_db_, LoggingModule &Logger_, boost::asio::io_service &io_service↩ _, FileManager &fm_, Host &localhost_, Printer &printer_, UdpModule &udp)

  Construct a new CLI object.

- std::thread init ()

  Method initiating CLI in the current thread.

7.1.1 Constructor & Destructor Documentation

7.1.1.1 CLI()

```
simpleP2P::CLI::CLI (
            ResourceDatabase & res_db_,
            LoggingModule & Logger_,
            boost::asio::io_service & io_service_,
            FileManager & fm_,
            Host & localhost_,
            Printer & printer_,
            UdpModule & udp )
```

Construct a new CLI object.

Parameters

*res_db_*	
*Logger_*	
*io_↩ service_*	
*fm_*	

Parameters

*localhost_*	
*printer_*	

7.1.2 Member Function Documentation

7.1.2.1 init()

```
std::thread simpleP2P::CLI::init ( )
```

Method initiating CLI in the current thread.

Returns

CLI std::thread

The documentation for this class was generated from the following files:

- include/CLI.h
- src/CLI.cpp

7.2 simpleP2P::CLICommand Class Reference

Public Member Functions

- **CLICommand** (std::string, std::string, std::function< Int32(const std::string &)>)
- void **operator()** (const std::string &) const
- std::string **getName** () const
- std::string **getDesc** () const

The documentation for this class was generated from the following files:

- include/CLICommand.h
- src/CLICommand.cpp

7.3 simpleP2P::CompleteResource Class Reference

Class representing a complete resource (resource and full its data)

```
#include <CompleteResource.h>
```

Public Member Functions

- CompleteResource (std::shared_ptr< Resource > resource_c)

  Construct a new Complete Resource object based on base Resource object.
- Uint8 * get_data ()

  Get the contents of the file.
- std::shared_ptr< Resource > get_resource () const

  Get the underlaying resource object.
- Segment get_segment ()

  Synchronised method returning Segment object representing the first unbusy and incomplete segment.
- void set_segment (Segment &segment)

  Synchronised method marking the given segment as completed.
- bool is_completed ()

  Synchronised method returning true if all segments have been downloaded, false otherwise.
- void unset_busy (SegmentId id)

  Synchronised method unmarking busy segment and notifying one of waithing worker threads.

7.3.1 Detailed Description

Class representing a complete resource (resource and full its data)

7.3.2 Constructor & Destructor Documentation

7.3.2.1 CompleteResource()

```
simpleP2P::CompleteResource::CompleteResource (
            std::shared_ptr< Resource > resource_c )
```

Construct a new Complete Resource object based on base Resource object.

Parameters

resource↩ _c	base resource object

7.3.3 Member Function Documentation

7.3.3.1 get_data()

```
Uint8 * simpleP2P::CompleteResource::get_data ( )
```

Get the contents of the file.

Returns

Uint8*

7.3.3.2 get_resource()

```
std::shared_ptr< Resource > simpleP2P::CompleteResource::get_resource ( ) const
```

Get the underlaying resource object.

Returns

std::shared_ptr<Resource>

7.3.3.3 get_segment()

```
Segment simpleP2P::CompleteResource::get_segment ( )
```

Synchronised method returning Segment object representing the first unbusy and incomplete segment.

If resource downlaoding has been completed, special Segment object with id set to NO_SEGMENT_ID is returned. If there is no appropriate segment found and downloading is not completed, method suspends calling worker thread pending some segment release.

Returns

Segment

7.3.3.4 is_completed()

```
bool simpleP2P::CompleteResource::is_completed ( )
```

Synchronised method returning true if all segments have been downloaded, false otherwise.

Returns

true if downloading is completed
false otherwise

7.3.3.5 set_segment()

```
void simpleP2P::CompleteResource::set_segment (
            Segment & segment )
```

Synchronised method marking the given segment as completed.

If all segments have been downloaded, method notifies all waiting worker threads so that they can join.

Parameters

segment	

7.3.3.6 unset_busy()

```
void simpleP2P::CompleteResource::unset_busy (
            SegmentId id )
```

Synchronised method unmarking busy segment and notifying one of waithing worker threads.

Parameters

id	

The documentation for this class was generated from the following files:

- include/CompleteResource.h
- src/CompleteResource.cpp

7.4 simpleP2P::DownloadService Class Reference

Used to download resource.

