# PSI - Projekt - dokumentacja końcowa

04.01.2024 r.
Zespół **32**, skład:
Jakub Proboszcz
Paweł Kochański
Krzysztof Pałucki
Kamil Michalak

## Opis projektu i implementacji

**Cel Projektu:**
Projekt "PseudoTor" został stworzony jako prosty system anonimizujący oparty na sieci Tor, który umożliwia tunelowanie ruchu TCP przez publiczną sieć. System opiera się na koncepcji węzłów pośredniczących, które wprowadzają pseudolosowe opóźnienia oraz dokonują zmiany segmentacji przesyłanego strumienia ruchu w celu anonimizacji przesyłanego ruchu.

**Zmiana w stosunku do ustaleń wstępnych**
W ustaleniach wstępnych przyjęliśmy, że do szyfrowania ruchu między klientem a węzłem pośredniczącym używany będzie protokół IPSec. Jednakże, zdecydowaliśmy się zmienić protokół zapewniający szyfrowanie na TLS. Powody są następujące:

- IPSec wymaga działania programu tunelującego w tle u klienta, oddzielnego od programu przesyłającego dane - w dokumentacji wstępnej ustalono, że po stronie klienta będzie potrzebna tylko biblioteka w języku Python.
- Program tunelujący IPSec wymaga uprawnień superużytkownika, co uniemożliwiłoby testowanie naszego rozwiązania w środowisku BigUbu.
- Program tunelujący IPSec wymaga komendy `systemctl`, która jest niedostępna w środowisku WSL, z którego korzysta 3 z 4 członków naszego zespołu.
- Korzystanie z IPSec znacznie utrudniłoby lub uniemożliwiłoby przeprowadzenie testów w środowisku kontenerów Docker.

Protokół TLS, w odróżnieniu od IPSec jest wspierany przez bibliotekę standardową języka Python i nie wymaga oddzielnych programów ani uprawnień superużytkownika.

**Implementacja:**
**Biblioteka kliencka (`client/`):**

- Klient korzysta z modułu `pseudotor_client` - aplikacja kliencka za pomocą jednej funkcji (dostępnej jako context manager) `pseudotor_wrap` uzyskuje połączenie z serwerem za pośrednictwem węzła pośredniczącego.
- Sygnatura funkcji `pseudotor_wrap`:

```
def pseudotor_wrap(connecting_socket: socket.socket, server_address: str,
                   port: int, overseer_address: str) -> SSLSocket
```

- Funkcja ta przy wywołaniu wykonuje pobranie listy węzłów pośredniczących, wybranie jednego do komunikacji i nawiązanie połączenia z węzłem pośredniczącym i zwraca połączone gniazdo gotowe do komunikacji.

- Przy wyjściu z bloku `with` funkcja zamyka gniazdo SSL, ale gniazdo przekazane do funkcji powinno być zamknięte przez użytkownika biblioteki.
- Reszta komunikacji wygląda tak samo, jak w przypadku zwykłego gniazda TCP w API Python.
- Przykładowy program kliencki korzystający z naszej biblioteki jest zawarty w pliku `client.py`. Argumenty: nazwa/adres serwera, port serwera, nazwa/adres nadzorcy

**`client/middle_node_chooser.py`:**

- Moduł odpowiadający za wybór węzła pośredniczącego z dostępnej listy.
- Wysyła zapytanie do overseera, odbiera listę węzłów, a następnie dokonuje wyboru losowego węzła spośród otrzymanych.

**`client/pseudotor_client.py`:**

- Przygotowuje podane gniazdo TCP do komunikacji przez sieć PseudoTor.
- Wykorzystuje moduł wyboru węzła pośredniczącego.
- Tworzy bezpieczne połączenie TLS z węzłem.

**Węzeł pośredniczący (`middle_node/`):**

- Uruchamia węzeł pośredniczący, nasłuchujący na połączenia od klientów.
- Obsługuje tunelowanie i modyfikację ruchu.
- Rejestruje się u nadzorcy.
- Przyjmuje parametry: nazwa/adres interfejsu, na którym ma działać węzeł, nazwa/adres nadzorcy, parametry modyfikacji ruchu

**`middle_node/middle_node.py`:**

- Główna klasa węzła pośredniczącego.
- Przetwarza argumenty wywołania programu.
- Uruchamia wątek rejestratora i wątki obsługujące połączenia z klientami za pomocą pozostałych modułów.

