Dokumentacja projektu z przedmiotu Programowanie Sieciowe

Autorzy
Soroka Hubert
Łukasz Jaremek
Daniel Kobiałka
Radosław Kostrzewski

Data
19 stycznia 2023

Polecenie

Napisać program obsługujący prosty protokół P2P (Peer-to-Peer). Założenia:

- Zasób to plik identyfikowany pewną nazwą, za takie same zasoby uważa się zasoby o takich samych nazwach.
- Rozmiar zasobu jest znaczny (tj. większy od jednorazowego transferu sieciowego)
- Początkowo dany zasób znajduje się w jednym węźle sieci, następnie może być propagowany do innych węzłów w ramach inicjowanego przez użytkownika ręcznie transferu (patrz dalej) – raz pobrany zasób zostaje zachowany jako kopia.
- Tak więc, po pewnym czasie działania systemu ten sam zasób może znajdować się w kilku węzłach sieci (na kilku maszynach).
- Program ma informować o posiadanych lokalnie (tj. w danym węźle) zasobach i umożliwiać ich pobranie.
- Program powinien umożliwiać:
  - wprowadzanie przez użytkownika nowych zasobów – z lokalnego systemu plików,
  - pobieranie konkretnych nazwanych zasobów ze zdalnego węzła (jednego naraz)
  - rozgłaszanie informacji o posiadanych lokalnie zasobach.
- W przypadku pobierania zdalnego zasobu użytkownik decyduje skąd zostanie on pobrany.
- Zasób pobrany do lokalnego węzła jest kopią oryginału, kopia jest traktowana tak samo jak oryginał (są nierozróżnialne) – tj.: istnienie kopii jest rozgłaszane, tak samo jak oryginału.
- Należy zwrócić uwagę na różne obsługę różnych sytuacji wyjątkowych – np. przerwanie transmisji spowodowane błędem sieciowym.
- Lokalizacja zasobów ma następować poprzez rozgłaszanie – wskazówka: użyć prot. UDP, ustawić opcje gniazda SO_BROADCAST, wykorzystać adresy IP rozgłaszające (same bity “1” w części hosta).
- Interfejs użytkownika – wystarczy prosty interfejs tekstowy, powinien on jednak obsługiwać współbieżny transfer zasobów (tj. Nie powinien się blokować w oczekiwaniu na przesłanie danego zasobu)

Warianty indywidualne:

1. W1 – zasymulować błędy protokołu (zgubienie rozgłaszanego UDP)
2. W2 – zasymulować błędy protokołu (zerwanie sesji TCP)
3. W3 – całość komunikacji (przesyłania zasobu) zrealizować na UDP, dodatkowo – zasób mieści się w całości w jednym datagramie (datagram danych może być zgubiony – należy to uwzględnić)

Informacje Ogólne

Skład zespołu:

- Soroka Hubert – 01158909@pw.edu.pl
- Łukasz Jaremek
- Daniel Kobiałka
- Radosław Kostrzewski

Data przekazania: 05.01.2023

Realizowany wariant: W1

Interpretacja treści zadania

Celem zadania jest napisanie programu z interfejsem tekstowym, który umożliwi obsługę sieci Peer-to-Peer, przez którą przekazywane są pliki pomiędzy węzłami. Użytkownik może łatwo uzyskać dostęp do pliku, który fizycznie znajduje się na węźle sieci. Jako węzeł rozumiemy instancję programu uruchomioną pod unikalnym adresem sieciowym. Warto zaznaczyć, że sieć nie umożliwia aktualizacji istniejących plików inaczej, niż przez zmianę nazwy, ponieważ zasoby o tych samych nazwach, są traktowane jednakowo, a zmiana zawartości pliku bez zmiany nazwy, nie będzie widoczna dla węzła sieci. Każdy węzeł będzie posiadał własną tablicę zasobów, w której będą znajdowały się informacje o dostępnych zasobach w każdym węźle sieci. Tablica zasobów będzie zmienną w klasie węzła, a nie plikiem.

Opis funkcjonalny

Przewidujemy udostępnienie użytkownikowi następujących funkcji:

- downloaded_files – wylistowanie zasobów fizycznie znajdujących się w lokalnym węźle
- available_files – wylistowanie unikalnych zasobów znajdujących się we wszystkich węzłach sieci
- download_file [nazwa pliku] – rozpocznij pobieranie pliku o danej nazwie dostępnego w sieci (zostanie wyświetlony

komunikat na konsoli, zawierający listę węzłów, na których szukany zasób jest dostępny). Nie powoduje pobrania, jeśli zasób już dostępny.