`#include <DownloadService.h>`

Public Member Functions

- DownloadService (LoggingModule &logging_module_c, boost::asio::io_service &io_service_c, FileManager &file_manager_c, ResourceDatabase &resource_database_c, std::shared_ptr< Resource > resource_c)

  Construct a new Download Service object.

- void init ()

  Method initiating downloading in the current thread.

- std::thread init_thread ()

  Method initiating downloading in a new thread.

7.4.1 Detailed Description

Used to download resource.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 DownloadService()

```
simpleP2P::DownloadService::DownloadService (
            LoggingModule & logging_module_c,
            boost::asio::io_service & io_service_c,
            FileManager & file_manager_c,
            ResourceDatabase & resource_database_c,
            std::shared_ptr< Resource > resource_c )
```

Construct a new Download Service object.

Parameters

logging_module_c	
io_service_c	
file_manager_c	
resource_database_c	
resource_c	

7.4.3 Member Function Documentation

7.4.3.1 init_thread()

```
std::thread simpleP2P::DownloadService::init_thread ( )
```

Method initiating downloading in a new thread.

Returns

std::thread

The documentation for this class was generated from the following files:

- include/DownloadService.h
- src/DownloadService.cpp

7.5 simpleP2P::DownloadWorker Class Reference

Class used to download resource segments from one host.

```
#include <DownloadWorker.h>
```

Public Member Functions

- DownloadWorker (LoggingModule &logging_module_c, boost::asio::io_service &io_service_c, std::shared_ptr< Host > host_c, std::shared_ptr< CompleteResource > complete_resource_c)

  Construct a new Download Worker object.

- std::thread init ()

  Method initiating connection to host and downloading resource segments in a new thread.

- void check_timeout ()

  Method checking worker timeout and taking appropriate actions if it exeeds:

- void close ()

  ++

- bool is_closed ()

  Synchronised method checking if worker is closed.

- bool is_unavailable ()

  Synchronised method checking if worker is available (not closed and associated host not retarded)

7.5.1 Detailed Description

Class used to download resource segments from one host.

7.5.2 Constructor & Destructor Documentation

7.5.2.1 DownloadWorker()

```
simpleP2P::DownloadWorker::DownloadWorker (
            LoggingModule & logging_module_c,
            boost::asio::io_service & io_service_c,
            std::shared_ptr< Host > host_c,
            std::shared_ptr< CompleteResource > complete_resource_c )
```

Construct a new Download Worker object.

Parameters

logging_module_c	
io_service_c	
host_c	
complete_resource_c	

7.5.3 Member Function Documentation

7.5.3.1 check_timeout()

```
void simpleP2P::DownloadWorker::check_timeout ( )
```

Method checking worker timeout and taking appropriate actions if it exeeds:

- the host timeout counter is increased, what may cause banning host for certain time,
- currently downloaded segment is marked as unbusy and may be downloaded by another worker.

7.5.3.2 close()

```
void simpleP2P::DownloadWorker::close ( )
```

++

Synchronised method closing worker politely.

7.5.3.3 init()

```
std::thread simpleP2P::DownloadWorker::init ( )
```

Method initiating connection to host and downloading resource segments in a new thread.

Returns

std::thread

7.5.3.4 is_unavailable()

```
bool simpleP2P::DownloadWorker::is_unavailable ( )
```

Synchronised method checking if worker is available (not closed and associated host not retarded)

Returns

true
false

The documentation for this class was generated from the following files:

- include/DownloadWorker.h
- src/DownloadWorker.cpp

7.6 simpleP2P::FileManager Class Reference

Handles read/write to the files on disc.

`#include <FileManager.h>`

Public Member Functions

- **FileManager** (LoggingModule &lm)
- bool get_segment (const std::string file_name, const Uint16 segment, Uint8 *result, const std::size_t requested_segment_size)

  Buffers the specificated segment of the specificated file in the char array.
- void store_resource (std::shared_ptr< CompleteResource > complete_resource)

  Stores the file contents in the physical file on disc.
- bool read_lock (const std::string file_name)

  Closes the file and unlocks it for writing.
- void read_unlock (const std::string file_name)

  Closes a file which was opened for reading and enables other threads to write to it.

7.6.1 Detailed Description

Handles read/write to the files on disc.