**`middle_node/registrant_thread.py`:**

- Wysyła regularne zgłoszenia rejestracyjne do nadzorcy.

**`middle_node/connection_handling.py`:**

- Tworzy wątki obsługujące połączenia z klientami.
- Tworzy połączenie z serwerem i przesyła potwierdzenie do klienta.

**`middle_node/forwarder.py`:**

- Obsługuje odbieranie danych z klienta i przekierowywanie ich do serwera oraz odwrotnie.
- Za pomocą poniższego modułu modyfikuje ruch.

**`middle_node/packet_modifying.py`:**

- Implementuje zmianę segmentacji oraz opóźnianie danych.

**Nadzorca (`overseer/`):**

- Obsługuje zgłoszenia rejestracyjne i zapytania o listę węzłów od węzłów pośredniczących.
- Przyjmuje parametr: nazwa/adres interfejsu, na którym ma działać nadzorca

**`overseer/overseer.py`:**

- Główny program nadzorcy - tworzy wątki na każde przychodzące zgłoszenie.

**`overseer/middle_nodes.py`:**

- Lista węzłów pośredniczących, zabezpieczona pod kątem wielowątkowości.
- Obsługuje dodawanie i usuwanie (po upłynięciu podanego czasu) węzłów pośredniczących.

**`overseer/request_handling.py`:**

- Obsługuje zapytania od węzłów pośredniczących i klientów.
- Za pomocą powyższej klasy rejestruje węzły i udostępnia listę dostępnych.

**Serwer (`server/`):**

- Serwer oczekujący na dane od klientów - nie jest częścią naszego rozwiązania, służy do testów.
- Przyjmuje parametry: nazwa/adres interfejsu oraz port, na którym ma działać

# Scenariusze komunikacji

1. **Komunikacja klient-nadzorca:**
    - Klient wysyła zapytanie o listę dostępnych węzłów pośredniczących.
    - Nadzorca odsyła listę węzłów pośredniczących.

2. **Komunikacja klient-węzeł pośredniczący:**
    - Klient i węzeł pośredniczący nawiązują bezpieczne połączenie TLS. Zakładamy, że węzeł pośredniczący posiada ważny certyfikat; do testów używamy certyfikatu samopodpisanego.
    - Klient przesyła do węzła pośredniczącego adres IP oraz port serwera, z którym chce się połączyć.
    - Węzeł pośredniczący przesyła potwierdzenie połączenia z serwerem, lub informację, że połączenie się nie powiodło.
    - Dane przesyłane przez klienta są przekazywane do serwera i odwrotnie, dopóki klient lub serwer nie zamknie połączenia z węzłem pośredniczącym.

3. **Komunikacja węzeł pośredniczący-nadzorca:**
    - Węzeł pośredniczący wysyła prośbę o rejestrację, co ustalony okres.

4. **Komunikacja węzeł pośredniczący-serwer:**
    - Węzeł pośredniczący nawiązuje połączenie z serwerem.
    - Dane przesyłane przez klienta są przekazywane do serwera i odwrotnie, dopóki klient lub serwer nie zamknie połączenia z węzłem pośredniczącym.

Cała komunikacja odbywa się za pomocą protokołu TCP.

# Definicje komunikatów

1. **Zapytanie o listę węzłów pośredniczących:**

Tekst w kodowaniu ASCII: “`PSEUDOTOR_GET_NODES`”

2. **Odpowiedź z listą węzłów pośredniczących:**

Lista adresów IP węzłów pośredniczących w formacie tekstowym, rozdzielone spacjami, w kodowaniu ASCII.

3. **Inicjacja połączenia klient-węzeł pośredniczący:**

Klient wysyła 6 bajtów danych: adres IP w postaci binarnej zwracanej przez funkcję `inet_aton` (4 bajty) oraz numer portu serwera, z którym chce się połączyć w postaci binarnej big-endian.

4. **Potwierdzenie połączenia z serwerem:**

Węzeł pośredniczący przesyła 2-bajtowy napis w kodowaniu ASCII: “`OK`” jeżeli połączenie się powiodło, “`NO`” jeśli nie.

5. **Tunelowane dane:**

Dane wysyłane przez klienta i serwera do siebie nawzajem są przekazywane w dokładnie tej postaci, w jakiej były wysłane.

6. **Rejestracja węzła pośredniczącego:**

Tekst w kodowaniu ASCII: “`PSEUDOTOR_REGISTER`”; adres IP węzła rejestrowanego jest pozyskiwany z adresu źródłowego połączenia.

