

# TIN - Projekt wstępny

**Skład zespołu:** Julita Ołtusek, Filip Rak, Michał Derdak, Maciej Wiraszka  
**Prowadzący projekt:** dr inż. Grzegorz Blinowski  
**Data przekazania:** 25.04.2017

## Treść zadania:

Program obsługujący prosty protokół Peer-to-Peer (P2P)  
(wariant 2.)

### 1. Interpretacja treści zadania

Program uruchamiany będzie z terminala. Jego poszczególne funkcje będą dostępne poprzez interfejs tekstowy, uruchamiane będą asynchronicznymi komendami użytkownika. Możliwe będzie działanie kilku funkcji programu równolegle (używając wielu wątków). Zasoby lokalne przechowywane będą w formie plików na lokalnym dysku hosta.

Program przechowywał będzie w pamięci dwie listy zasobów: listę wszystkich oryginałów dostępnych w sieci, listę zasobów które w danej chwili posiada host z informacjami o ich właścicielach, rozmiarze, czasie wprowadzenia i o tym czy dany zasób jest aktualnie zablokowany / unieważniony / dostępny.

Po uruchomieniu programu wysyła komunikat z prośbą o przesłanie aktualnej listy oryginałów w sieci i porównując ją ze stanem swojej listy dokonuje jej synchronizacji.

Przed każdą operacją na zasobach (pobranie, dodanie, usunięcie) host synchronizować będzie najpierw listę swoich zasobów.

Po wprowadzeniu zasobu program automatycznie wyśle rozgłaszany komunikat o dostępności tego zasobu. Analogicznie stanie się podczas usuwania, blokowania czy też unieważniania zasobu - informacja o tym zdarzeniu zostanie rozpropagowana odpowiednim pakietem do wszystkich hostów w sieci. Po każdej zmianie stanu zasobu na dowolnym hoscie, rozgłosi on ten fakt i reszta hostów zsynchronizuje swoje listy zasobów.

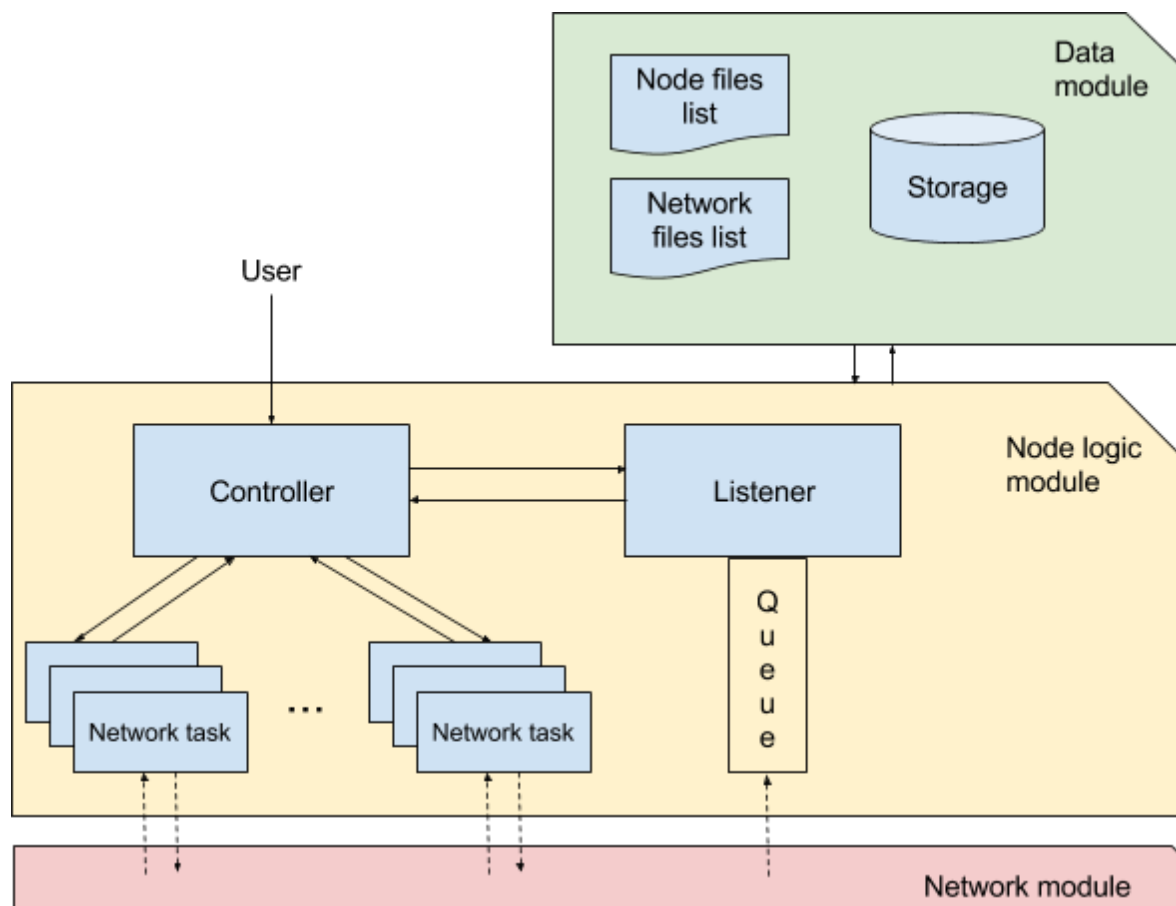
Importowanie (pobieranie) zasobu będzie odbywało się zawsze od jak największej liczby hostów (które posiadają dany zasób) jednocześnie (na kilku *socketach*, każdy dla innego nadawcy), od każdego z hostów inną część zasobu, aby maksymalnie przyspieszyć pobieranie zasobu. Po pobraniu wszystkich części fragmenty zostają połączone w jeden plik.