An API which provides:

- buffering contents of a requested segment of a specificated local file,
- storing a complete, downloaded file physically on the local disc. Ensures synchronization of those operations.

7.6.2 Member Function Documentation

7.6.2.1 get_segment()

```
bool simpleP2P::FileManager::get_segment (
            const std::string file_name,
            const Uint16 segment,
            Uint8 * result,
            const std::size_t requested_segment_size )
```

Buffers the specificated segment of the specificated file in the char array.

Warning

Calling this function MUST be preceded by a successful call to read_lock() on the file specified in resource_header equal to the one carried by the SegmentRequest.

Parameters

request	Specifies file and its segment to buffer.
result	The array to buffer the segment in. Its size must be greater or equal to the 3rd parameter!
requested_segment_size	Size of the segment to buffer.

Returns

If the segment has been buffered successfully.

7.6.2.2 read_lock()

```
bool simpleP2P::FileManager::read_lock (
            const std::string file_name )
```

Closes the file and unlocks it for writing.

Parameters

resource_header	The file to be closed and unlocked. Opens a file for reading and prevents other threads from overwriting it until it's closed.
resource_header	The file to be opened and read-locked.

Returns

If the file has been opened successfully.

7.6.2.3 read_unlock()

```
void simpleP2P::FileManager::read_unlock (
            const std::string file_name )
```

Closes a file which was opened for reading and enables other threads to write to it.

Warning

This function MUST be called after the reading has completed, so that the file can be overwritten in future.

Parameters

7.6.2.4 store_resource()

```
void simpleP2P::FileManager::store_resource (
            std::shared_ptr< CompleteResource > complete_resource )
```

Stores the file contents in the physical file on disc.

Parameters

resource	File to store on the disc. The data will not be interpreted, so make sure it's complete and ready to store.

The documentation for this class was generated from the following files:

- include/FileManager.h
- src/FileManager.cpp

7.7 simpleP2P::Host Class Reference

Class contains node information and points to files it possess.

`#include <Host.h>`

Public Member Functions

- Host (boost::asio::ip::address ip)

  Constructor.
- **Host** (const Host &other)
- bool has_resource (const Resource &res)

  Determines if host has resource.
- bool operator== (const Host &other) const

  Operator == checks host_ip for equality.
- bool operator!= (const Host &other) const

  Operator != checks host_ip for equality.
- boost::asio::ip::tcp::endpoint get_endpoint () const

  Method returning boost tcp endpoint (ip address and port) of the host.
- bool is_retarded ()

  Method returning true if program considers a given host to be retarded.
- void increase_timeout_counter ()

  Method increasing a host timeout counter.
- std::chrono::system_clock::time_point get_ban_time_point () const

  Method returning a time point to which a host is considered to be retarded.
- std::vector< std::weak_ptr< Resource > > **get_possesed** () const
- void remove_resource (const Resource &res)

removes Resource res

- void add_resource (const std::shared_ptr< Resource > &res)

  adds Resource res

- Uint16 get_missed_updates () const

  Getter for no of missed updates.

- void reset_missed_updates ()

  reset missed updates

- void inc_missed_updates ()

  increments missed updates couter

- void remove_all_resources ()

  clears possesd resources

7.7.1 Detailed Description

Class contains node information and points to files it possess.

7.7.2 Constructor & Destructor Documentation

7.7.2.1 Host()

```
simpleP2P::Host::Host (
            boost::asio::ip::address ip )
```

Constructor.

Parameters

ip	Ip of the Host

7.7.3 Member Function Documentation

7.7.3.1 add_resource()

```
void simpleP2P::Host::add_resource (
            const std::shared_ptr< Resource > & res )
```

adds Resource res

Parameters

res	Resource to add

7.7.3.2 get_ban_time_point()

```
std::chrono::system_clock::time_point simpleP2P::Host::get_ban_time_point ( ) const
```

Method returning a time point to which a host is considered to be retarded.

Returns

std::chrono::system_clock::time_point

7.7.3.3 get_endpoint()

```
boost::asio::ip::tcp::endpoint simpleP2P::Host::get_endpoint ( ) const
```

Method returning boost tcp endpoint (ip address and port) of the host.

Returns

boost::asio::ip::tcp::endpoint

7.7.3.4 get_missed_updates()

```
uint16_t simpleP2P::Host::get_missed_updates ( ) const
```

Getter for no of missed updates.