# Opisy zachowania podmiotów komunikacji

1. **Biblioteka kliencka:**

- Prosi o listę węzłów pośredniczących.
- Zmniejsza listę węzłów pośredniczących o adresy IP węzłów oznaczonych jako niedziałające. Jeżeli powstała lista jest pusta, zwraca błąd.
- Wybiera węzeł pośredniczący z listy.
- Nawiązuje połączenie z wybranym węzłem.
- Przesyła adres IP i port serwera do węzła pośredniczącego.
- Jeżeli połączenie lub wysyłanie danych do węzła pośredniczącego zakończyło się błędem, oznacza adres IP tego węzła jako niedziałający i wraca do początku procedury.
- Odbiera potwierdzenie połączenia z serwerem. Jeżeli połączenie się nie powiodło, zwraca błąd.
- Dalsza komunikacja zależy od konkretnego klienta - w testach wygląda następująco:
  Klient przesyła dane: 10000 bajtów ‘A’, po czym 3 bajty ‘END’, w kodowaniu ASCII
  Klient otrzymuje 3 bajty: ‘END’ w kodowaniu ASCII od serwera.
  Klient przesyła 3 bajty: ‘Bye’ w kodowaniu ASCII do serwera.

2. **Nadzorca:**

- Nasłuchuje na porcie 8001.
- Gdy nadzorca otrzymuje prośbę o rejestrację od węzłów pośredniczących, dodaje adres IP węzła do listy, lub, jeśli był zarejestrowany wcześniej, aktualizuje znacznik czasowy.

- Gdy nadzorca otrzymuje prośbę o udostępnienie listy węzłów pośredniczących, usuwa z listy węzły, które nie zarejestrowały się od określonego czasu, po czym przesyła listę klientowi.

3. **Węzeł Pośredniczący:**

- Uruchamia w tle wątek, który rejestruje się u nadzorcy raz na określony czas.
- Oczekuje na połączenia od klientów - każdy klient otrzymuje wątek na obsługę jego połączenia. Nasłuchuje na porcie 8000.
- Gdy otrzyma połączenie od klientów, otrzymuje adres IP serwera i port.
- Łączy się z podanym serwerem.
- Przesyła potwierdzenie połączenia z serwerem do klienta.
- Uruchamia 2 wątki do tunelowania danych: jeden od serwera do klienta, drugi odwrotnie.
- Każdy z tych wątków:
  Oczekuje na otrzymanie danych. Jeżeli nie otrzyma danych przez określony czas, wysyła zebrane wcześniej dane i opróżnia bufor.
  Otrzymane dane dodaje do bufora. Jeśli jest ich więcej niż określona ilość, wysyła je i opróżnia bufor.
  Jeżeli połączenie zostało zerwane lub drugi wątek zasygnalizował konieczność zakończenia, wysyła zebrane dane i sygnalizuje drugiemu wątkowi konieczność zakończenia.