- upload_file [nazwa pliku]– umożliwienie pobrania pliku z tego węzła pozostałym węzłom z użyciem download_file
- download_progress [nazwa_pliku] – wyświetla komunikat o stanie przetwarzania danego pliku (dostępny lokalnie / pobierany / niepobierany)

Komunikacja

Użyjemy protokołu UDP do rozgłaszania tablicy dostępnych zasobów wszystkim węzłom z użyciem opcji BROADCAST oraz protokołu TCP do przesyłania zasobów pomiędzy węzłami.

Rozgłaszanie w UDP zostanie zrealizowane z użyciem gniazda, które będzie czekać na sygnały rozgłaszające w wątku programu do tego przeznaczonym, a następnie synchroniczne je obsługiwać poprzez aktualizacje tablicy zasobów. Po dodaniu nowego zasobu w węźle, z tego samego gniazda, będzie wysyłany komunikat rozgłaszający do pozostałych węzłów.

Przesyłanie plików w TCP również będzie wymagało stale nasłuchującego gniazda w osobnym wątku. Różnica jest taka, że połączenia będą obsługiwane współbieżnie z użyciem wątków. Węzeł, po otrzymaniu komunikatu zawierającego żądanie pobrania zasobu, zacznie przesyłać żądany plik lub wyśle komunikat. Gniazda inicjujące połączenia z innymi gniazdami będą tworzone w osobnych wątkach po wywołaniu funkcji download_file przez użytkownika. Program nie będzie czekał na pobranie pliku, lecz umożliwi użytkownikowi sprawdzenie stanu pobierania w funkcji download_progress, poprzez sprawdzenie stanu wątku pobierającego dany zasób.

Aktualizacja tablicy zasobów, a więc wyznacznika stanu dostępności zasobów w węzłach, będzie aktualizowana dopiero po pobraniu pliku, aby nie doszło do sytuacji, gdzie użytkownik otrzymuje informacje o dostępności pliku, który jeszcze nie jest kompletny. Po udanym pobraniu pliku, aktualizacja tablicy zasobów będzie rozgłaszana po UDP, wg opisu powyżej.

Po wywołaniu upload_file, plik o danej nazwie jest kopiowany z dysku do węzła, po czym jest rozgłaszana informacja o jego dostępności, analogicznie jak dla pobrania.

Gniazda słuchające obu protokołów będą nasłuchiwać na tym samym porcie.

Przy pobieraniu pliku z innego węzła, węzeł inicjujący połączenie wyśle komunikat, który będzie zawierał tylko nazwę żądanego zasobu (download_request). Węzeł docelowy najpierw odpowie mu prostym komunikatem (download_response) zawierającym potwierdzenie / zaprzeczenie lokalnej dostępności pliku (available) oraz, jeśli zasób nie jest dostępny na węźle docelowym, ale jest dostępny na innym w jego tablicy, adresem węzła, który posiada szukany zasób (node_addr). W przeciwnym przypadku, nastąpi wysyłanie pliku z użyciem istniejącego połączenia. Węzeł żądający pobrania najpierw sprawdzi, czy otrzymał potwierdzenie dostępności, a potem zacznie odbierać plik lub przerwie połączenie i skieruje się do wskazanego węzła.

Rozgłaszanie UDP będzie realizowane z użyciem tylko jednego komunikatu (broadcast_msg), który będzie zawierał adres węzła (node_addr) oraz listę zasobów na nim dostępnych (resources). Każdy inny węzeł po otrzymaniu takiego komunikatu zaktualizuje tablicę zasobów, poprzez ustawienie wartości dla nadającego węzła na otrzymaną listę zasobów.

Komunikaty będą miały w kodzie formę prostych obiektów, które będą serializowane przed przesłaniem, z użyciem biblioteki pickle.

Struktura komunikatów

Nazwy struktur i pól są robocze. Adresy węzłów zawierają też ich porty.

Komunikaty używane w TCP:

```
struct download_request {
    char[64] name;
}
```

```
struct download_response {
    bool available;
    char[24] node_addr;
}
```

Komunikat używany w UDP:

```
struct broadcast_msg {
    char[24] node_addr;
    char[64][512] resources;
}
```