Returns

missed updates count

7.7.3.5 has_resource()

```
bool simpleP2P::Host::has_resource (
            const Resource & res )
```

Determines if host has resource.

Parameters

res	Resource to be checked

Returns

true if Host has Resource res

7.7.3.6 increase_timeout_counter()

```
void simpleP2P::Host::increase_timeout_counter ( )
```

Method increasing a host timeout counter.

After exeeding timeout limit, a host is considered to be retarded up to some point in time.

7.7.3.7 is_retarded()

```
bool simpleP2P::Host::is_retarded ( )
```

Method returning true if program considers a given host to be retarded.

Returns

true if a host is retarded
false otherwise

7.7.3.8 operator"!=()

```
bool simpleP2P::Host::operator!= (
            const Host & other ) const
```

Operator != checks host_ip for equality.

Parameters

other	other

Returns

true if not equal

7.7.3.9 operator==()

```
bool simpleP2P::Host::operator== (
            const Host & other ) const
```

Operator == checks host_ip for equality.

Parameters

other	other

Returns

true if equal

7.7.3.10 remove_resource()

```
void simpleP2P::Host::remove_resource (
            const Resource & res )
```

removes Resource res

Parameters

res	Resource to remove

The documentation for this class was generated from the following files:

- include/Host.h
- src/Host.cpp

7.8 simpleP2P::LoggingModule Class Reference

class Providing logging support based on text logs

`#include <LoggingModule.h>`

Public Member Functions

- LoggingModule (std::ostream &output_c=std::cerr)

  Constructor.
- std::thread init ()

  Init thread for the logging module.
- void add_log_line (std::string line, const std::time_t &time)

  Synchronised method for logging output.

7.8.1 Detailed Description

class Providing logging support based on text logs

7.8.2 Constructor & Destructor Documentation

7.8.2.1 LoggingModule()

```
simpleP2P::LoggingModule::LoggingModule (
            std::ostream & output_c = std::cerr )
```

Constructor.

Parameters

output_c	Output stream for the logs

Note

if output stream is a file you must explicitly close it

7.8.3 Member Function Documentation

7.8.3.1 add_log_line()

```
void simpleP2P::LoggingModule::add_log_line (
            std::string line,
            const std::time_t & time )
```

Synchronised method for logging output.

Parameters

line	

The documentation for this class was generated from the following files:

- include/LoggingModule.h
- src/LoggingModule.cpp

7.9 simpleP2P::Printer Class Reference

class printing outputs, using queue in order to avoid races

```
#include <Printer.h>
```

Public Member Functions

- Printer (std::ostream &output_c=std::cout)

  Constructor for the printer.
- std::thread init ()

Init methods run worker in thread and returns it.

- void print (std::string line)

  Synchronised method for printing output.

7.9.1 Detailed Description

class printing outputs, using queue in order to avoid races

7.9.2 Constructor & Destructor Documentation

7.9.2.1 Printer()

```
simpleP2P::Printer::Printer (
            std::ostream & output_c = std::cout )
```

Constructor for the printer.

Parameters

output_c	Output stream for couts

7.9.3 Member Function Documentation

7.9.3.1 init()

```
std::thread simpleP2P::Printer::init ( )
```

Init methods run worker in thread and returns it.

Returns

printing std::thread

7.9.3.2 print()

```
void simpleP2P::Printer::print (
            std::string line )
```

Synchronised method for printing output.

Parameters

line	

The documentation for this class was generated from the following files:

- include/Printer.h
- src/Printer.cpp

7.10 simpleP2P::RequestServer Class Reference

Asynchronous TCP server.

`#include <RequestServer.h>`

Public Member Functions

- RequestServer (boost::asio::io_service &io_service, Uint16 port, FileManager &fm, LoggingModule &lm)

  Constructor allows setting the parameters for the connnetion acceptor.
- std::thread init ()

  Turns on the listening and accepting connections and returns the thread in which it works.

7.10.1 Detailed Description

Asynchronous TCP server.

It accepts connections asynchronously and for each of them creates a worker object to handle it.

7.10.2 Constructor & Destructor Documentation

7.10.2.1 RequestServer()

```
simpleP2P::RequestServer::RequestServer (
            boost::asio::io_service & io_service,
            Uint16 port,
            FileManager & fm,
            LoggingModule & lm )
```

Constructor allows setting the parameters for the connnetion acceptor.