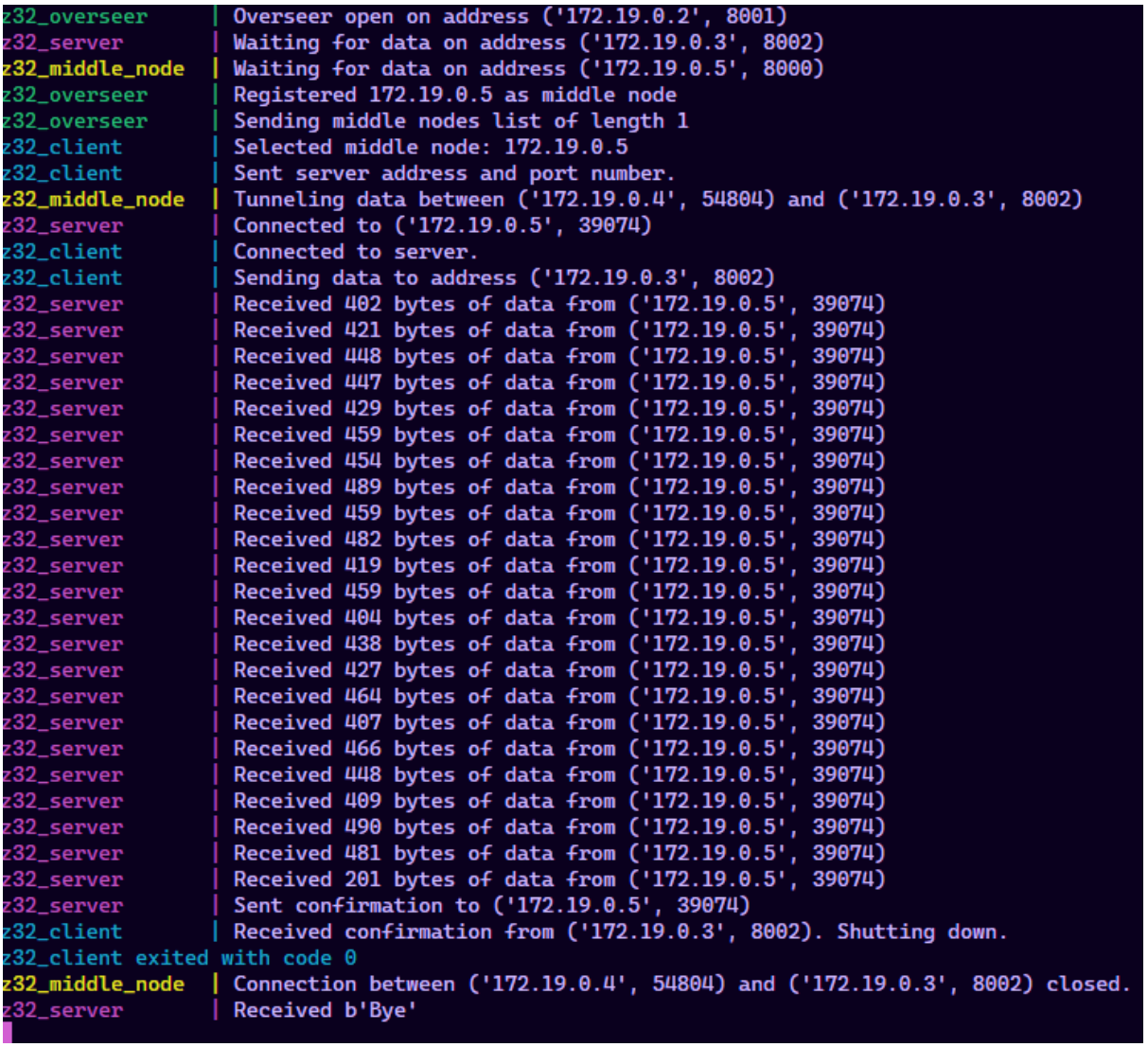
4. **Serwer:**

- Komunikacja zależy od konkretnego serwera - w testach wygląda następująco:
  Serwer otrzymuje dowolne dane zakończone 3 bajtami ‘END’ w kodowaniu ASCII; na bieżąco wypisuje, ile bajtów danych otrzymał.
  Serwer wysyła 3 bajty: ‘END’ w kodowaniu ASCII do klienta.
  Serwer otrzymuje 3 bajty danych od klienta.

# Wyniki testów

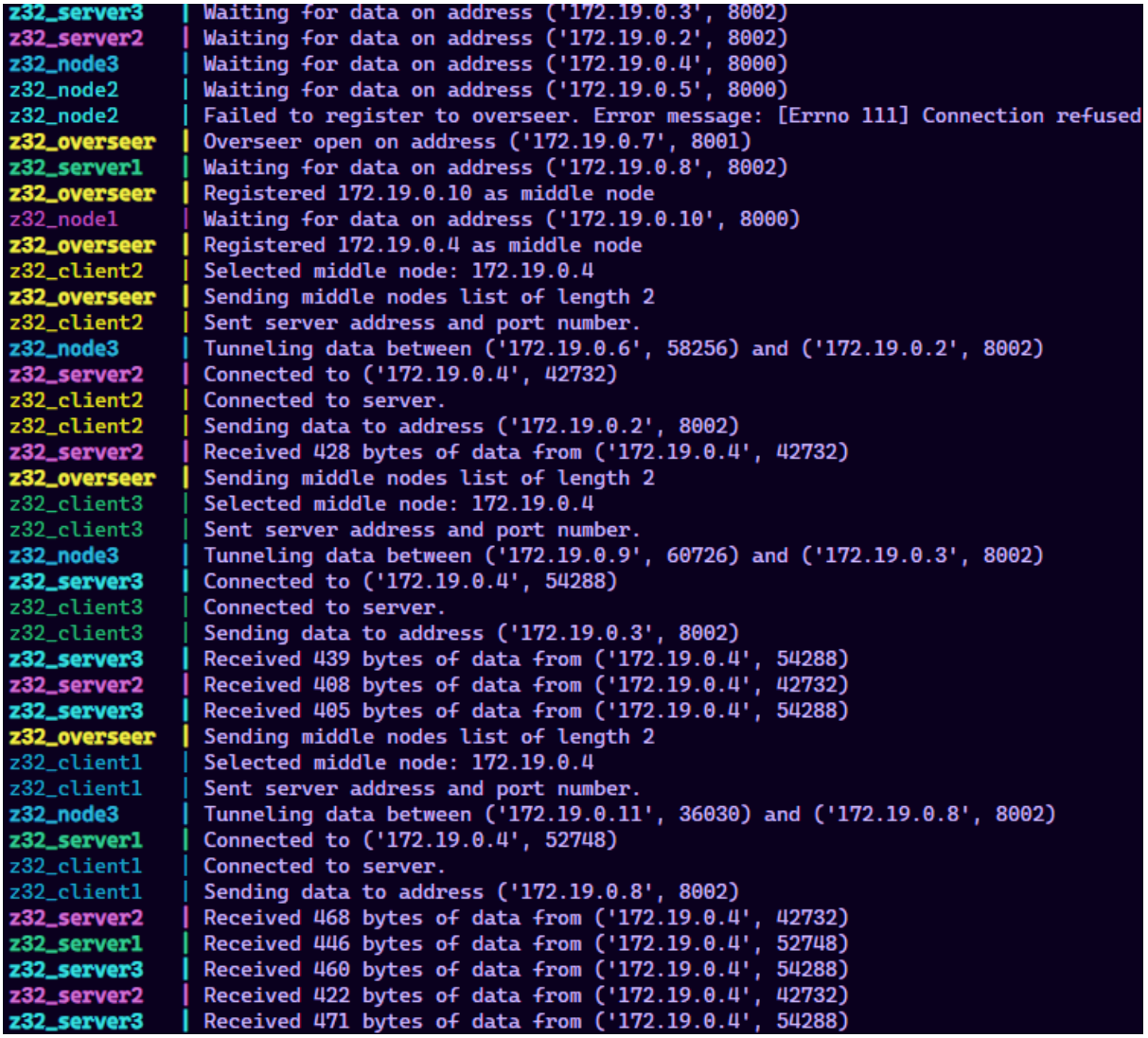
1. **Test poprawności połączenia:**

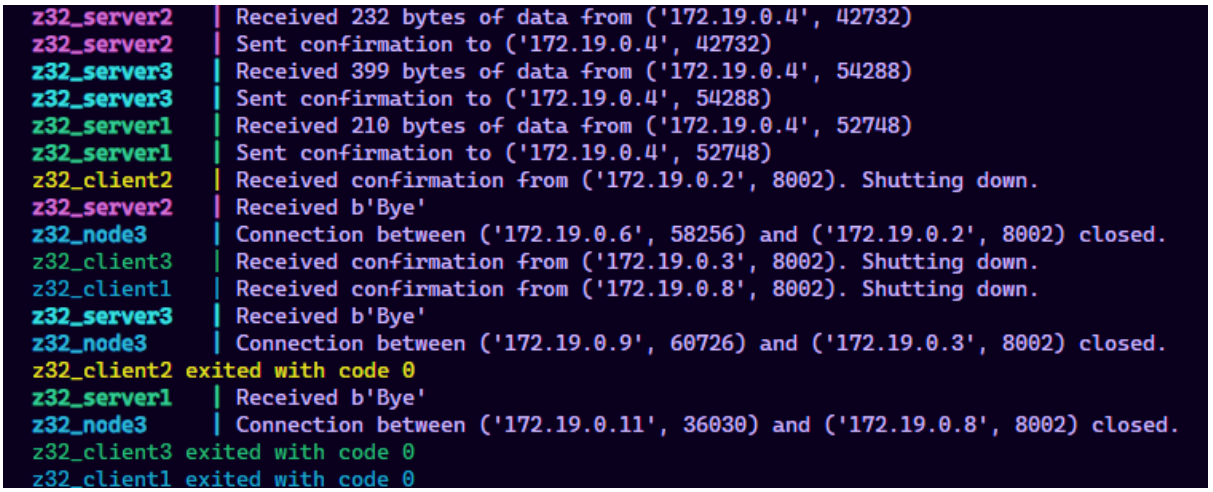
Podstawowa komunikacja klient-serwer za pośrednictwem sieci Pseudo-Tor.



2. **Test wielu równoległych połączeń:**

Analogicznie do poprzedniego testu, ale z 3 parami klient-serwer działającymi równolegle.




Pomimo awarii jednego z trzech middle-node'ów (spowodowanej uruchomieniem przed nadzorcą) komunikacja między parami klient-serwer zakończyła się sukcesem. W logach można zauważyć, że wykorzystywane są różne node'y do transportu danych.

3. **Test awarii nadzorcy:**

Próba powtórzenia testu 1, gdy nadzorca nie działa.
Próba pozyskania listy węzłów przez klienta:

```
❯ python client.py '127.0.0.1' 8002 "127.0.0.1"
Error occurred with message: Failed to obtain middle nodes list from overseer.
```

Próba rejestracji węzła:

```
❯ python middle_node.py '127.0.0.1' '127.0.0.1' -t 0.5 -ss 400 500
Waiting for data on address ('127.0.0.1', 8000)
Failed to register to overseer. Error message: [Errno 111] Connection refused
```

Awaria nadzorcy w trakcie komunikacji między klientem a serwerem:

```
Waiting for data on address ('127.0.0.1', 8002)
Waiting for data on address ('127.0.0.1', 8000)
Selected middle node: 127.0.0.1
Sent server address and port number.
Tunneling data between ('127.0.0.1', 60902) and ('127.0.0.1', 8002)
Connected to ('127.0.0.1', 46494)
Connected to server.
Sending data to address ('127.0.0.1', 8002)
Received 454 bytes of data from ('127.0.0.1', 46494)
Received 410 bytes of data from ('127.0.0.1', 46494)
Received 455 bytes of data from ('127.0.0.1', 46494)
Received 485 bytes of data from ('127.0.0.1', 46494)
Received 401 bytes of data from ('127.0.0.1', 46494)
Received 486 bytes of data from ('127.0.0.1', 46494)
Received 452 bytes of data from ('127.0.0.1', 46494)
Received 484 bytes of data from ('127.0.0.1', 46494)
Received 462 bytes of data from ('127.0.0.1', 46494)
Received 465 bytes of data from ('127.0.0.1', 46494)
Received 433 bytes of data from ('127.0.0.1', 46494)
Received 454 bytes of data from ('127.0.0.1', 46494)
Received 436 bytes of data from ('127.0.0.1', 46494)
Received 462 bytes of data from ('127.0.0.1', 46494)
Received 484 bytes of data from ('127.0.0.1', 46494)
Received 438 bytes of data from ('127.0.0.1', 46494)
Received 435 bytes of data from ('127.0.0.1', 46494)
Received 483 bytes of data from ('127.0.0.1', 46494)
Received 411 bytes of data from ('127.0.0.1', 46494)
Received 494 bytes of data from ('127.0.0.1', 46494)
Received 451 bytes of data from ('127.0.0.1', 46494)
Received 468 bytes of data from ('127.0.0.1', 46494)
Sent confirmation to ('127.0.0.1', 46494)
Received confirmation from ('127.0.0.1', 8002). Shutting down.
Received b'Bye'
Connection between ('127.0.0.1', 60902) and ('127.0.0.1', 8002) closed.
```

Komunikacja odbyła się poprawnie.

4. **Test awarii węzła pośredniczącego:**

Próba powtórzenia testu 1, gdy węzeł pośredniczący przestaje działać po zarejestrowaniu się u nadzorcy.
Nadzorca:

```
❯ python overseer.py '127.0.0.1'
Overseer open on address ('127.0.0.1', 8001)
Registered 127.0.0.1 as middle node
Sending middle nodes list of length 1
Sending middle nodes list of length 1
```

Klient:

```
❯ python client.py '127.0.0.1' 8002 "127.0.0.1"
Selected middle node: 127.0.0.1
Error occurred with message: No available PseudoTor middle nodes are registered
```

## Podsumowanie

Projekt "PseudoTor" dostarcza prosty, ale skuteczny system anonimizujący, umożliwiający bezpieczne tunelowanie ruchu TCP. Wprowadzanie pseudolosowych opóźnień i zmiana rozmiaru/segmentacji przesyłanego strumienia ruchu zwiększają poziom anonimizacji. Testy potwierdziły poprawność funkcji systemu, co czyni go użytecznym narzędziem dla użytkowników poszukujących dodatkowej warstwy prywatności w sieci.