Symulacja błędów

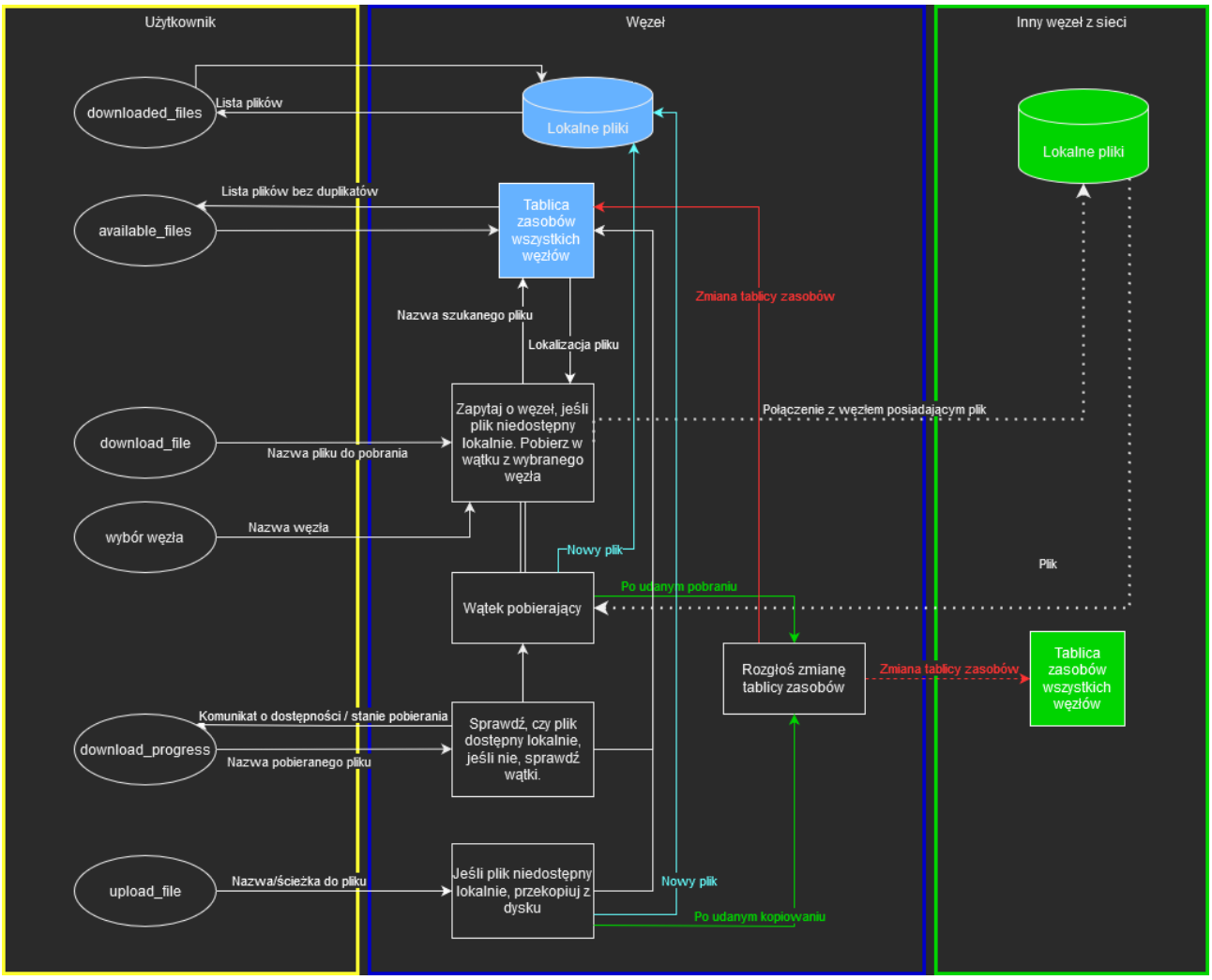
W ramach zadania, zasymulujemy błędy protokołu UDP. UDP będzie używane tylko i wyłącznie do rozgłaszania informacji o aktualizacji zasobów na danym węźle. W przypadku, gdy zostaną zgubione tylko komunikaty do pojedynczych węzłów, ale przynajmniej jeden z nich dotrze do celu, nadal możliwe jest przesyłanie plików. Podczas próby połączenia celem pobrania pliku z innego węzła (opisane w **Komunikacja**), w przypadku braku pliku na węźle docelowym, nastąpi sprawdzenie dostępności zasobu na innych węzłach wg tablicy zasobów węzła docelowego. Jeśli użytkownik zechce pobrać zasób, którego obecność nie została dobrze rozgłoszona, i jako węzeł docelowy ustali węzeł, który otrzymał poprawne rozgłoszenie, nastąpi przekierowanie do węzła zawierającego zasób.

W przypadku, gdy żaden węzeł nie otrzyma informacji o dostępności nowego zasobu, jest jeden scenariusz umożliwiający naprawienie sytuacji. Chodzi o ponowne wywołanie upload_file lub download_file dla dowolnych nowych zasobów na węźle, z którego rozgłaszanie się nie udało. Obie funkcje, po udanej operacji rozgłaszają swój stan innym węzłom. Przy rozgłaszaniu informują o wszystkich zasobach, więc te niedodane ostatnio, również będą rozgłoszone.