Parameters

io_service	boost::asio::io_service for the acceptor.
port	Port for the acceptor to listen on.

The documentation for this class was generated from the following files:

- include/RequestServer.h
- src/RequestServer.cpp

7.11 simpleP2P::RequestServerModule Class Reference

Module of the TCP server receiving file requests and sending the requested files' segments.

`#include <RequestServerModule.h>`

Public Member Functions

- RequestServerModule (Uint16 port_, FileManager &fm, LoggingModule &lm)

  Constructor, allows setting the port for the server.
- std::thread init ()

  Returns the thread object for the module.

7.11.1 Detailed Description

Module of the TCP server receiving file requests and sending the requested files' segments.

7.11.2 Member Function Documentation

7.11.2.1 init()

`std::thread simpleP2P::RequestServerModule::init ( )`

Returns the thread object for the module.

Starts the server and returns the thread in which the server works.

The documentation for this class was generated from the following files:

- include/RequestServerModule.h
- src/RequestServerModule.cpp

7.12 simpleP2P::RequestWorker Class Reference

TCP connection handler, created by the TCP server.

`#include <RequestWorker.h>`

Public Member Functions

- RequestWorker (boost::asio::io_service &io_service, FileManager &fm, LoggingModule &lm)

  Constructor allows setting the socket on which the connection is established.
- void start ()

  Start handling the request.
- tcp::socket & socket ()

  Get socket.

7.12.1 Detailed Description

TCP connection handler, created by the TCP server.

Receives the file request, buffers requested segments and sends them to the client.

7.12.2 Constructor & Destructor Documentation

7.12.2.1 RequestWorker()

```
simpleP2P::RequestWorker::RequestWorker (
            boost::asio::io_service & io_service,
            FileManager & fm,
            LoggingModule & lm )
```

Constructor allows setting the socket on which the connection is established.

Parameters

io_service	io_service in which the server runs
fm	FileManager for accessing requested files
lm	LoggingModule for logging events

The documentation for this class was generated from the following files:

- include/RequestWorker.h
- src/RequestWorker.cpp

7.13 simpleP2P::Resource Class Reference

Forward declaration.

```
#include <Resource.h>
```

Public Member Functions

- Resource (std::string name, Uint64 size, std::string path="./")

  Constructor.
- Resource (std::vector< Uint8 > resource_header)

  Constructor makes resource from header.
- **Resource** (const Resource &other)
- std::vector< Uint8 > generate_resource_header ()

  Generates Resource header.
- bool has_host (Host host)

  Determines if resource is possesed by Host.
- Uint16 calc_segments_count () const

  Calculates and returns segment count.
- void set_revoked ()

  Function used to set invalidated flag.
- bool **isInvalidated** ()
- Uint64 getSize () const

  Getter for file size.
- const std::string & getName () const

  Getter for file name.
- const std::string & getPath () const

  Getter for file path.
- bool operator== (const Resource &other) const

  Operator == checks file size and name for equality.
- bool operator!= (const Resource &other) const

  Operator != checks file size and name for equality.
- std::vector< std::weak_ptr< Host > > **get_hosts** ()
- void **remove_host** (const Host &host)
- void **add_host** (const std::shared_ptr< Host > &host)

7.13.1 Detailed Description

Forward declaration.

Class contains file information and points to nodes with file possesion

7.13.2 Constructor & Destructor Documentation

7.13.2.1 Resource() [1/2]

```
simpleP2P::Resource::Resource (
            std::string name,
            Uint64 size,
            std::string path = "./" )
```

Constructor.

Parameters

name	filename
size	filesize
path	filepath, default is "./"

7.13.2.2 Resource() [2/2]

```
simpleP2P::Resource::Resource (
            std::vector< Uint8 > resource_header )
```

Constructor makes resource from header.

Parameters

resource_header	Resource header

7.13.3 Member Function Documentation

7.13.3.1 calc_segments_count()

```
Uint16 simpleP2P::Resource::calc_segments_count ( ) const  [inline]
```

Calculates and returns segment count.

Returns

segment count

7.13.3.2 generate_resource_header()

```
std::vector< Uint8 > simpleP2P::Resource::generate_resource_header ( )
```

Generates Resource header.

Returns

Resource header

7.13.3.3 getName()

```
const std::string & simpleP2P::Resource::getName ( ) const
```

Getter for file name.

Returns

file name

7.13.3.4 getPath()

```
const std::string & simpleP2P::Resource::getPath ( ) const
```

Getter for file path.

Returns

file path

7.13.3.5 getSize()

```
Uint64 simpleP2P::Resource::getSize ( ) const
```

Getter for file size.

Returns

file size

7.13.3.6 has_host()

```
bool simpleP2P::Resource::has_host (
            simpleP2P::Host host )
```

Determines if resource is possesed by Host.

Parameters

host	Host

Returns

true if resource is possessed by host

7.13.3.7 operator"!=()

```
bool simpleP2P::Resource::operator!= (
            const Resource & other ) const
```

Operator != checks file size and name for equality.

Parameters

other	other

Returns

true if not equal

7.13.3.8 operator==()

```
bool simpleP2P::Resource::operator== (
            const Resource & other ) const
```

Operator == checks file size and name for equality.

Parameters

other	other

Returns

true if equal

7.13.3.9 set_revoked()

```
void simpleP2P::Resource::set_revoked ( )
```

Function used to set invalidated flag.

To allow references on resource outside database to gather information about revoke

The documentation for this class was generated from the following files:

- include/Resource.h
- src/Resource.cpp

7.14 simpleP2P::ResourceDatabase Class Reference

Class holding information about files in network and on localhost.

```
#include <ResourceDatabase.h>
```

Public Member Functions

- ResourceDatabase (Host localhost)

  Constructor.
- void add_file (const Resource &res, const Host &host)

  Adds connection between file and resource, adn creates them if they do not exist.
- void remove_file (const Resource &res, const Host &host)

  Removes connection between file and resource.
- void update_host (const Host &host)

  Updates the list of resources aviable from host Triggered after receive of full Beacon Packet.
- void revoke_resource (const Resource &resource)

  Revokes resource and disconnects it from Hosts in database and database itself Resource will still point to Hosts that possess it.
- void add_file (const Resource &res)

  same as add_file(Resource, Host) but host is localhost
- void remove_file (const Resource &res)

  same as remove_file(Resource, Host) but host is localhost
- std::shared_ptr< Resource > who_has_file (std::vector< Uint8 > resource_header)

  Returns shared pointer to resource to allow access to information about file owners.
- std::shared_ptr< Resource > who_has_file (const Resource &res)

  Returns shared pointer to resource to allow access to information about file owners.
- std::vector< std::vector< Uint8 > > generate_database_headers ()

  Generates listing of localhost content in a header.
- std::shared_ptr< Host > get_localhost () const

  Get localhost information.
- void check_for_gone_hosts (Uint16 left_margin=LEFT_MARGIN)

  Checks if any of the host missed a couple beacon intervals as we suppose they left the network.
- std::vector< std::shared_ptr< Resource > > getResources () const

  Obtains copy of possessed resources.

7.14.1 Detailed Description

Class holding information about files in network and on localhost.

7.14.2 Constructor & Destructor Documentation

7.14.2.1 ResourceDatabase()

```
simpleP2P::ResourceDatabase::ResourceDatabase (
            Host localhost )
```

Constructor.

Parameters

localhost	localhost

7.14.3 Member Function Documentation

7.14.3.1 add_file() [1/2]

```
void simpleP2P::ResourceDatabase::add_file (
            const Resource & res,
            const Host & host )
```

Adds connection between file and resource, adn creates them if they do not exist.

Parameters

res	Resource to be added
host	Host which possess Resource res

7.14.3.2 add_file() [2/2]

```
void simpleP2P::ResourceDatabase::add_file (
            const Resource & res )
```

same as add_file(Resource, Host) but host is localhost

Parameters

res	Resource to be added

7.14.3.3 check_for_gone_hosts()

```
void simpleP2P::ResourceDatabase::check_for_gone_hosts (
            Uint16 left_margin = LEFT_MARGIN )
```

Checks if any of the host missed a couple beacon intervals as we suppose they left the network.

Parameters

left_margin	number of left margins needed for deletion of the host

7.14.3.4 generate_database_headers()

```
std::vector< std::vector< Uint8 > > simpleP2P::ResourceDatabase::generate_database↩
_headers ( )
```

Generates listing of localhost content in a header.

Returns

listing header of localhost resources

7.14.3.5 get_localhost()

```
std::shared_ptr< Host > simpleP2P::ResourceDatabase::get_localhost ( ) const
```

Get localhost information.

Returns

localhost

7.14.3.6 getResources()

```
std::vector< std::shared_ptr< Resource > > simpleP2P::ResourceDatabase::getResources
( ) const
```

Obtains copy of possessed resources.

Returns

copy of possessed resources vector

7.14.3.7 remove_file() [1/2]

```
void simpleP2P::ResourceDatabase::remove_file (
            const Resource & res,
            const Host & host )
```

Removes connection between file and resource.

Parameters

res	Resource to be removed from host list
host	Host which resource will be removed

7.14.3.8 remove_file() [2/2]

```
void simpleP2P::ResourceDatabase::remove_file (
            const Resource & res )
```

same as remove_file(Resource, Host) but host is localhost

Parameters

res	Resource to be removed from localhost list

7.14.3.9 revoke_resource()

```
void simpleP2P::ResourceDatabase::revoke_resource (
            const Resource & resource )
```

Revokes resource and disconnects it from Hosts in database and database itself Resource will still point to Hosts that possess it.

Parameters

resource	Resource to be revoked

7.14.3.10 update_host()

```
void simpleP2P::ResourceDatabase::update_host (
            const Host & host )
```

Updates the list of resources aviable from host Triggered after receive of full Beacon Packet.

Parameters

host	Host and possesed resources in a struct

7.14.3.11 who_has_file() [1/2]

```
shared_ptr< Resource > simpleP2P::ResourceDatabase::who_has_file (
            std::vector< Uint8 > resource_header ) [inline]
```

Returns shared pointer to resource to allow access to information about file owners.

Parameters

res	Resource about which information is gathered

Returns

shared pointer to res

7.14.3.12 who_has_file() [2/2]

```
shared_ptr< Resource > simpleP2P::ResourceDatabase::who_has_file (
            const Resource & res )
```

Returns shared pointer to resource to allow access to information about file owners.

Parameters

res	Resource about which information is gathered

Returns

shared pointer to res

The documentation for this class was generated from the following files:

- include/ResourceDatabase.h
- src/ResourceDatabase.cpp

7.15 simpleP2P::Segment Class Reference

Class representing a segment of a resource.

`#include <Segment.h>`

Public Member Functions

- Segment (SegmentId id_c, Uint8 *data_c)

  Construct a new Segment object.
- SegmentId get_id () const

  Get the segment id.
- Uint8 * get_data_ptr () const

  Get the pointer to the segment data.
- std::vector< Uint8 > serialize_id ()

Static Public Member Functions

- static Segment no_segment_left ()

  Special method returning Segment object with id set to NO_SEGMENT_ID indicating no segment to download in complete resource object.

Static Public Attributes

- static const SegmentId NO_SEGMENT_ID = static_cast<SegmentId>(-1)

  Special id number indicating no segment to download in complete resource object.

7.15.1 Detailed Description

Class representing a segment of a resource.

It contains an id of the segment and a pointer to the segment data in complete resource object.

7.15.2 Constructor & Destructor Documentation

7.15.2.1 Segment()

```
simpleP2P::Segment::Segment (
            SegmentId id_c,
            Uint8 * data_c )
```

Construct a new Segment object.

Parameters

id_c	segment id
data_c	pointer to the segment data in complete resource object

7.15.3 Member Function Documentation

7.15.3.1 get_data_ptr()

```
Uint8 * simpleP2P::Segment::get_data_ptr ( ) const
```

Get the pointer to the segment data.

Returns

Uint8*

7.15.3.2 get_id()

```
SegmentId simpleP2P::Segment::get_id ( ) const
```

Get the segment id.

Returns

SegmentId

7.15.3.3 no_segment_left()

```
Segment simpleP2P::Segment::no_segment_left ( )  [static]
```

Special method returning Segment object with id set to NO_SEGMENT_ID indicating no segment to download in complete resource object.

Returns

Segment

7.15.3.4 serialize_id()

```
std::vector<Uint8> simpleP2P::Segment::serialize_id ( )
```

Returns

std::vector<Uint8>

The documentation for this class was generated from the following files:

- include/Segment.h
- src/Segment.cpp

7.16 simpleP2P::UdpClient Class Reference

class UDP Client to handle all outgoing packets

```
#include <UdpClient.h>
```

Inherits enable_shared_from_this< UdpClient >.

Public Member Functions

- UdpClient (boost::asio::io_service &io_service, ResourceDatabase &database, LoggingModule &logger, const boost::asio::ip::address &broadcast_address, Uint16 broadcast_port, Uint32 timeout=5 ∗60)

  Constructor of UDP Client.

- ∼UdpClient ()

  Destructor closes socket.

- void revoke_file (Resource resource)

  Constructs revoke header sends it.

7.16.1 Detailed Description

class UDP Client to handle all outgoing packets

7.16.2 Constructor & Destructor Documentation

7.16.2.1 UdpClient()

```
simpleP2P::UdpClient::UdpClient (
            boost::asio::io_service & io_service,
            ResourceDatabase & database,
            LoggingModule & logger,
            const boost::asio::ip::address & broadcast_address,
            Uint16 broadcast_port,
            Uint32 timeout = 5 * 60 )
```

Constructor of UDP Client.

Parameters

io_service	asio Io Service
database	Database
logger	Logger
broadcast_address	address on which packets will be sent
broadcast_port	port on which packets will be sent
timeout	beacon interval

7.16.3 Member Function Documentation

7.16.3.1 revoke_file()

```
void simpleP2P::UdpClient::revoke_file (
            simpleP2P::Resource resource )
```

Constructs revoke header sends it.

Parameters

resource	Resource to be revoked

The documentation for this class was generated from the following files:

- include/UdpClient.h
- src/UdpClient.cpp

7.17 simpleP2P::UdpModule Class Reference

Class containing all UDP related resources and logic.

`#include <UdpModule.h>`

Public Member Functions

- UdpModule (ResourceDatabase &database_c, LoggingModule &logger_c, boost::asio::ip::address broadcast_address, Uint16 port, Uint32 beacon_interval)

  Constructor.
- std::thread init ()

  Init methods run worker in thread and returns it.
- bool revoke_file (const Resource &resource)

  Sends revoke datagram.

7.17.1 Detailed Description

Class containing all UDP related resources and logic.

7.17.2 Constructor & Destructor Documentation

7.17.2.1 UdpModule()

```
simpleP2P::UdpModule::UdpModule (
            ResourceDatabase & database_c,
            LoggingModule & logger_c,
            boost::asio::ip::address broadcast_address,
```

```
            Uint16 port,
            Uint32 beacon_interval )
```

Constructor.

Parameters

broadcast_address	address on which packets will be sent
port	port on which packets will be sent
beacon_interval	beacon interval

7.17.3 Member Function Documentation

7.17.3.1 init()

```
std::thread simpleP2P::UdpModule::init ( )
```

Init methods run worker in thread and returns it.

Returns

logging thread

7.17.3.2 revoke_file()

```
bool simpleP2P::UdpModule::revoke_file (
            const Resource & resource )
```

Sends revoke datagram.

Parameters

resource	Resource t obe revoked

Returns

true on success

The documentation for this class was generated from the following files:

- include/UdpModule.h
- src/UdpModule.cpp

7.18 simpleP2P::UdpServer Class Reference

class UDP Server to handle all incoming packets

```
#include <UdpServer.h>
```

Inherits enable_shared_from_this< UdpServer >.

Public Member Functions

- UdpServer (boost::asio::io_service &io_service, ResourceDatabase &database, LoggingModule &logger, const boost::asio::ip::address &broadcast_address, Uint16 broadcast_port)

  Constructor of UDP Server.

- ~UdpServer ()

  Destructor closes socket.

7.18.1 Detailed Description

class UDP Server to handle all incoming packets

7.18.2 Constructor & Destructor Documentation

7.18.2.1 UdpServer()

```
simpleP2P::UdpServer::UdpServer (
            boost::asio::io_service & io_service,
            ResourceDatabase & database,
            LoggingModule & logger,
            const boost::asio::ip::address & broadcast_address,
            Uint16 broadcast_port )
```

Constructor of UDP Server.

Parameters

io_service	asio Io Service
database	Database
logger	Logger
broadcast_address	address on which Server will listen
broadcast_port	port on which Server will listen

The documentation for this class was generated from the following files:

- include/UdpServer.h
- src/UdpServer.cpp