Do przesyłania wszelkiego rodzaju komunikatów skorzystamy z protokołu UDP oraz formatu JSON, sama transmisja zawartości zasobów odbywać się będzie natomiast w trybie binarnym, z wykorzystaniem protokołu TCP. Program będzie implementował nasz własny protokół bazując na wyżej wymienionych, oraz na zdefiniowanych w dalszej części sprawozdania zasadach i algorytmach interakcji hostów.

## 2. Planowany podział na moduły

Główne moduły programu (zobrazowane schematem poniżej):

- moduł danych *"Data"* (przechowywanie i obsługa lokalnych zasobów: głównie odczyt z dysku, zapis na dysk, zarządzanie listami zasobów)
- moduł sieciowy *"Network"* (obsługa gniazd BSD, połączeń itp. cała warstwa sieciowa)
- moduł kontrolera *"Node logic"* (sterowanie programem, obsługa komend użytkownika, tworzenie wątków, obsługa odebranych przez *Network* pakietów, wysyłanie własnych pakietów)



## 3. Krótki opis funkcjonalny

Planowane funkcje programu:

- **Wprowadzenie nowego zasobu** - Program umożliwia załadowanie dowolnego zasobu do lokalnego węzła poprzez odpowiednie polecenie oraz wskazanie lokalnej ścieżki do pliku (jeśli na liście istniejących w sieci plików taki plik nie istnieje). Listy plików zostają zaktualizowane, komunikat o wprowadzeniu nowego zasobu jest rozpropagowywany w sieci.