W programie, aby móc wymuszać takie sytuacje na potrzeby testów, dodaliśmy komendę imitate_packet_loss, która włącza flagę skutkującą niewysyłaniem komunikatów rozgłoszenia przy dodawaniu plików. Oczywiście, użytkownik nie powinien z funkcji skorzystać.

Struktura

Schemat planowanej architektury aplikacji jest zobrazowany na grafie:



Owale oznaczają operacje użytkownika, prostokąty operacje programu, błękitny oznacza zasoby lokalnego węzła, zielony to zasoby innego węzła z sieci, napis nad strzałką oznacza zwracaną wartość, strzałka to przepływ sterowania.

Implementacja

Stworzymy klasę Node, która będzie reprezentować jeden węzeł w sieci. Każdy węzeł będzie posiadać plik *config.json*, który będzie informował o adresach dostępnych w sieci węzłów. Warto zaznaczyć, że węzły muszą mieć unikalne kombinacje adresów i portów.

Interfejs użytkownika jest obsługiwany przez klasę P2PShell, której metody odpowiadają komendom z linii poleceń. P2PShell przechowuje referencję do obiektu klasy Node. Po otrzymaniu dozwolonej komendy, wywoływane są odpowiednie metody obiektu Node.

W konstruktorze Node, tworzone są dwa wątki: jeden realizuje nasłuchiwanie na połączenie TCP, drugi nasłuchuje na komunikat rozgłoszeniowy po UDP. Dla gniazd w obu wątkach włączone jest ponowne używanie adresu, a dla gniazda UDP, opcja broadcast.

Wątek gniazda UDP czeka na komunikaty rozgłoszeniowe (klasy BroadcastMessage), a po odebraniu aktualizuje dostępne zasoby na węźle, którego adres jest podany w komunikacie. Jeśli w tablicy zasobów nie istnieją zasoby z tego węzła – tworzony jest nowy rekord.

Wątek gniazda serwerowego TCP w nieskończonej (dopóki klient nie opuści interfejsu) pętli oczekuje na połączenia z żądaniem pobrania zasobu (klasy DownloadRequest). Jeśli żądany plik nie istnieje, odsyłany jest komunikat tekstowy. W przeciwnym razie, z użyciem sendall(), odsyłany jest lokalny plik o żądanej nazwie.

Pobieranie plików z innych węzłów jest stosunkowo proste. Po otrzymaniu od użytkownika informacji o nazwie żądanego pliku i docelowego węzła, następuje utworzenie obiektu żądania DownloadRequest, stworzenie gniazda TCP, połączenie się pod docelowy adres i stworzenie oddzielnego wątku, którzy przejmuje odpowiedzialność za pobranie zasobu.

Wątek pobierający plik po TCP jest bardzo prosty. Tworzy plik o danej nazwie, i zapisuje do niego po kolei odebrane z węzła docelowego dane. Dopiero po skończeniu pobierania, następuje wywołanie funkcji aktualizującej stan tablicy zasobów.

Funkcja update_file_list dzieli komunikat na kawałki wielkości CHUNK_SIZE, czyli globalnej stałej, oznaczającej liczbę bajtów (1024), które wysyłamy w ramach jednego pakietu TCP lub UDP. Po podzieleniu wysyła kawałki z użyciem gniazda UDP, tego samego, które jest też używane do odbierania informacji. Nie ma potrzeby tworzyć drugiego gniazda UDP.

Interfejs

Funkcje interfejsu są opisane w Opisie Funkcjonalnym. Przykładowa sesja korzystania z programu:

```
hubsor@DESKTOP-AVUM3TM:/mnt/d/Studia/python-c-socketing-psi/3$ /usr/bin/python3.9 /mnt/d/Studia/python-c-socketing-psi/3/main_test.py
Welcome! Type ? to list commands
> ?

Documented commands (type help <topic>):
========================================
available_files  download_progress  exit  imitate_packet_loss
download_file    downloaded_files   help  upload_file

> help available_files
List all files located in all nodes
> help download_file
Download file from the network
> available_files
List of available files:
-> pan-tadeusz.txt
> help upload_file
Upload file to the network
> upload file
Default: upload file
> upload_file
Please provide file name
> upload_file abc
File does not exist
> upload_file abc.txt
> download_file
Please provide file name
> download_file abc.txt
File already exists
> x
Bye
```

Więcej przykładów korzystania z interfejsu jest widocznych w Testach.

Aby wywołać program, należy za pomocą pythona 3.9 lub nowszego wywołać plik main.py.

Pliki

Przykładowy wygląd pliku config.json:

```
{
  "nodes": [
    {
      "node_name": "Bomba",
      "node_addr": "192.168.1.180",
      "node_port": 4200
    },
    {
      "node_name": "Radkoski",
      "node_addr": "192.168.1.115",
      "node_port": 4200
    },
    {
      "node_name": "Nolokon",
      "node_addr": "192.168.1.212",
      "node_port": 4200
    }
  ]
}
```

Plik zawiera nazwy trzech skonfigurowanych węzłów, wraz z ich adresami i portami.

Przykładowy wygląd pliku node_config.json:

```
{
   "node_name": "Bomba",
   "node_addr": "0.0.0.0",
   "node_port": 4200
}
```

Plik zawiera opis lokalnego węzła. Każdy węzeł powinien mieć kopię pliku config.json i swój własny plik node_config.json.

Zasoby lokalne węzła znajdują się w folderze node_data. Znajdujące się tam pliki są uznawane za dostępne dla węzła.

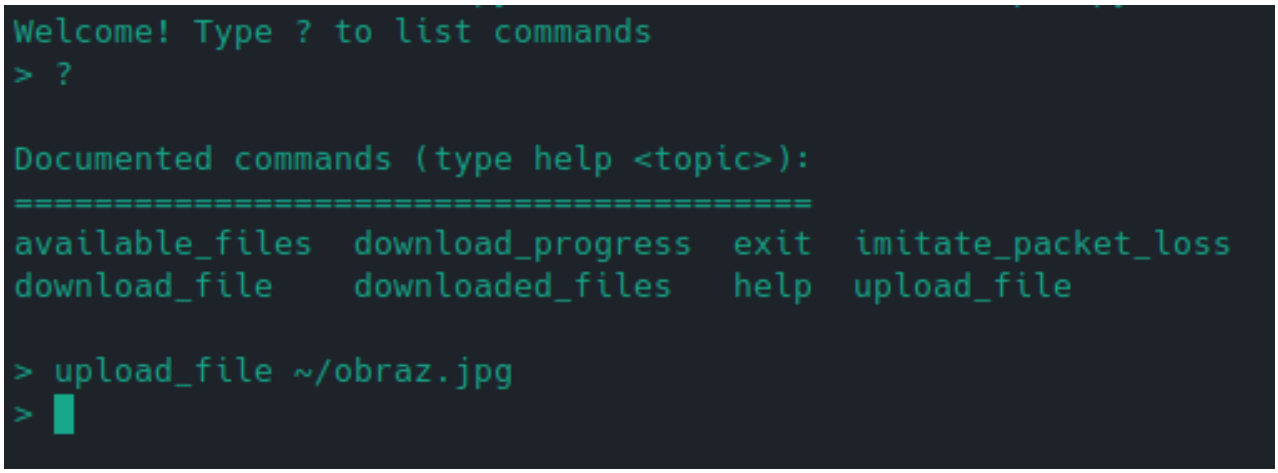
Narzędzia

Kod napisany w pythonie 3.9, z użyciem bibliotek: pickle – do serializacji przesyłanych obiektów, cmd – do interfejsu użytkownika oraz bibliotek wbudowanych do obsługi wielowątkowości, działań na plikach oraz formatu json. Pracowaliśmy w Visual Studio Code i Pycharmie. Do uruchamiania programu używaliśmy terminala bashowego.

Testy

Pierwszy test jest dość prosty. Chcemy sprawdzić, czy poprawnie przesyłają się komunikaty rozgłoszeniowe (innymi słowy, chcemy sprawdzić, czy występują błędy w części programu realizowanej w UDP).

Użytkownicy Bomba i Radkoski włączają program, po czym Bomba wgrywa plik obraz.jpg:



Następnie Nolokon włącza program (co oznacza, że nie otrzymał komunikatu o wgraniu pliku przez użytkownika Bomba) i Radkoski pobiera plik obraz.jpg od użytkownika Bomba:

```
Welcome! Type ? to list commands
> available_files
No files available
> available_files
List of available files:
-> obraz.jpg
> download_file obraz.jpg
Available file owners:
-> Bomba
Choose node to download from: Bomba
>
```

Po czym Nolokon również pobiera plik obraz.jpg od użytkownika Bomba:

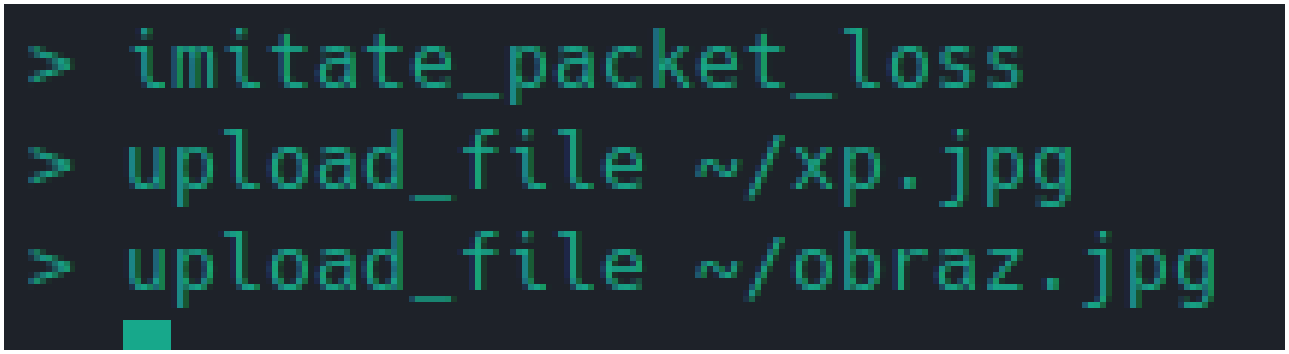
```
Welcome! Type ? to list commands
> available_files
List of available files:
-> obraz.jpg
> download_file obraz.jpg
Available file owners:
-> Radkoski
Choose node to download from: Radkoski
>
```

Jak widać, działa pobieranie plików oraz rozgłaszanie pobranej kopii, inaczej Nolokon nie mógłby pobrać pliku od Radkoskiego.

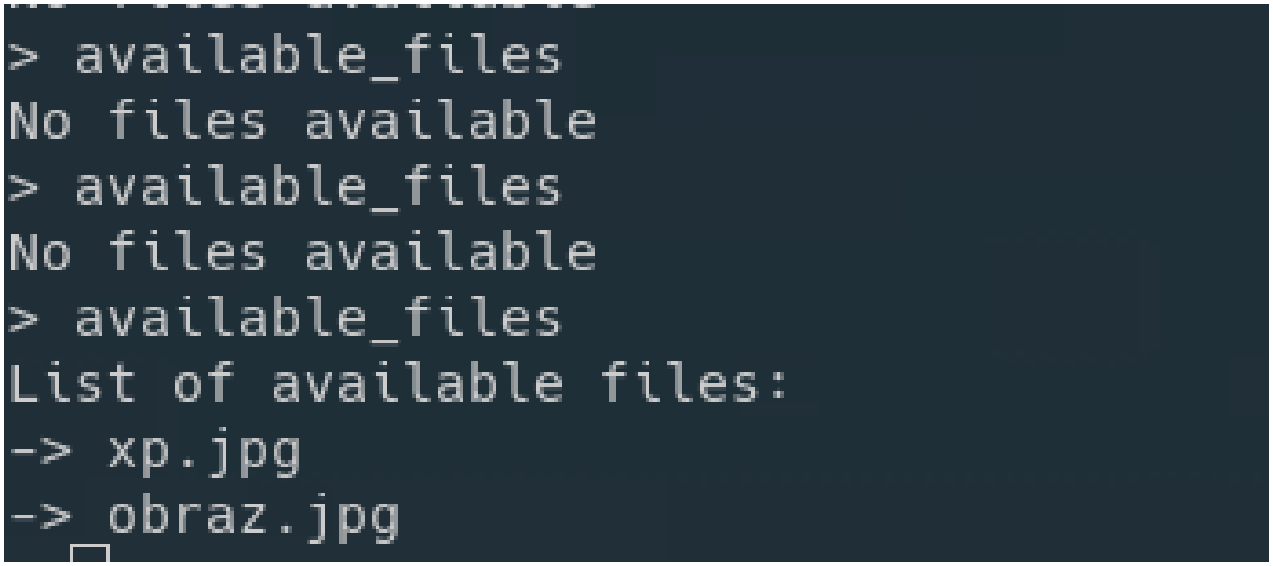
Drugi test polega na sprawdzeniu, czy wystąpią błędy w sytuacji, gdy rozgłoszenie pierwszego pliku jest zgubione, lecz rozgłoszenie następnego z tego samego węzła przechodzi poprawnie. Innymi słowy, sprawdzamy, czy poprawnie rozgłaszane są komunikaty o starszych plikach.

Aby zasymulować taką sytuację, użytkownik Bomba wyłączy rozgłaszanie przy pierwszym wywołaniu upload_file (symulując zgubienie pakietów UDP z komunikatem), po czym, przy następnym wywołaniu tej funkcji, włączy rozgłaszanie z powrotem.