- **Usunięcie zasobu** - polecenie usuwa plik z lokalnego zbioru oraz wysyła komunikat o usunięciu do wszystkich węzłów. W pozostałych węzłach taki zasób zostanie usunięty po pewnym czasie.
- **Unieważnienie zasobu** - właściciel zasobu wysyła do wszystkich węzłów komunikat o unieważnieniu. Rozpoczęte operacje importowania danego zasobu zostaną dokończone, jednak nie można rozpocząć nowych.
- **Blokowanie zasobu** - właściciel zasobu może czasowo go zablokować, transfery takiego zasobu zostają przerwane, żaden z posiadaczy kopii lub oryginału zasobu nie może go udostępnić aż do chwili odblokowania zasobu przez właściciela.
- **Odblokowanie zasobu** - właściciel zasobu odblokowuje uprzednio zablokowany zasób.
- **Wyświetlenie listy zasobów** - użytkownik wywołując odpowiednią komendę pobiera listę nazw zasobów dostępnych w sieci P2P, wraz z ich wielkościami i właścicielami. Można także wyspecyfikować konkretny węzeł z którego te informacje zostaną pobrane.
- **Wyszukanie zasobu** - możliwe jest po nazwie lub jej części może wyszukać interesujące go zasoby dostępne w sieci.
- **Importowanie zasobu** - użytkownik wpisuje komendę oraz nazwę pliku. System w sposób nieznany użytkownikowi wybiera optymalny sposób pobierania.

## 4. Opis i analiza stosowanych protokołów

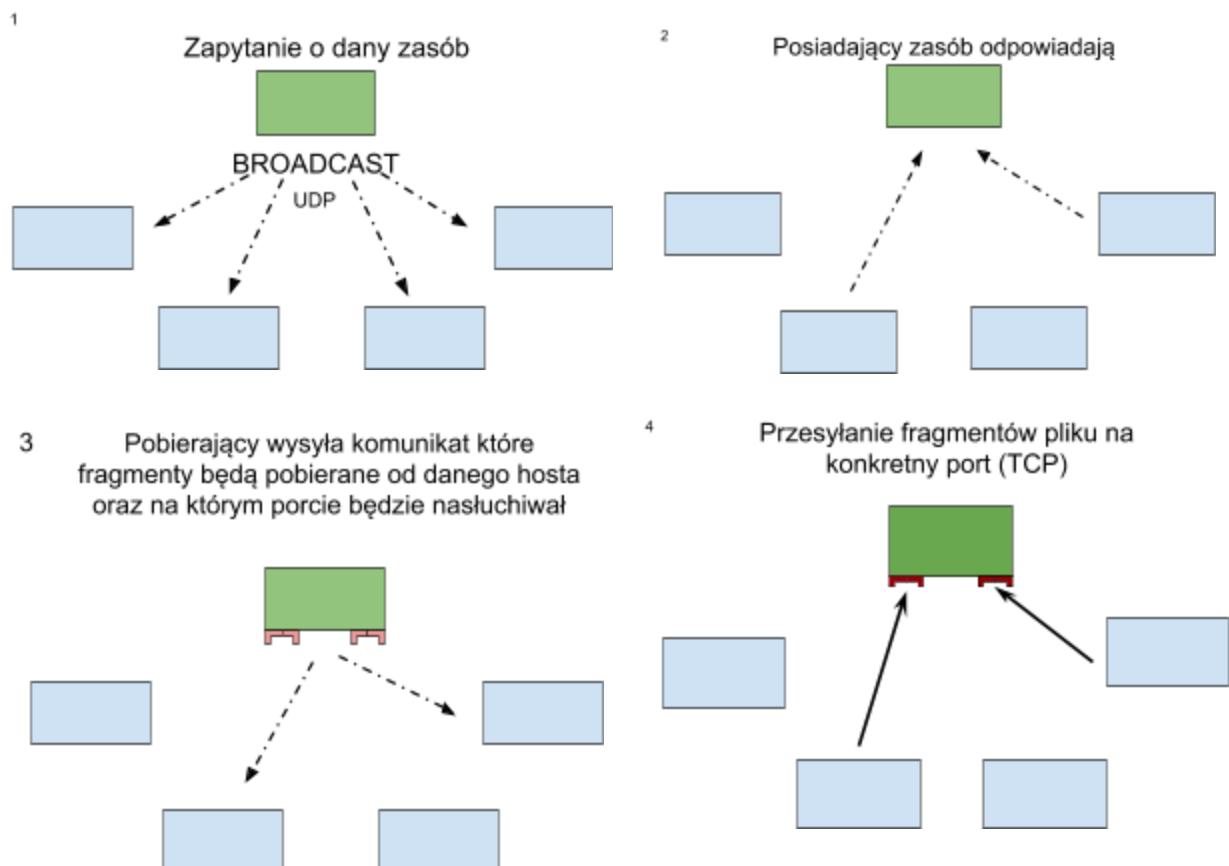
Pakiety stosowane w naszym protokole będą miały formę pakietów TCP (do transferu plików - transferowe) lub UDP (do wszystkiego innego - komunikacyjne). Pakiet komunikacyjny składa się z metadanych w standardzie określonym przez protokół UDP oraz z danych w formacie JSON. Pakiet transferowy natomiast składa się z metadanych w standardzie TCP oraz danych w postaci kolejnych bajtów pliku lub jego fragmentu.

### Pakiety komunikacyjne (bazujące na UDP)

Pakiety komunikacyjne zawierać będą przede wszystkim informacje o typie komunikatu (pole "type") oraz inne parametry zależnie od rodzaju komunikatu. Pakiety te mogą być rozgłaszane lub wysyłane do jednego konkretnego hosta. Rodzaje pakietów komunikacyjnych:

1. **Powitanie** - zawartość pakietu: [ typ: *GREETING*, nadawca: <nazwa\_hosta> ] - komunikat rozsyłany broadcastem podczas wystartowania programu na hoście, hosty, podłączone do sieci odpowiadają takim samym komunikatem (zawierającym swoją nazwę hosta), w przypadku, gdy dana nazwa jest już zajęta (przyjdzie odpowiedź z identyczną nazwą hosta która podaliśmy), program prosi ponownie o wybranie innej nazwy.
2. **Prośba o wysłanie listy zasobów** - zawartość pakietu: [ typ: *REQLIST* ] - komunikat wysyłany do konkretnego adresata, zobowiązany jest on do odpowiedzi komunikatem broadcastowym typu *RESPLIST* (poniżej opis). Dzięki temu zainicjuje on synchronizację list oryginałów w całej sieci. Przed każdą operacją: pobrania, dodania, usunięcia zasobu lub od razu po uruchomieniu host inicjujący ją będzie przedtem synchronizował listę oryginałów wysyłając komunikat *REQLIST*.

3. **Rozgłoszenie swojej listy oryginałów** - zawartość pakietu: [ typ: RESPLIST, lista plików wraz ich właścicielami i z flagami (unieważniony, zablokowany) ] - pakiet będzie wysyłany broadcastem w odpowiedzi na komunikat REQLIST dowolnego z hostów (opisany powyżej). Broadcastowanie listy pomoże zachować spójność w całej sieci - lista zasobów oryginalnych każdego hosta powinna mieć taką samą zawartość.
4. **Prośba o pobranie zasobu** - zawartość pakietu: [ typ: REQFILE, nazwa zasobu: <nazwa\_pliku> ] - komunikat rozsyłany broadcastem, hosty, które posiadają dany plik odpowiadają komunikatem RESPFILE. Następnie program na podstawie liczby hostów, które odpowiedziały pozytywnie, odpowiednio dzieli plik na fragmenty i otwiera porty TCP odbiorcy, które będą oczekiwały na fragmenty pliku (pojedynczy port dla pojedynczego dostawcy fragmentu). Wysyłane są jednocześnie prośby o pobranie fragmentu zasobu - komunikaty REQFDATA. Poniżej rysunki obrazujące algorytm komunikatu:



5. **Odpowiedź na prośbę pobrania pliku** - zawartość pakietu: [ typ: RESPFILE, nazwa zasobu: <nazwa\_pliku> ] - host wysyła dany komunikat jako odpowiedź na komunikat REQFILE, powiadamia tym samym o gotowości do udostępnienia konkretnego zasobu, którego nazwa jest w komunikacie.
6. **Prośba o pobranie fragmentu zasobu** - zawartość pakietu: [ typ: REQFDATA, nazwa zasobu: <nazwa\_pliku>, offset: <int>, size: <int>, nr portu odbioru: <nr\_portu> ] - komunikat do konkretnego hosta, który ma dany zasób. Wysyłany jest tylko po upewnieniu się wcześniej komunikatem REQFILE, że adresat posiada pożądany

zasób. W komunikacie zawarta jest informacja o tym, które fragmenty i jakiego pliku mają być przesłane oraz na którym porcie TCP odbiorca będzie nasłuchiwał danych. W przypadku wystąpienia błędu podczas pobierania (timeout - 30s lub niekompletne dane), odbiorca ponawia prośbę. Po 3 nieudanych próbach przesyła prośbę o ten fragment do innego hosta.

7. **Powiadomienie o usunięciu / dodaniu zasobu** - zawartość pakietu: [ typ: *DELFILE* / *ADDFILE*, nazwa zasobu: *<nazwa\_pliku>*, właściciel: *<nazwa\_hosta>*, czas operacji: *<data-godzina>* ] - komunikat broadcast o usunięciu / dodaniu zasobu przez właściciela. Usunięty plik w innych węzłach także jest usuwany z list zasobów i dysku hosta. Nie wysyłamy tego komunikatu, gdy usunięta została kopia pliku. Wszystkie trwające w momencie usunięcia transfery danego zasobu zostają przerwane. W przypadku dodania nowego pliku pozostałe hosty po otrzymaniu tego komunikatu dodają nowy zasób do listy oryginałów w sieci.
8. **Unieważnienie zasobów** - zawartość pakietu: [ typ: *REVFILE*, nazwa zasobu: *<nazwa\_pliku>* ] - komunikat broadcastowy o unieważnieniu zasobu, wszystkie trwające w tym momencie transfery danego zasobu są kontynuowane. na listach zasobów hosty odznaczają dany zasób jako unieważniony.
9. **Zablokowanie / odblokowanie zasobu** - zawartość pakietu: [ typ: *LOCFILE* / *UNLOFILE*, nazwa zasobu: *<nazwa\_pliku>* ] - gdy właściciel blokuje/odblokowuje dany zasób oznaczając go odpowiednią flagą stanu blokowania - informacja o stanie blokowania zostaje rozdana w formie komunikatu broadcastowego odpowiednio typu *LOCFILE* / *UNLOFILE*. Pozostałe hosty po otrzymaniu takiego pakietu ustawiają flagę "zablokowany" dla danego zasobu na swoich listach zasobów.

## Pakiety transferowe (bazujące na TCP)

**Przesłanie fragmentu pliku pakietem transferowym (po uprzednim uzgodnieniu możliwości pobrania danego zasobu komunikatami REQFILE i REQFDATA):**

- a. odbiorca pliku nasłuchuje na porcie TCP którego numer wysłał do nadawcy
- b. na podany port przesyłane są kolejno fragmenty pliku w blokach 1024 bajtowych (tzn. po każdych 1024 przesyłanych bajtach następuje zapisanie ich do tymczasowego pliku reprezentującego pobrany fragment o który host wysłał prośbę)
- c. jeśli wystąpił błąd transmisji lub upłynął czas 30s (timeout) - wysyłamy ponownie (pakietem UDP z pkt. 4) prośbę o wysłanie danego fragmentu do tego samego hosta i pobieramy od nowa. Jeśli pobranie od konkretnego hosta nie powiedzie się 3 razy to kierujemy prośbę do innego hosta posiadającego plik.
- d. po otrzymaniu wszystkich bajtów fragmentu jest on pobrany i gotowy do konkatenacji z pozostałymi fragmentami pliku
- e. zamykamy port TCP

## Techniczna realizacja protokołu

Do nadawania i odbierania komunikatów użyjemy gniazd BSD z odpowiednio podanym adresem docelowym - dla rozgłoszeń: opcja `SO_BROADCAST` oraz adresy składające się z 1 w części hosta. Gniazda będą tworzone dynamicznie w zależności od potrzeb. Zawsze jednak otwarte będzie jedno nasłuchujące gniazdo UDP w celu przyjmowania z sieci komunikatów UDP od innych hostów. Moduł Network udostępnił będzie funkcje: `sendUDP()`, `sendTCP()`, `listenTCP()`, `broadcastUDP()`, które tworzyć będą gniazda odpowiedniego typu oraz wysyłać podane w argumentach dane.

Na każdy odebrany komunikat program reagował będzie w odpowiedni sposób asynchronicznie, tak aby nie blokować głównego wątku.

## Analiza poprawności protokołu (potencjalne błędy)

Zakładamy że nazwa jest unikalnym identyfikatorem zasobu, nie mogą istnieć w sieci dwa zasoby o identycznych nazwach. Nazwa będzie ciągiem max. 255 znaków char.

Możliwa jest sytuacja, gdy dwa zasoby i identycznych nazwach zostaną dodane w tej samej chwili przez dwa różne hosty, w takim wypadku sprawdzony zostanie na każdym hoście indywidualnie czas dodania takiego zasobu (czas dodania jest dołączony w komunikacie rozgłaszanym po dodaniu zasobu) i dodany do list oryginałów będzie jedynie zasób, którego data dodania będzie wcześniejsza, zasób z późniejszą datą zostanie usunięty z listy w przypadku gdy został już na nią wcześniej dodany z przyczyny wcześniejszego otrzymania komunikatu przez dany host. Host, który później doda zasób będzie natychmiast go usuwał.

Inne sytuacje błędne mogą wynikać z utraconych pakietów UDP np. wskutek nagłego rozłączenia / awarii sieci. W takim wypadku host, który zostanie przywrócony do sieci będzie synchronizował natychmiast swoją listę oryginałów. Gdy mimo wszystko listy oryginałów na jakichś hostach będą się różniły, istnieje ryzyko próby pobrania unieważnionego zasobu, zasobu nieistniejącego na hoście lub zablokowanego zasobu, zostanie ona wówczas zignorowana.

Gdy z jakiegoś powodu dokończenie rozpoczętej transmisji pliku okaże się niemożliwe, pliki tymczasowe z dotychczas pobranymi częściowymi danymi zostaną wyczyszczone.

## 5. Realizacja współbieżności

Współbieżność będzie realizowana w następujący sposób: W aplikacji będą istniały współbieżnie 2 główne wątki: wątek nasłuchujący, oczekujący na komunikaty od pozostałych węzłów sieci (*Listener* na schemacie w punkcie 2.) oraz wątek służący do obsługi zapytań użytkownika (*Controller* na schemacie w punkcie 2.). Oprócz tego, w przypadku przyścia komunikatu z zewnątrz, na obsługę każdej prośby (np. o udostępnienie zasobów, unieważnienie istniejącego zasobu) tworzony jest osobny wątek (*Network task* na

schemacie w punkcie 2.). Także do przetworzenia poleceń użytkownika (np. utworzenie nowego zasobu, pobrania danego zasobu) tworzone są nowe wątki (także *Network task* - każdy taki wątek obsługuje indywidualnie połączenia z siecią poprzez moduł *Network*).

## 6. Zarys koncepcji implementacji

Program zaimplementujemy przy użyciu C++. Komunikaty opisujące pakiety stworzone będą w standardzie JSON, do ich tworzenia i czytania użyjemy biblioteki JSONCpp. Zdecydowaliśmy się stworzyć własne metody statyczne wykorzystujące API gniazd BSD do celów naszego projektu. Umieścimy je w odrębnej klasie.

Do realizacji współbieżności wykorzystamy biblioteki standardowe C++11:

- *thread* (realizacja wielowątkowości)
- *mutex* (do synchronizacji).