Bomba:



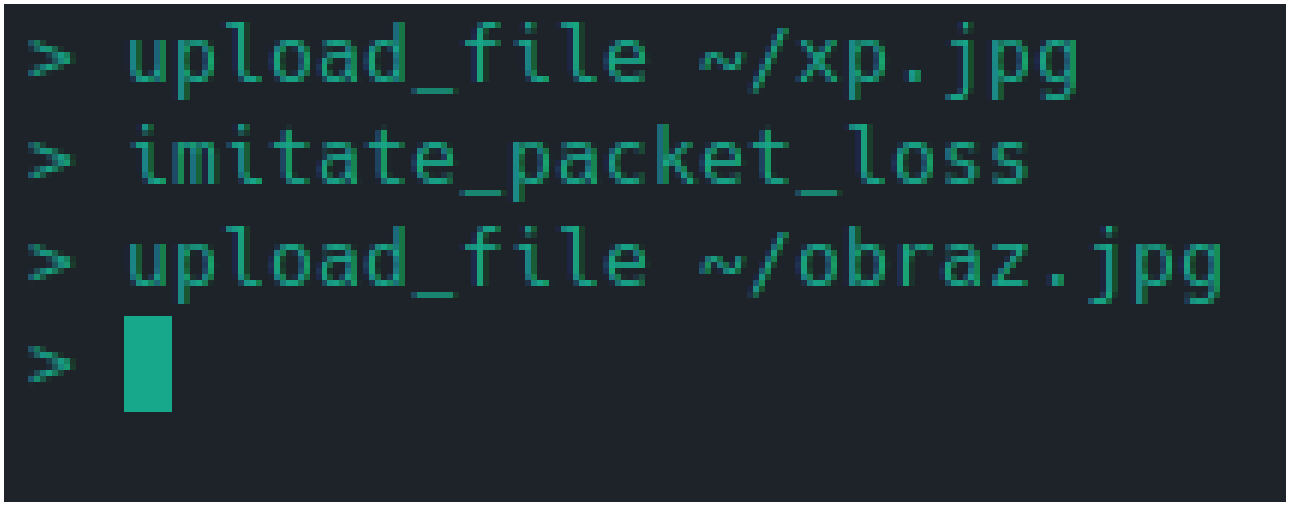
Radkoski, po drugim upload_file widzi oba pliki:



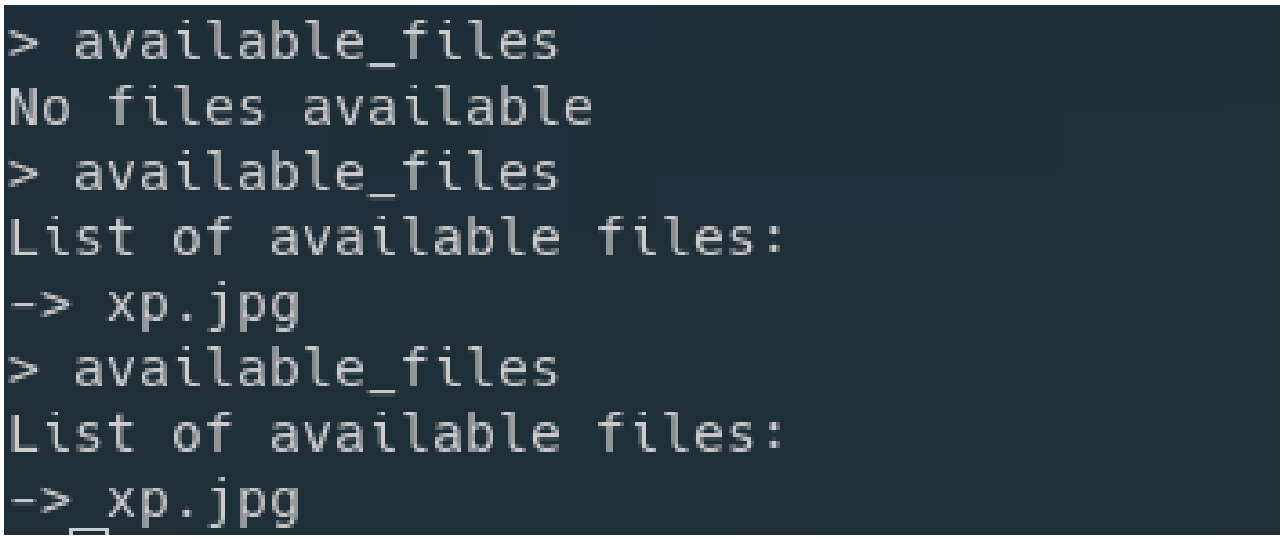
Rozgłaszanie działa poprawnie, przesyłając całą listę dostępnych plików. Jest to bardzo przydatne w sytuacjach zgubienia pakietów UDP, ponieważ następne udane rozgłoszenie „naprawia” wszystkie wcześniejsze zaginione.

Trzeci test jest podobny do poprzedniego, lecz tym razem użytkownik Bomba „gubi” pakiety UDP za drugim razem, przez co użytkownik Radkoski widzi tylko pierwszy plik. Zgubienie pakietów rozgłoszeniowych nie uniemożliwia pobrania plików, które były wcześniej rozgłoszone.

Bomba:

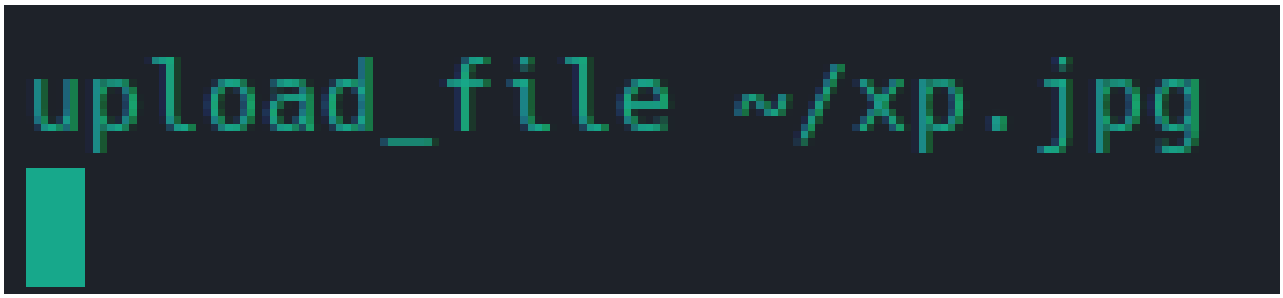


Radkoski:

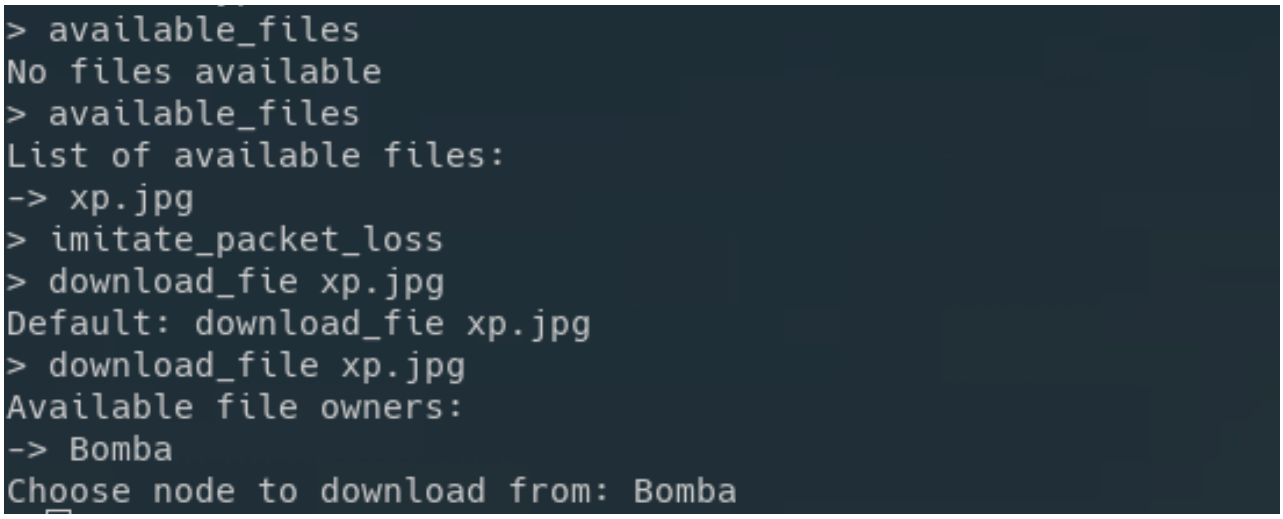


Czwarty, a zarazem ostatni test działa bardzo podobnie do pierwszego, lecz tym razem Radkoski wyłącza rozgłaszanie pakietów.

Użytkownicy Bomba i Radkoski włączają program, po czym Bomba wgrywa plik xp.jpg:



Następnie Nolokon włącza program (co oznacza, że nie otrzymał komunikatu o wgraniu pliku przez użytkownika Bomba) i Radkoski pobiera plik xp.jpg od użytkownika Bomba, ale z racji na wyłączenie rozgłaszania, informacja o kopii xp.jpg na komputerze Radoskiego nie rozniosła się po sieci:



Nolokon nie widzi żadnych dostępnych plików w sieci: