

**Politechnika Poznańska, Wydział Elektryczny**

Bartosz Dominiak 122091  
Mateusz Byczkowski 121982  
BSI-1

# **Podstawy Teleinformatyki**

**Monitor ruchu w sieci  
z uwzględnieniem topologii**

## Dane kontaktowe:

Bartosz Dominiak bartoszjandominiak@gmail.com

Mateusz Byczkowski byczkowski38@gmail.com

## Harmonogram Prac

Data	Wykonane prace
23.03	Zrobienie prezentacji, zapoznanie się z tematem, zbieranie informacji, umieszczenie projektu na GitHub
6.04	Podział prac, uruchomienie aplikacji serwera/klienta
20.04	Prace nad aplikacjami serwera/klienta, uruchomienie TCP Dump
4.05	Prace nad aplikacjami serwera/klienta, testy przesyłu danych
18.05	Prace nad aplikacjami serwera, wnioskowanie ścieżki/topologii
1.06	Rysowanie topologii na aplikacji serwera np. za pomocą GraphViz
15.06	Poprawki, usunięcie potencjalnych błędów, poprawa UI

# Spis treści

1. Wprowadzenie.
  - 1.1. Cel.
  - 1.2. Zakres.
  - 1.3. Podział prac.
2. Opis ogólny.
  - 2.1. Charakterystyki aktorów.
  - 2.2. Wymagania funkcjonalne.
  - 2.3. Wymagania pozafunkcjonalne.
  - 2.4. Środowisko i technologie.
  - 2.5. Ograniczenia.
3. Diagramy.
  - 3.1. Diagram przypadków użycia.
  - 3.2. Diagram klas dla aplikacji klienta
  - 3.3. Diagram klas dla aplikacji serwera.
4. Efekty prac na poszczególnych etapach.
  - 4.1. Iteracja pierwsza.
  - 4.2. Iteracja druga.
  - 4.3. Iteracja trzecia.
  - 4.4. Iteracja czwarta.
  - 4.5. Iteracja piąta.
5. Instrukcja użytkownika aplikacji.
6. Plany możliwego dalszego rozwoju.
7. Podsumowanie.

# 1. Wprowadzenie

Niniejsza dokumentacja zawiera szczegółowy opis aplikacji służącej do monitorowania sieci oraz wyznaczania jej topologii. Ideą projektu jest zapoznanie się z sposobem działania połączeń między komputerami. Monitorowanie sieci, wykrywanie topologii i ruchu to tematy które cieszą się zainteresowaniem w związku z zapewnieniem bezpieczeństwa systemów komputerowych i sieci. Wybraliśmy ten temat ponieważ bliskie są nam kwestie bezpieczeństwa systemów komputerowych z racji wybranej przez specjalizacji.

## 1.1 Cel

Celem projektu jest przybliżenie wiedzy z zakresu sieci, gniazd sieciowych, komunikacji między komputerami i aplikacjami oraz zapewnienia im bezpieczeństwa.

## 1.2 Zakres

Aplikacja obejmować będzie część serwera i mnogą część aplikacji klienckich. Aplikacje klienckie będą komunikować się ze sobą oraz będą zapisywać logi komunikacyjne w postaci zrzutów logów pakietów przepływających przez aplikację. Okresowo logi w postaci zrzutów wysyłane będą do serwera którego głównym zadaniem będzie rysowanie topologii i utrzymywanie połączeń klientów.

## 1.3 Podział prac

Bartosz Dominiak	Mateusz Byczkowski
Aplikacja serwera Wnioskowanie ścieżki na podstawie przesłanych od klientów zrzutów Rysowanie graficzne topologii za pomocą programu GraphViz Dokumentacja Projektowa Poprawki w UI	Aplikacja Kliencka Cykliczne monitorowanie ruchu i wysyłanie go do serwera Testy przesyłu danych Konfiguracja i zarządzanie GitHub'em Prezentacja Dokumentacja Projektowa

## 2. Opis ogólny

Aplikacja skierowana jest dla celów edukacyjnych, do początkujących specjalistów bezpieczeństwa sieciowego. Projekt służy edukacji młodzieży i rozwojowi wiedzy z zakresów sieci i topologii oraz komunikacji między aplikacjami.

### 2.1 Charakterystyki aktorów

W tej sekcji znajdują się charakterystyki poszczególnych aktorów projektu to jest klienta i serwera.

- Klient- aplikacja kliencka uruchamiana na kilku badanych stacjach roboczych gdzie przechwytywane i zapisywane są pakiety
- Serwer- aplikacja serwera uruchamiana na jednej maszynie i wysyłająca do klientów żądania w celu pozyskania informacji o sieci.

### 2.2 Wymagania funkcjonalne

W tej sekcji prezentowane są wymagania funkcjonalne odnośnie aplikacji klienta i serwera.

#### Strona klienta

- Łączenie się z serwerem
- Wykonywanie polecenia ping do innych klientów.
- Nasłuchiwanie wysyłanych i odbieranych pakietów
- Zapisywanie podsłuchanych pakietów
- Wysyłanie podsłuchanych pakietów do serwera na żądanie
- Próba nawiązania połączenia z innymi klientami podłączonymi do serwera

#### Strona serwera

- Utrzymywanie połączeń z klientami
- Rozsyłanie aktualnej list adresów klientów połączonych z serwerem
- Wysyłanie żądań do klientów w celu uzyskania konkretnych informacji
- Odbiór i przechowywanie przesyłanych informacji od klientów
- Filtrowanie otrzymanych informacji
- Wyświetlanie odebranych pakietów
- Rysowanie topologii sieci z pomocą GraphViz

### 2.3 Wymagania pozafunkcjonalne

W tej sekcji znajdują się wymagania pozafunkcjonalne dotyczące projektu.

- Dostęp do sieci LAN.
- System Windows
- Co najmniej dwie maszyny wirtualne lub komputery
- Wymagane uprawnienia administratora

## 2.4 Środowisko i technologie

W poniższej sekcji prezentowane są technologie użyte w projekcie oraz środowisko uruchomieniowe aplikacji.

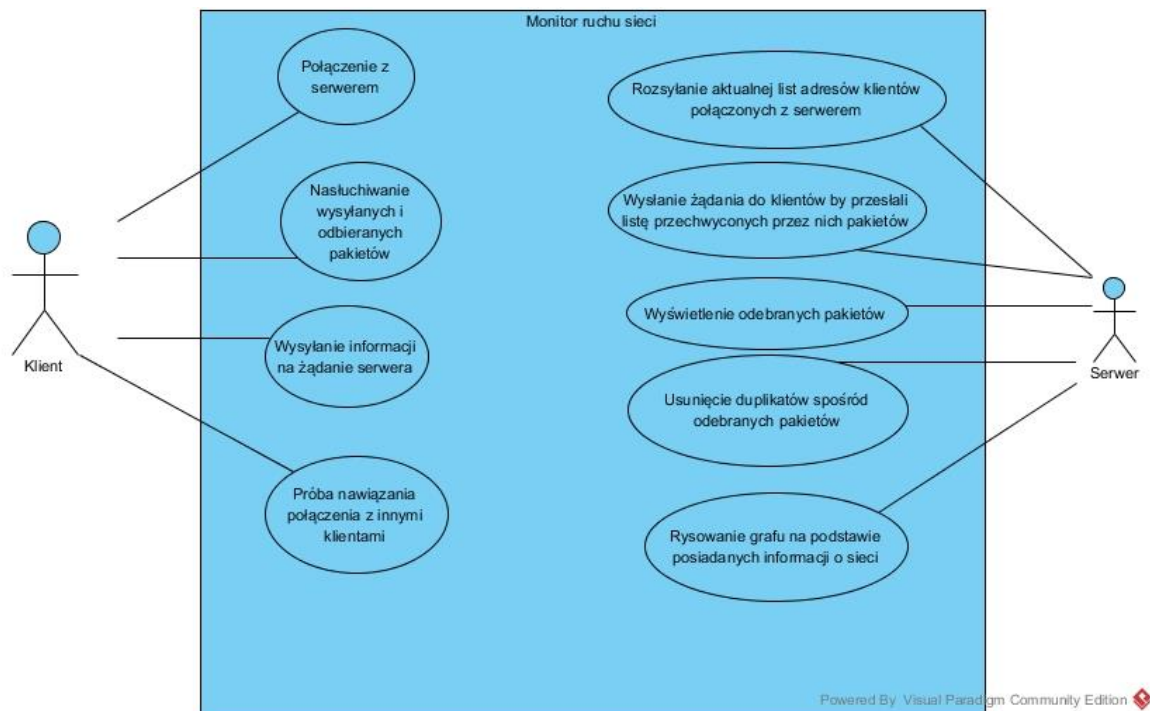
- Środowisko Visual Studio 2015.
- Język C#.
- Aplikacja konsolowa.
- GraphViz
- Program do przechwytywania pakietów MJSniffer

## 3. Diagramy.

W tej sekcji znajdują się diagramy które prezentują w sposób wizualny projekt.

### 3.1. Diagram przypadków użycia.

W tej sekcji znajduje się diagram przypadków użycia. W diagramie wyróżniamy dwóch aktorów, serwer i klient. Aktorzy są połączeni z przypadkami użycia które ich dotyczą.



*Diagram przypadków użycia.*

### 3.2. Diagram aktywności.

W tej sekcji znajduje się diagram aktywności. Pokazuje on kolejne działania po wywołaniu komendy dump w aplikacji serwera i jego wpływ na aplikacje klienta.

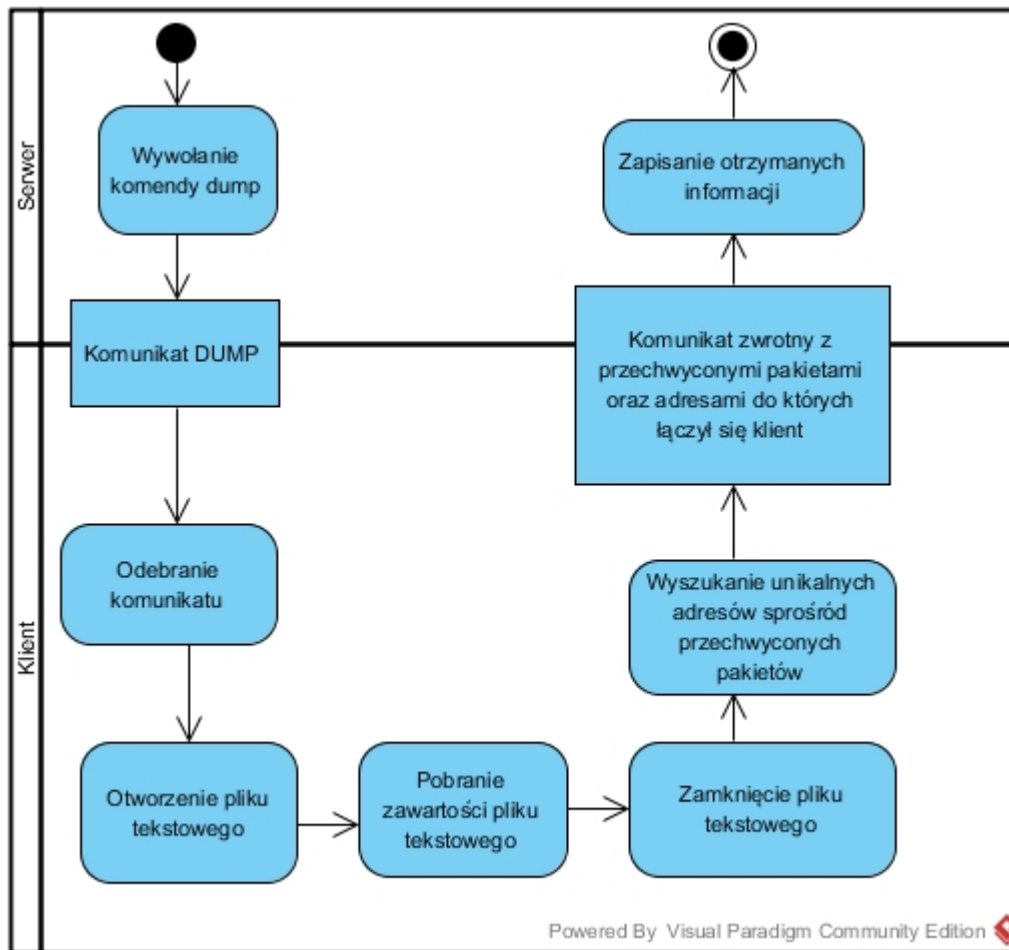
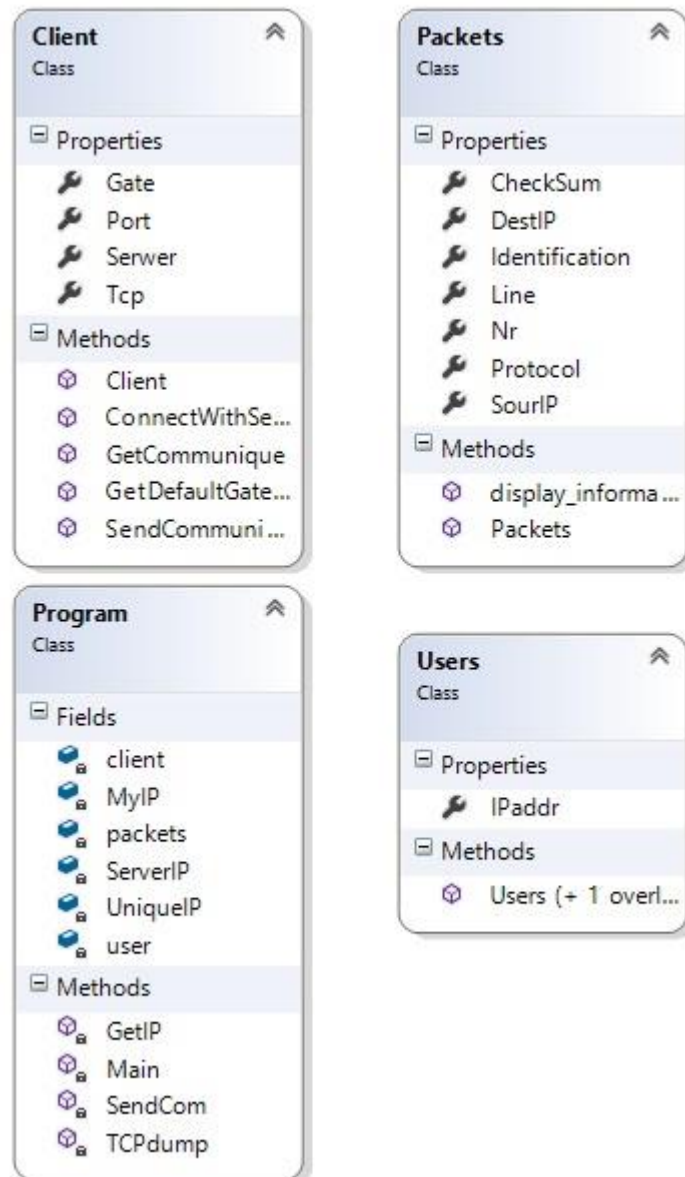


Diagram aktywności

### 3.3. Diagram klas dla aplikacji klienta.

W tej sekcji znajduje się diagram klas dla aplikacji klienta. Pokazuje on pola i metody zaimplementowane w projekcie Client.



*Diagram klas dla aplikacji klienta*



### 3.4. Diagram klas dla aplikacji serwera.

W tej sekcji znajduje się diagram klas dla aplikacji klienta. Pokazuje on pola i metody zaimplementowane w projekcie Serwer.

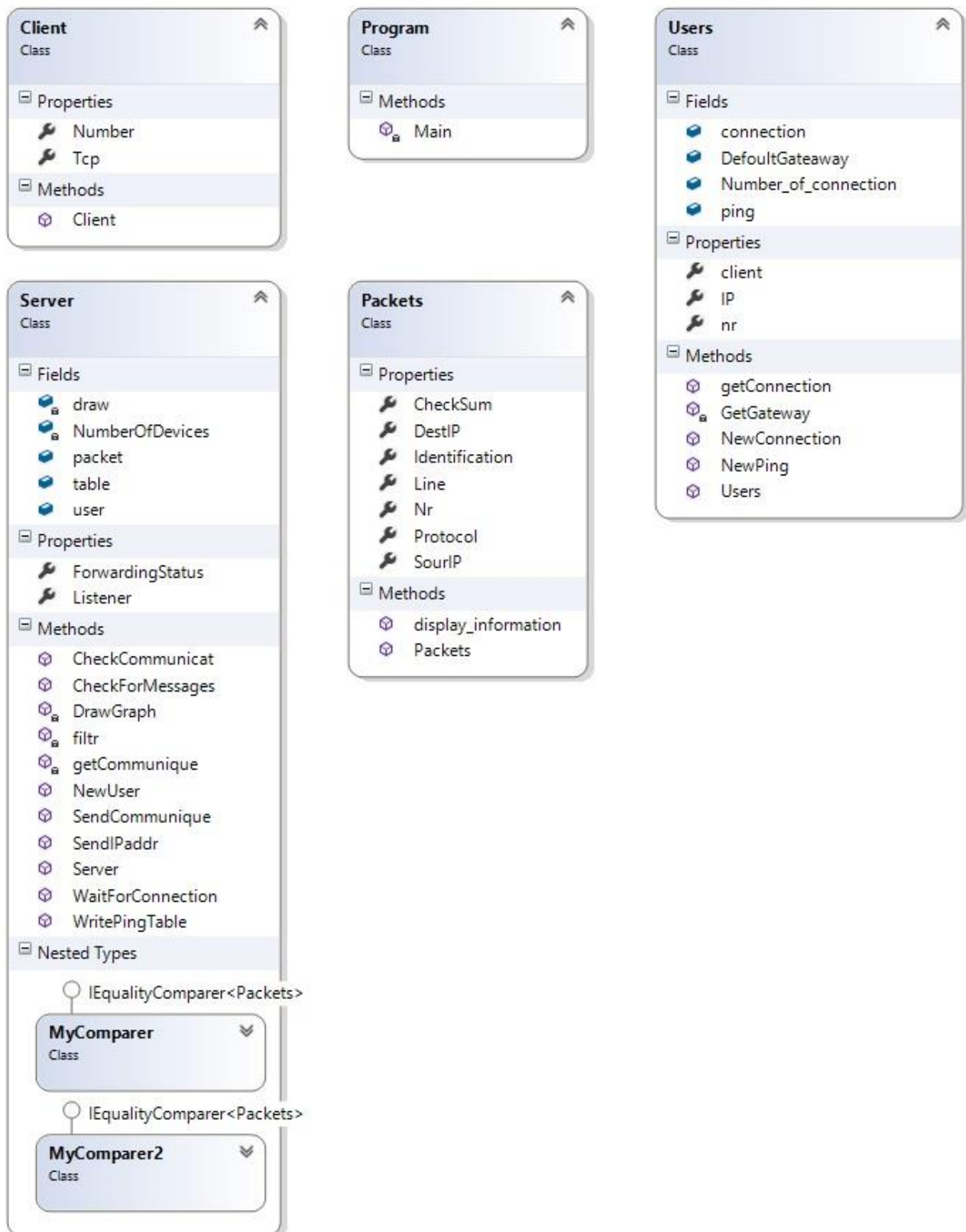


Diagram klas dla aplikacji serwera

## 4. Efekty prac na poszczególnych etapach

W tej sekcji znajdują się wyniki pracy nad projektem. Znajdują się tutaj też testy aplikacji na różnych etapach prac oraz testy finalnej wersji aplikacji.

### 4.1 Iteracja pierwsza

Podczas pierwszej iteracji wybrany został temat projektu oraz ustalono wstępne założenia aplikacji. Grupa zdecydowała, że aplikacja wykonana będzie jako aplikacja konsolowa z wykorzystaniem języka C#, gdyż przesyłane informacje z wielu maszyn oraz ich przetwarzanie mogłyby zająć dużo zasobów na maszynie serwera. Ze względu na optymalizację aplikacji pod względem wydajności i braku wymogu w sprawie warstwy wizualnej wybór ten okazał się najlepszym wyborem. Dodatkowo zdecydowano się na taki rodzaj programu, gdyż program skierowany jest dla osób interesujących się technologiami sieciowymi, a do tego niezbędna jest umiejętność posługiwania się konsolą, zatem program wciąż jest przyjazny dedykowanemu użytkownikowi mimo zastosowaniu konsoli.

Dla ułatwienia pracy zespołowej postanowiono rozdzielić system do architektury klient-serwer, by każdy w grupie był odpowiedzialny za którąś z części. Dodatkowo przyjęto, że mogą pojawić się pewne problemy z którąś z aplikacji i w takim wypadku prace całej grupy skupią się nad jedną.

Założono, że aplikacja kliencka korzystać będzie z programu TCPdump do przechwytywania przesyłanych i odbieranych pakietów, następnie na żądanie serwera informacje te będą do niego przesyłane i tam poddawane analizie.

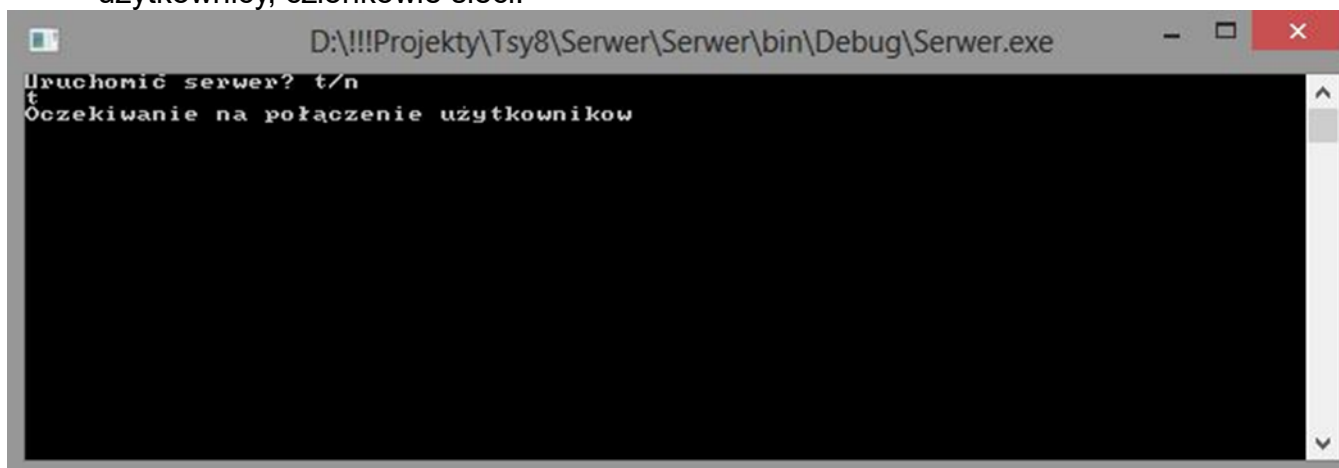
Zaplanowano, że efektem końcowym aplikacji będzie możliwość utworzenia grafu dla topologii sieci, rysowanego na podstawie otrzymanych przez serwer informacji.

## 4.2 Iteracja druga

Druga iteracja przebiegła bez większych problemów. Utworzono aplikację kliencką i serwerową z użyciem socketów dostępnych w bibliotece języka C# w środowisku Visual Studio 2013 oraz udało się nawiązać między nimi połączenie wysyłając przykładowe komunikaty. W następnym etapie w serwerze utworzono możliwość połączenia się z kilkoma klientami jednocześnie i odbiór od nich komunikatów, co w efekcie przypominało chat. Efekt ten został osiągnięty za pomocą wątków i tasków, które są integralną częścią języka C#.

**Poniżej przedstawiono test systemu na koniec drugiej iteracji.**

Uruchomiono aplikację serwera, serwer nasłuchuje i czeka aż połączą się do niego użytkownicy, członkowie sieci.



*Rys. 1 Uruchomienie aplikacji serwera*

Następnie uruchamiamy aplikację klienta, wyświetla się monit o podanie adresu IP serwera.



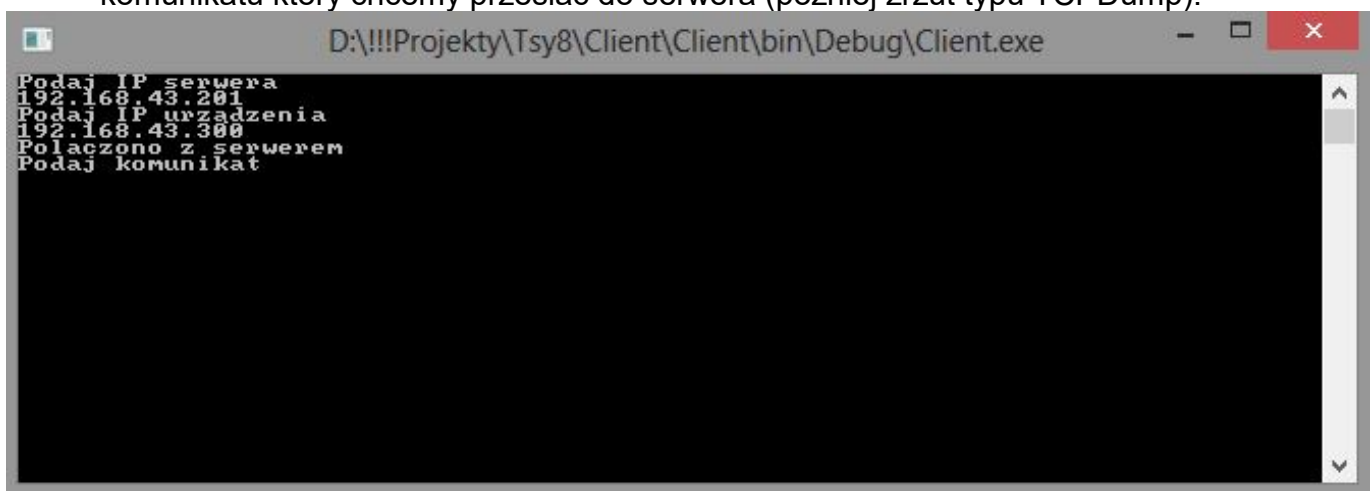
*Rys. 2 Uruchomienie aplikacji klienta*

W aplikacji klienckiej podajemy IP serwera które wcześniej zostało uzgodnione. Zatwierdzamy klawiszem enter.



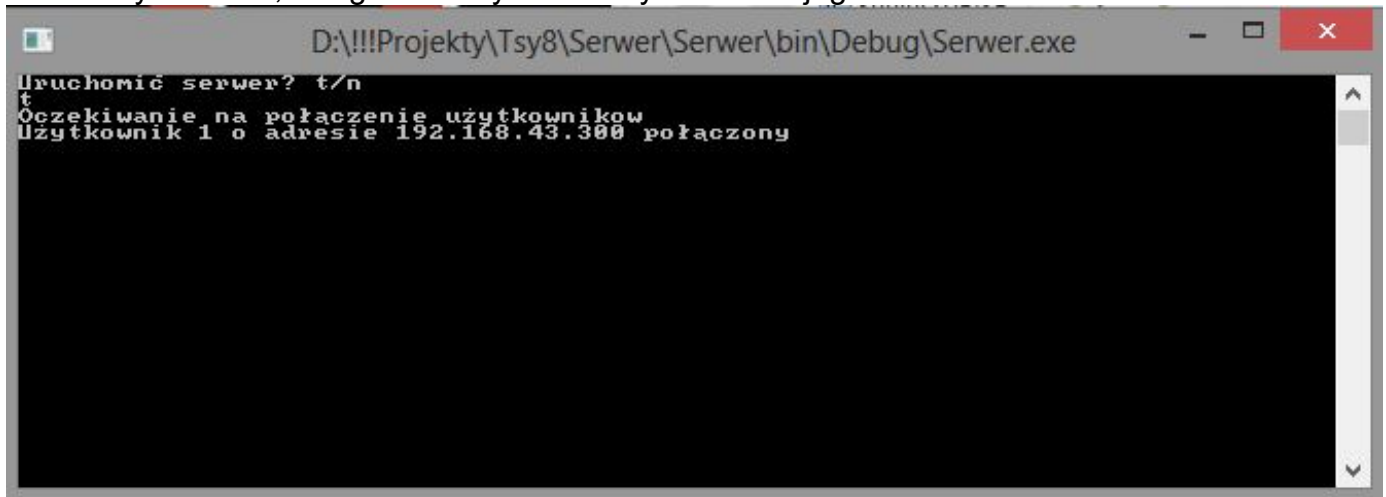
*Rys. 3 Podanie adresu IP serwera*

Podajemy IP urządzenia, jest to IP wpisywane do konsoli i na tym etapie prac może być dowolne, zatwierdzamy klawiszem enter. Wyświetlony zostaje monit o podanie komunikatu który chcemy przesłać do serwera (później zrzut typu TCPDump).



*Rys. 4 Podanie adresu IP klienta*

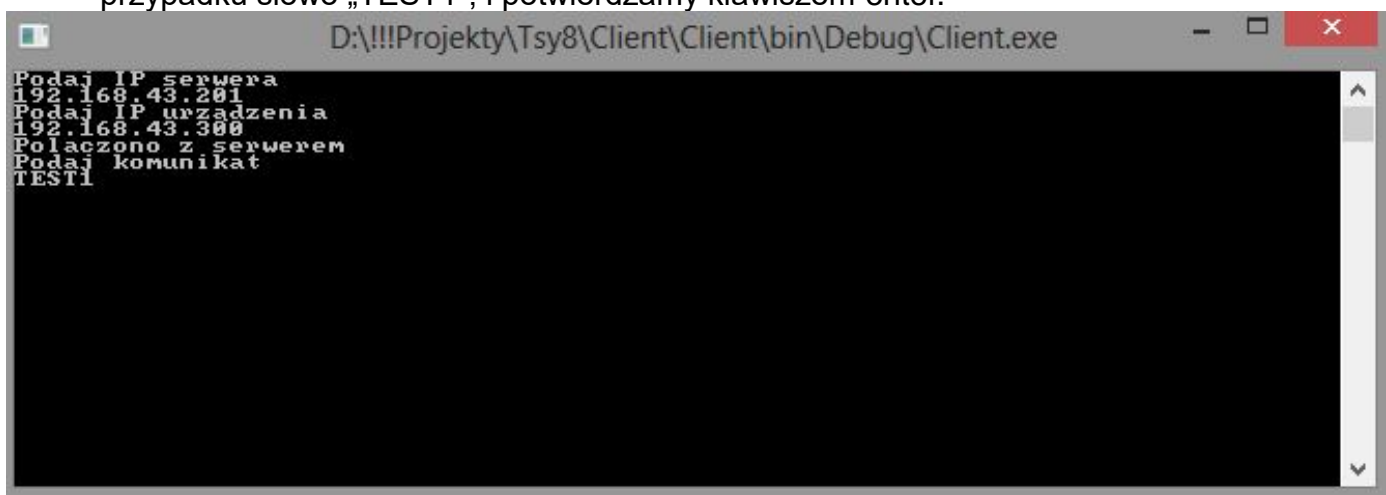
Widzimy że w aplikacji serwera wyświetlony został log o połączeniu się nowego użytkownika, w logu widzimy adres użytkownika i jego status.



```
D:\!!!Projekty\Tsy8\Serwer\Serwer\bin\Debug\Serwer.exe
Uruchomić serwer? t/n
t
Oczekiwanie na połączenie użytkowników
Użytkownik 1 o adresie 192.168.43.300 połączony
```

*Rys. 5 Wyświetlanie przez serwer adresów połączonych klientów*

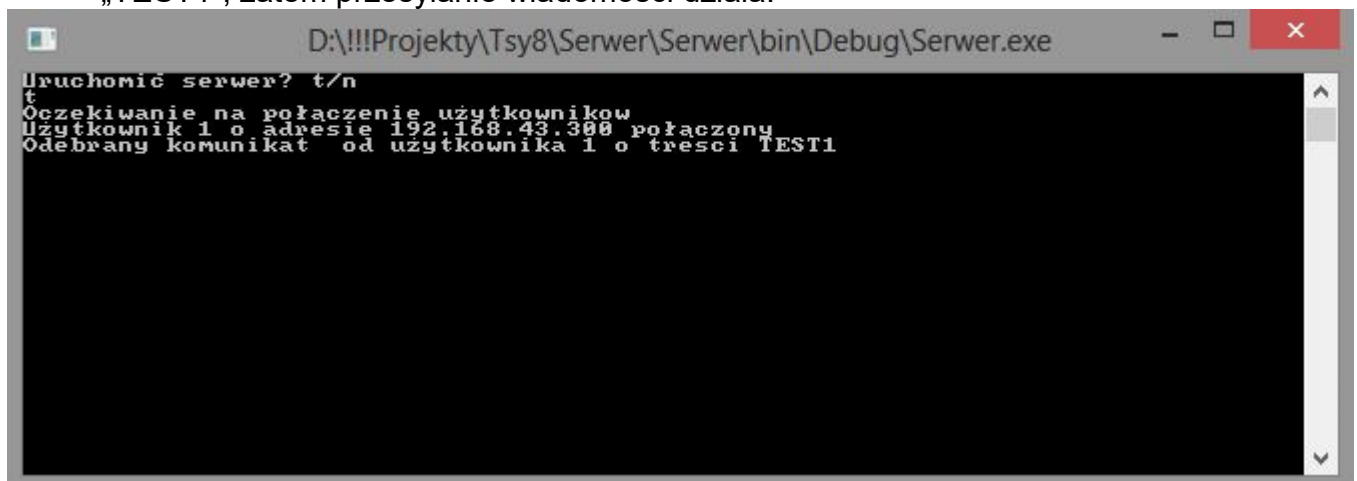
W aplikacji klienckiej wpisujemy komunikat, który chcemy przesłać, w tym przypadku słowo „TEST1”, i potwierdzamy klawiszem enter.



```
D:\!!!Projekty\Tsy8\Client\Client\bin\Debug\Client.exe
Podaj IP serwera
192.168.43.201
Podaj IP urządzenia
192.168.43.300
Połączono z serwerem
Podaj komunikat
TEST1
```

*Rys. 6 Użytkownik wysyła komunikat do serwera*

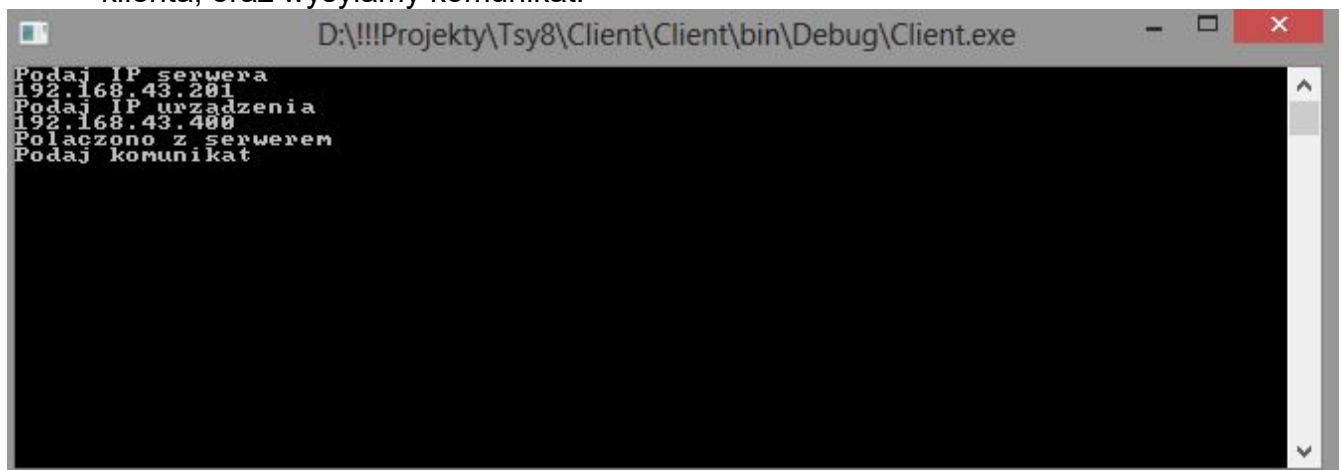
W konsoli aplikacji serwera otrzymujemy komunikat od użytkownika o treści „TEST1”, zatem przesyłanie wiadomości działa.



```
D:\!!!Projekty\Tsy8\Serwer\Serwer\bin\Debug\Serwer.exe
Uruchomić serwer? t/n
t
Oczekiwanie na połączenie użytkowników
Użytkownik 1 o adresie 192.168.43.300 połączony
Odebrany komunikat od użytkownika 1 o treści TEST1
```

*Rys. 7 Serwer odbiera komunikat*

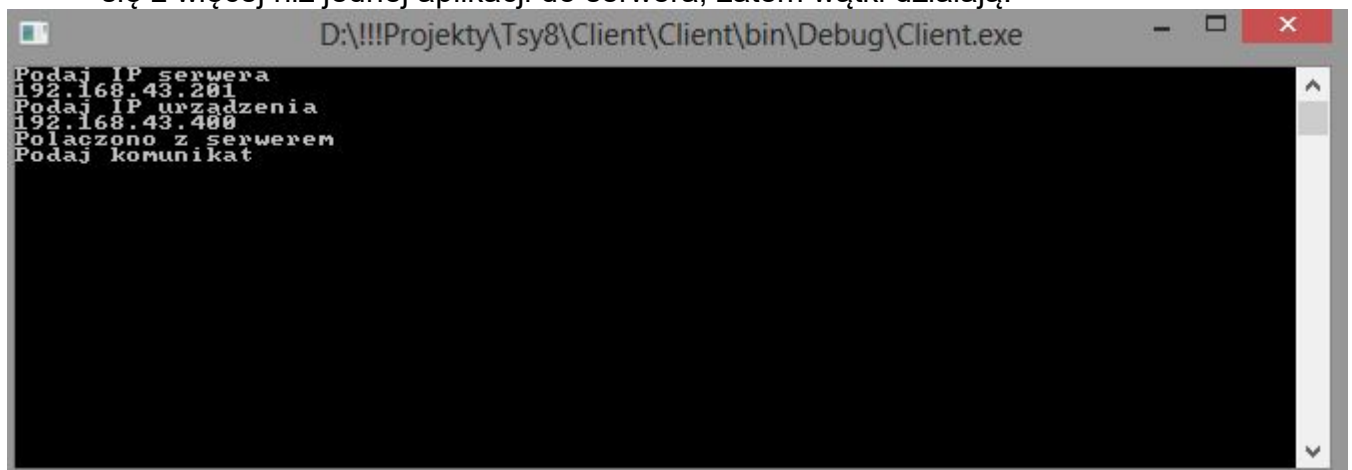
W nowej instancji klienckiej powtarzamy operacje podania IP serwera, podania IP klienta, oraz wysyłamy komunikat.



```
D:\!!!Projekty\Tsy8\Client\Client\bin\Debug\Client.exe
Podaj IP serwera
192.168.43.201
Podaj IP urządzenia
192.168.43.400
Połączono z serwerem
Podaj komunikat
```

*Rys. 8 Uruchamiamy nową aplikację kliencką.*

Nastąpiło połączenie klienta do serwera. Widzimy że możliwe jest połączenie się z więcej niż jednej aplikacji do serwera, zatem wątki działają.



```
D:\!!!Projekty\Tsy8\Client\Client\bin\Debug\Client.exe
Podaj IP serwera
192.168.43.201
Podaj IP urządzenia
192.168.43.400
Połączono z serwerem
Podaj komunikat
```

*Rys. 9 Łączenie klienta do serwera z drugiej maszyny*

Finalnie widzimy dwie aplikacje klienckie i aplikację serwera, w których możemy podawać komunikaty i zostają one odebrane przez serwer.

```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...  
Uruchomić serwer? t/n  
t  
Oczekiwanie na połączenie użytkowników  
Użytkownik 1 o adresie 192.168.43.300 połączony  
Odebrany komunikat od użytkownika 1 o treści TEST1  
Użytkownik 2 o adresie 192.168.43.400 połączony  
Odebrany komunikat od użytkownika 2 o treści TEST2  
Odebrany komunikat od użytkownika 2 o treści TEST3  
Odebrany komunikat od użytkownika 1 o treści TEST4  
  
D:\!!!Projekty\Tsy8\Client\Client\bin\...  
Podaj IP serwera  
192.168.43.201  
Podaj IP urządzenia  
192.168.43.300  
Połączono z serwerem  
Podaj komunikat  
TEST1  
Podaj komunikat  
TEST4  
Podaj komunikat  
  
D:\!!!Projekty\Tsy8\Client\Client\bin\Debug\Client.exe  
Podaj IP serwera  
192.168.43.201  
Podaj IP urządzenia  
192.168.43.400  
Połączono z serwerem  
Podaj komunikat  
TEST2  
Podaj komunikat  
TEST3  
Podaj komunikat
```

Rys. 10 Przesyłane komunikaty odbierane są niezależnie od kolejności ich wysyłania

### 4.3 Iteracja trzecia

Trzeci etap rozpoczął się od usunięcia konieczności podawania adresów IP serwera i klienta w obu aplikacjach. Zamiast tego aplikacja automatycznie pobiera własny adres IP oraz adres routera potrzebny do późniejszego rysowania topologii.

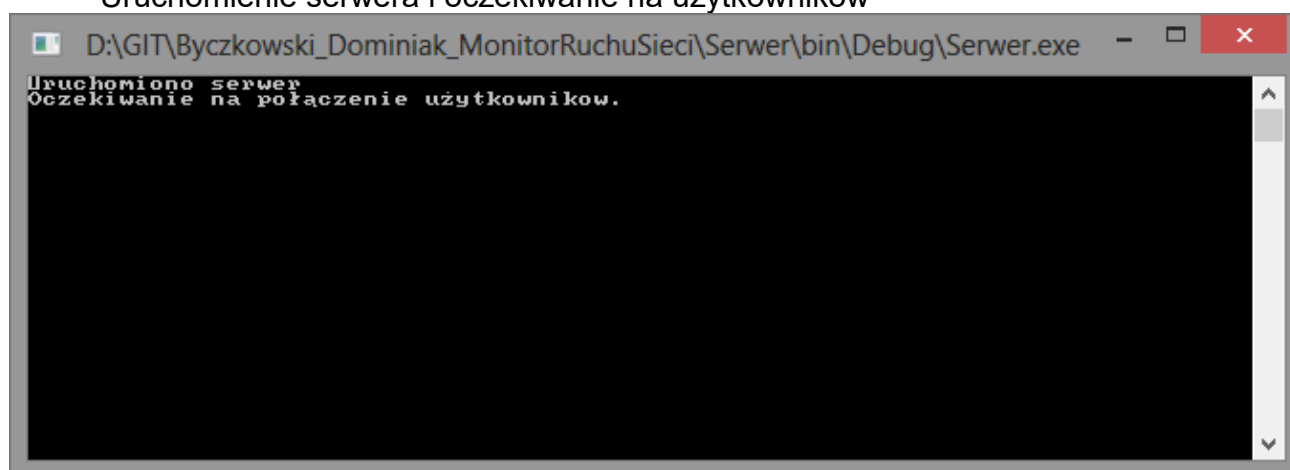
Podczas tej iteracji pojawił się problem, gdyż po zaimplementowaniu w aplikacji klienta obsługi TCPdumpa i próbie uruchomienia jej na kilku komputerach okazało się, że nie każdy komputer umożliwia przechwytywanie pakietów z użyciem tego programu. TCPDump to aplikacja zewnętrzna z tego powodu niemożliwe było pełne zdebugowanie jej. To stworzyło duży problem w stosunku do założeń powstałych podczas etapu planowania projektu. Próbowano zastąpić TCPdump innymi programami, ale nie spełniały one oczekiwań aplikacji, dlatego przechwytywanie pakietów przełożono na dalsze iteracje by nie stwarzać opóźnień w dostarczaniu oprogramowania.

Zgodnie z zasadami SCRUM podjęto decyzję o tymczasowym zastąpieniu TCPdumpa operacją ping, tak aby aplikacja mimo nie spełniania całości założeń, była funkcjonalna i testowalna. Nasłuchiwanie zastąpiono przez próbę nawiązania połączenia między aplikacjami i wykonania między nimi komendy ping. Tablica adresów klientów została pobrana z listy klientów podłączonych do serwera. Aby to umożliwić, serwer po dołączeniu nowego klienta do serwera aktualizował adresy klientów i rozsyłał je wszystkie z żądaniem próby połączenia się na te adresy. Aplikacja każdego z klientów odbierała adresy, zapisywała je w pamięci i próbowała się z tymi adresami łączyć, po czym wysyłając komunikaty informowała czy może połączyć się z nimi czy też nie.

Posiadając te informacje w serwerze można było przystąpić do próby uruchomienia narzędzia GraphViz rysującego topologie. Po kilku podejściach udało się narysować pierwszy, skromny graf generowany automatycznie. Do jego narysowania potrzebny jest plik tekstowy z odpowiednimi komendami, uruchamiany następnie przy pomocy konsoli cmd, co zostało odpowiednio zautomatyzowane w aplikacji serwera.

**Poniżej przedstawiono test systemu na koniec trzeciej iteracji.**

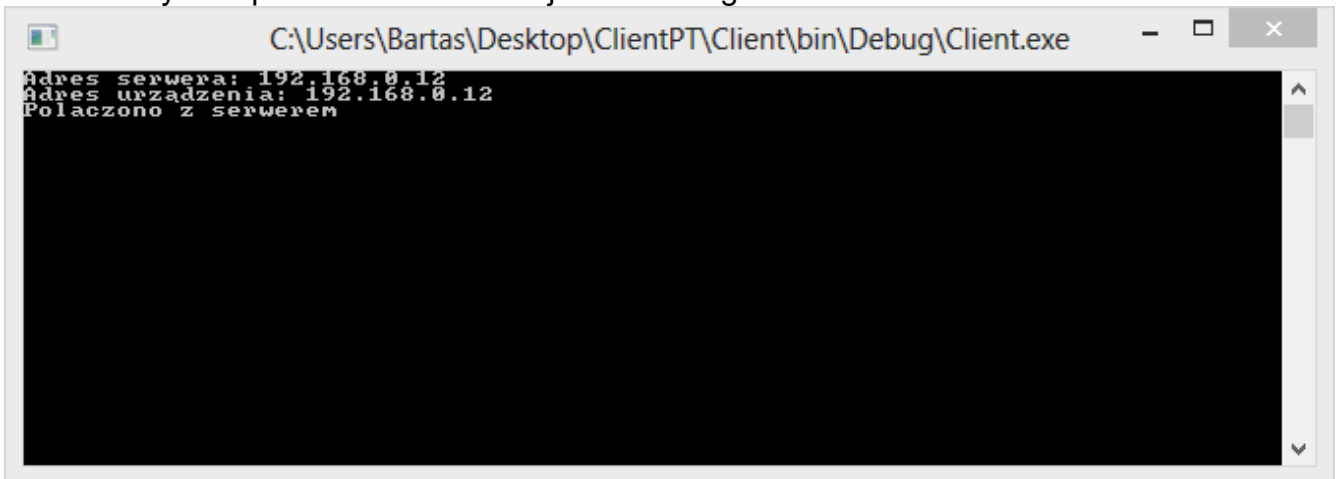
Uruchomienie serwera i oczekiwanie na użytkowników



Rys. 11 Uruchomienie aplikacji serwera- adres serwera pobierany jest automatycznie



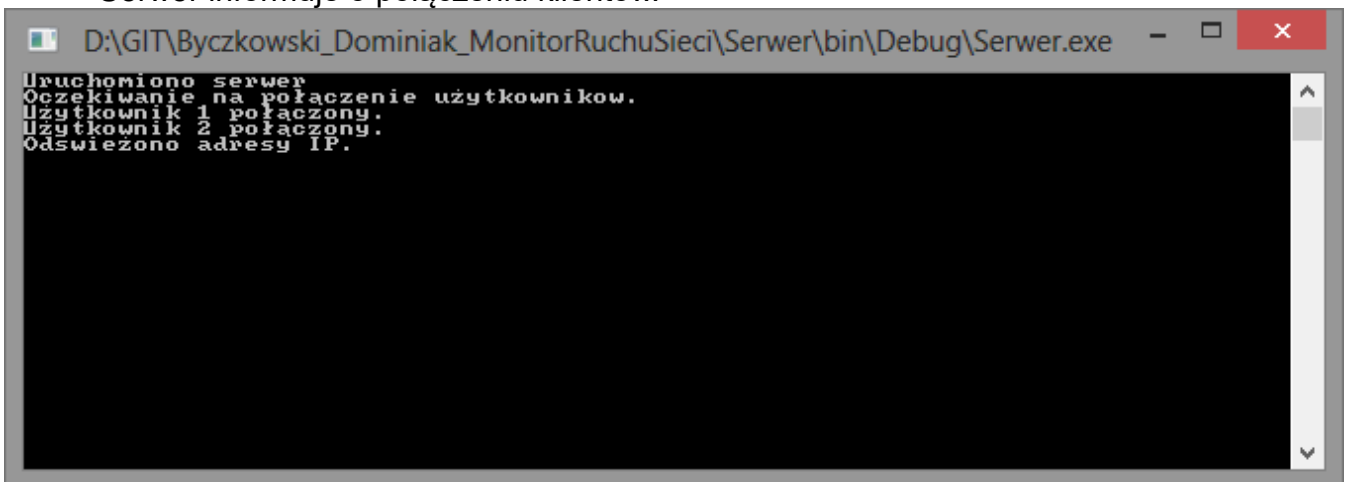
Uruchomienie aplikacji klienta podanie adresu serwera i adresu urządzenia, domyślnie pobranie ich z interfejsu sieciowego.



```
C:\Users\Bartas\Desktop\ClientPT\Client\bin\Debug\Client.exe
Adres serwera: 192.168.0.12
Adres urządzenia: 192.168.0.12
Połączono z serwerem
```

*Rys. 12 Uruchomienie aplikacji klienta - adres klienta pobierany jest automatycznie*

Serwer informuje o połączeniu klientów.



```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
Uruchomiono serwer
Oczekiwanie na połączenie użytkowników.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
```

*Rys. 13 Połączenie dwóch aplikacji do serwera - dodany został nowy adres IP a zaktualizowaną listę adresów przesłano do klientów*

Serwer po podaniu polecenia ping pokazuje wynik polecenia w stosunku do pierwszego jak i drugiego klienta.

```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
Uruchomiono serwer
Oczekiwanie na połączenie użytkownikow.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
ping
Wynik pingowania do urządzeń z klienta nr 1:
192.168.0.12: True
192.168.0.13: False
Wynik pingowania do urządzeń z klienta nr 2:
192.168.0.12: True
192.168.0.13: True
```

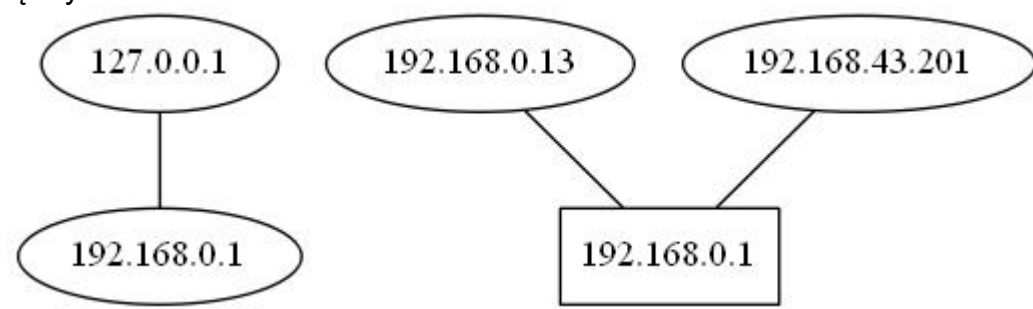
Rys. 14 Wykonanie polecenia ping- w serwerze umożliwia wyświetlenie adresów z którymi połączyć mogą się klienci

Polecenie graph wywołuje okno z grafem prezentującym obecną tablicę działań między klientami i serwerem.

```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
Uruchomiono serwer
Oczekiwanie na połączenie użytkownikow.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
ping
Wynik pingowania do urządzeń z klienta nr 1:
192.168.0.12: True
192.168.0.13: False
Wynik pingowania do urządzeń z klienta nr 2:
192.168.0.12: True
192.168.0.13: True
graph
```

Rys. 15 Wykonanie komendy graph w serwerze

Wynik programu GraphViz, prezentuje adresy klientów i serwera oraz połączenia między nimi.



Rys. 16 Pierwszy wygenerowany graf z użyciem programu GraphViz

#### 4.4 Iteracja czwarta

Kolejna iteracja przyniosła możliwość przechwytywania pakietów. Udało się znaleźć odpowiedni program nasłuchujący pakiety, który jednocześnie dał się przebudować do potrzeb i wymagać projektowych. MJSniffer - jest aplikacją wyświetlającą przechwycone informacje, rodzaj protokołu oraz nagłówki IP. Dla naszych potrzeb zamaskowano jego działanie, a potrzebne informacje zapisane zostały do pliku tekstowego.

Aplikacja kliencka na żądanie serwera otwiera plik tekstowy i zapisuje w pamięci przechwycone pakiety, każdy w innym obiekcie. Następnie filtruje spośród przechwyconych adresów, nadawcy i odbiorcy, te które są unikalne. Następnie całą listę informacji odsyła do serwera. Im więcej danych zostało podsłuchanych tym dłuższy jest czas przesyłania informacji. Stąd odebranie wszystkich pakietów przez serwer zajmuje około 10-60 sekund dla 2-3 klientów.

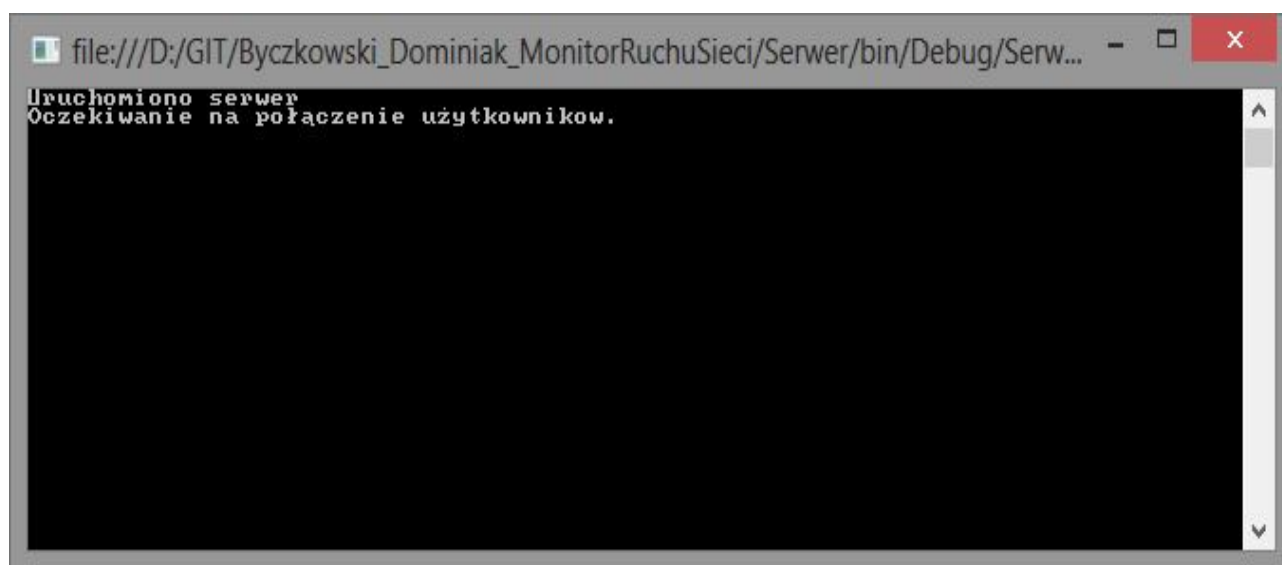
Aby wysłać żądanie nadesłania tych informacji, w aplikacji serwera należy wykonać komendę *dump* i odczekać do momentu, gdy spłyną dane od wszystkich klientów, o czym informują odpowiednie komunikaty.

Otrzymane pakiety można wyświetlić na aplikacji serwera przy pomocy komendy *packets*. Wyświetlenie ich zaraz po otrzymaniu spowoduje, że wystąpi wiele duplikatów np. dlatego, że klient A i klient B komunikują się ze sobą, a więc jeden i drugi przechwycił taki komunikat.

Do pozbycia się duplikatów służy komenda *filtr* która zapisuje jedynie unikalne pakiety, pozostałe następnie usuwając.

W dalszym etapie podczas tej iteracji skupiono się na odpowiednim wyświetleniu topologii. Na podstawie wcześniej wysłanych, unikalnych adresów IP z którymi łączą się klienci, oraz ich bramie domyślnej rysowany jest graf. Bramy domyślne zaznaczane są jako prostokąty. Wykonanie grafu, przy dużych danych nie jest przewidywalne, przez co trudno zaimplementować tę część systemu. Głębsze zapoznanie się z tym tematem i dopracowanie czytelności grafu zaplanowano na ostatnią iterację.

**Poniżej przedstawiono test systemu na koniec czwartej iteracji.**



Rys. 17 Uruchomienie serwera

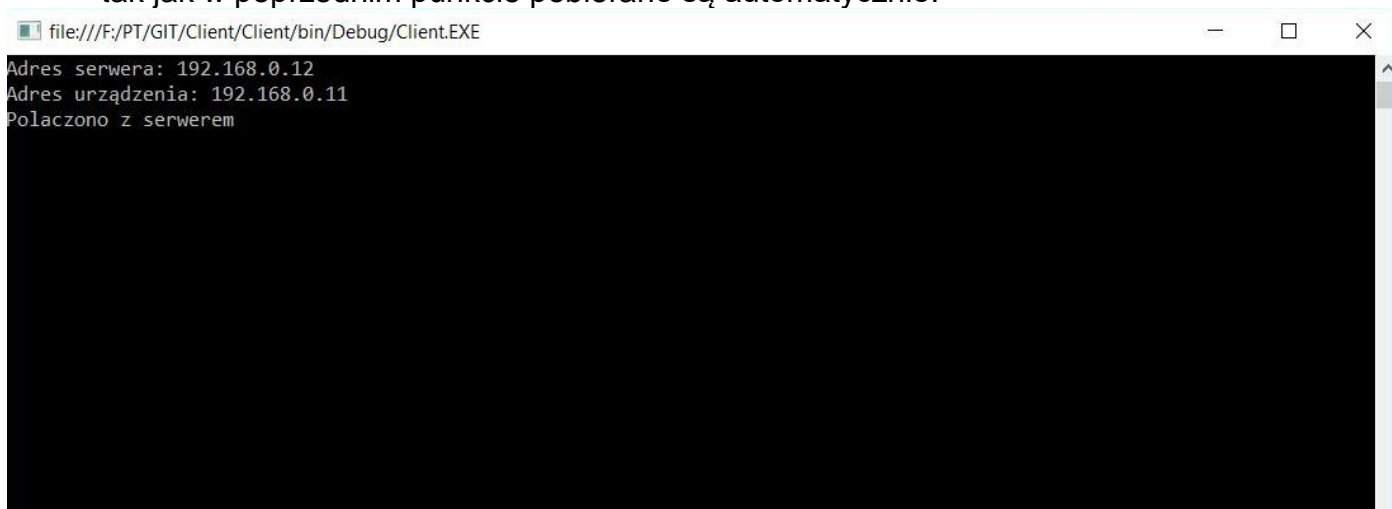
Standardowa operacja uruchomienia klienta, pobranie adresu serwera i urządzenia, oraz połączenia się z serwerem.



```
file:///D:/GIT/Client/Client/bin/Debug/Client.EXE
Adres serwera: 192.168.0.12
Adres urządzenia: 192.168.0.12
Połączono z serwerem
```

Rys. 18 Uruchomienie pierwszego klienta o adresie 192.168.0.12

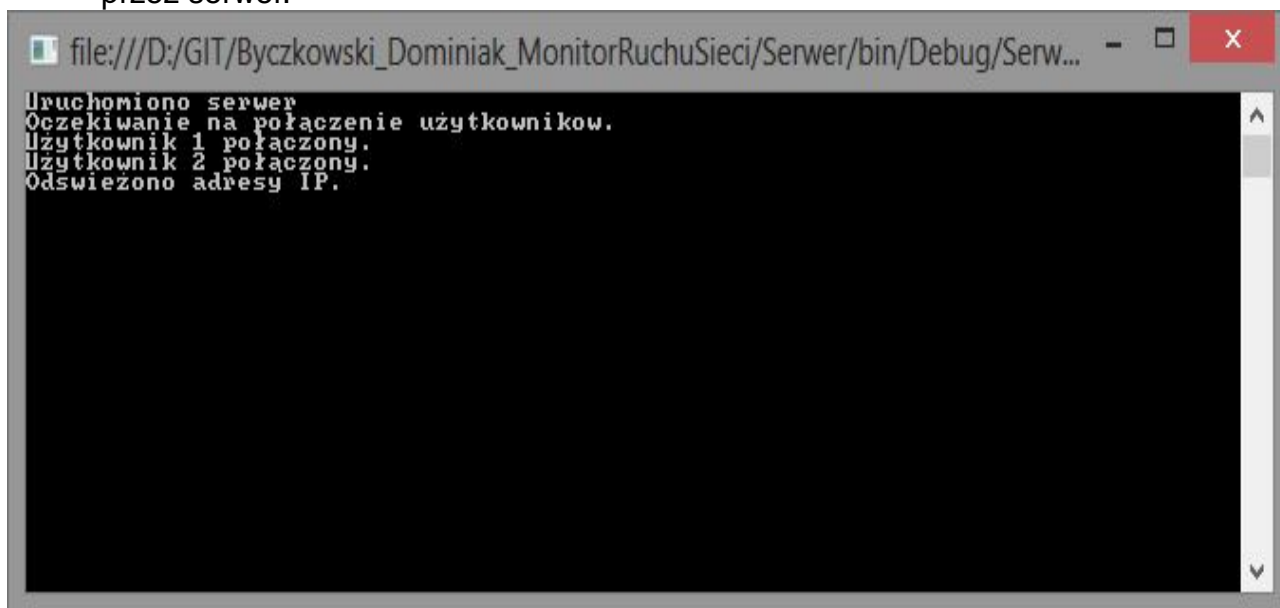
W ten sam sposób uruchomiono aplikację drugiego klienta, o innym adresie. Adresy tak jak w poprzednim punkcie pobierane są automatycznie.



```
file:///F:/PT/GIT/Client/Client/bin/Debug/Client.EXE
Adres serwera: 192.168.0.12
Adres urządzenia: 192.168.0.11
Połączono z serwerem
```

Rys. 19 Uruchomienie drugiego klienta o adresie 192.168.0.11

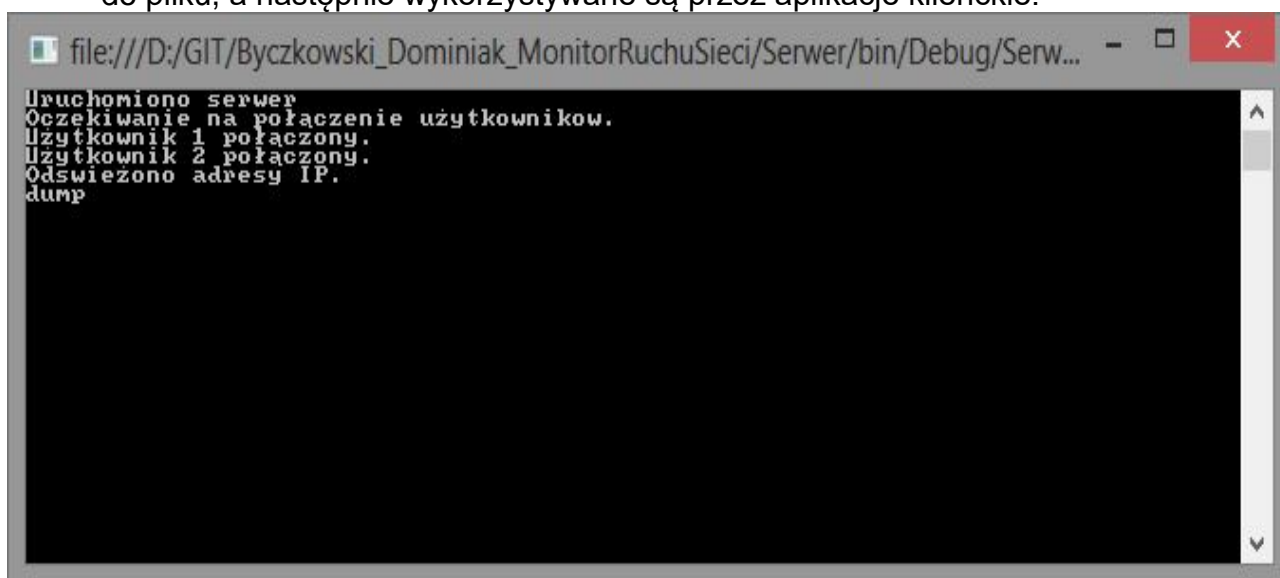
W widoku aplikacji serwera widzimy że nastąpiło poprawne połączenie obu użytkowników. Serwer informuje nas także że odświeżono tablicę adresów IP. W rezultacie adresy IP klientów zostały dopisane do tej tablicy przechowywanej przez serwer.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...  
Uruchomiono serwer  
Oczekiwanie na połączenie użytkowników.  
Użytkownik 1 połączony.  
Użytkownik 2 połączony.  
Odświeżono adresy IP.
```

Rys. 20 Widok serwera po połączeniu z klientami

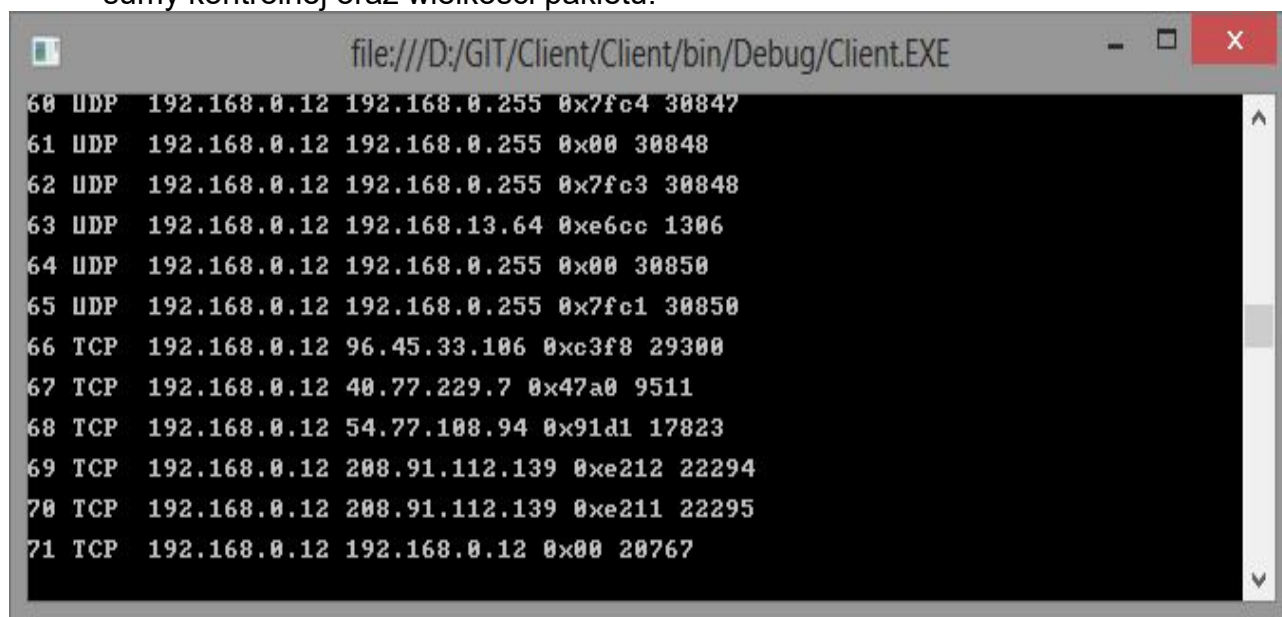
Wywołujemy polecenie dump i czekamy na jego rezultat. W tym momencie w tle uruchamiany jest proces programu MJSniffer, którego wyniki zapisywane są do pliku, a następnie wykorzystywane są przez aplikacje klienckie.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...  
Uruchomiono serwer  
Oczekiwanie na połączenie użytkowników.  
Użytkownik 1 połączony.  
Użytkownik 2 połączony.  
Odświeżono adresy IP.  
dump
```

Rys. 21 Uruchomienie komendy dump

Widzimy efekt wykonania polecenia dump w aplikacjach klienckich. Logi składają się z numeru identyfikacyjnego, rodzaju pakietu, adresu źródłowego i docelowego, sumy kontrolnej oraz wielkości pakietu.

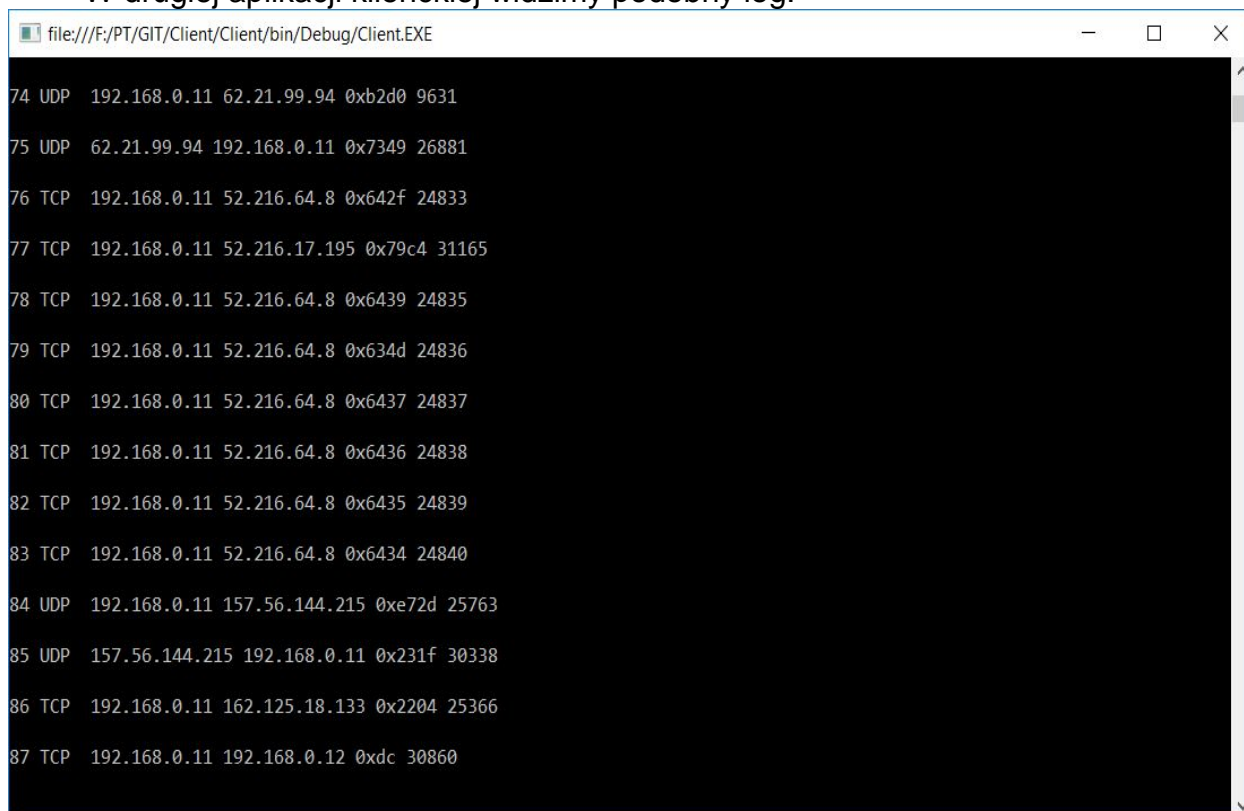


The screenshot shows a Windows command prompt window with the title bar "file:///D:/GIT/Client/Client/bin/Debug/Client.EXE". The window contains a list of network packet logs, numbered 60 to 71. Each log entry consists of a line number, a protocol (UDP or TCP), a source IP address, a destination IP address, a hexadecimal checksum, and a packet size. The logs are displayed in a monospaced font on a black background.

Line	Protocol	Source IP	Destination IP	Checksum	Size
60	UDP	192.168.0.12	192.168.0.255	0x7fc4	30847
61	UDP	192.168.0.12	192.168.0.255	0x00	30848
62	UDP	192.168.0.12	192.168.0.255	0x7fc3	30848
63	UDP	192.168.0.12	192.168.13.64	0xe6cc	1306
64	UDP	192.168.0.12	192.168.0.255	0x00	30850
65	UDP	192.168.0.12	192.168.0.255	0x7fc1	30850
66	TCP	192.168.0.12	96.45.33.106	0xc3f8	29300
67	TCP	192.168.0.12	40.77.229.7	0x47a0	9511
68	TCP	192.168.0.12	54.77.108.94	0x91d1	17823
69	TCP	192.168.0.12	208.91.112.139	0xe212	22294
70	TCP	192.168.0.12	208.91.112.139	0xe211	22295
71	TCP	192.168.0.12	192.168.0.12	0x00	20767

Rys. 22 Efekt wykonania komendy dump w kliencie 1

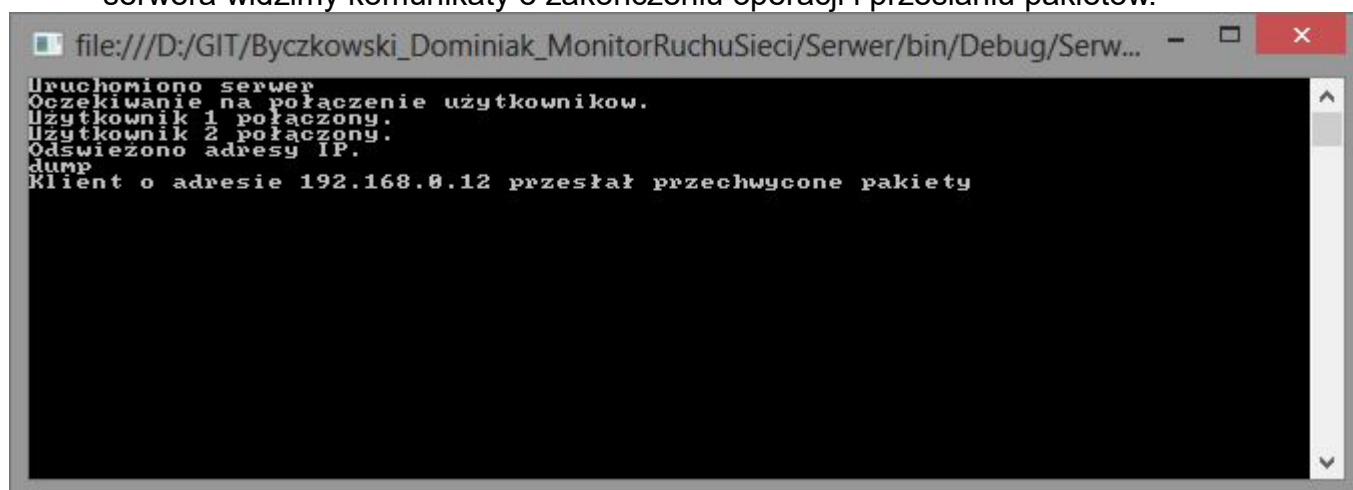
W drugiej aplikacji klienckiej widzimy podobny log.



```
file:///F:/PT/GIT/Client/Client/bin/Debug/Client.EXE
74 UDP 192.168.0.11 62.21.99.94 0xb2d0 9631
75 UDP 62.21.99.94 192.168.0.11 0x7349 26881
76 TCP 192.168.0.11 52.216.64.8 0x642f 24833
77 TCP 192.168.0.11 52.216.17.195 0x79c4 31165
78 TCP 192.168.0.11 52.216.64.8 0x6439 24835
79 TCP 192.168.0.11 52.216.64.8 0x634d 24836
80 TCP 192.168.0.11 52.216.64.8 0x6437 24837
81 TCP 192.168.0.11 52.216.64.8 0x6436 24838
82 TCP 192.168.0.11 52.216.64.8 0x6435 24839
83 TCP 192.168.0.11 52.216.64.8 0x6434 24840
84 UDP 192.168.0.11 157.56.144.215 0xe72d 25763
85 UDP 157.56.144.215 192.168.0.11 0x231f 30338
86 TCP 192.168.0.11 162.125.18.133 0x2204 25366
87 TCP 192.168.0.11 192.168.0.12 0xdc 30860
```

Rys. 23 Efekt wykonania komendy dump w kliencie 2

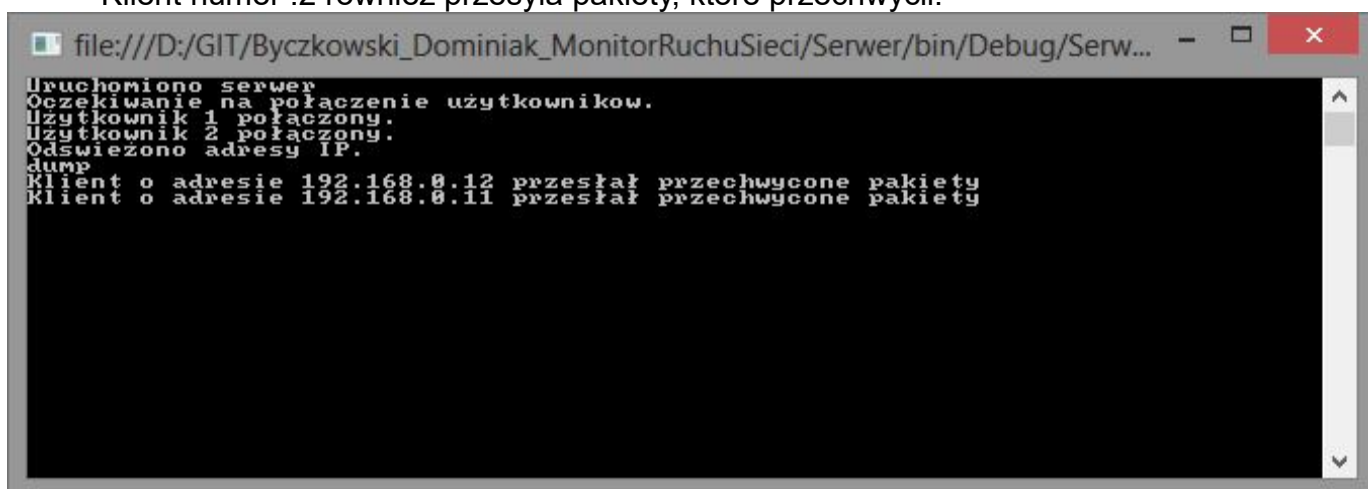
Po wykonaniu polecenia dump, klienci przesyłają wyniki do serwera. W aplikacji serwera widzimy komunikaty o zakończeniu operacji i przesłaniu pakietów.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...
Uruchomiono serwer
Oczekiwanie na połączenie użytkowników.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
dump
Klient o adresie 192.168.0.12 przesłał przechwycone pakiety
```

Rys. 24 Komunikat o otrzymaniu danych od klienta o adresie 192.168.0.12

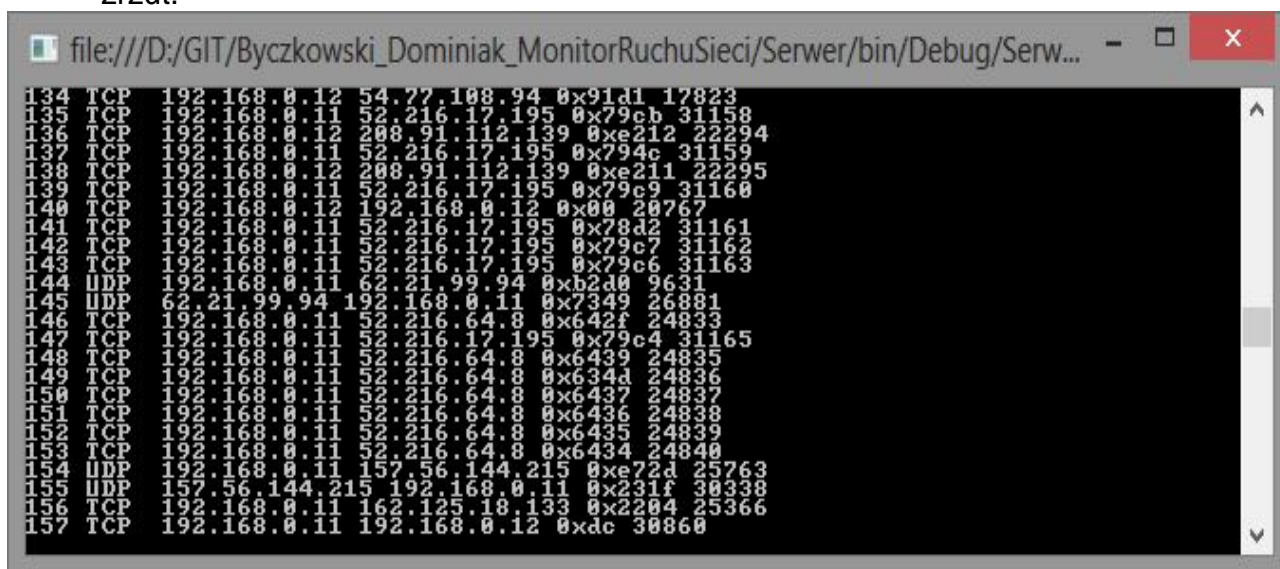
Klient numer .2 również przesyła pakiety, które przechwylił.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...
Uruchomiono serwer
Oczekiwanie na połączenie użytkowników.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
dump
Klient o adresie 192.168.0.12 przesłał przechwycone pakiety
Klient o adresie 192.168.0.11 przesłał przechwycone pakiety
```

Rys. 25 Komunikat o otrzymaniu danych od klienta o adresie 192.168.0.11- po otrzymaniu wszystkich danych można wyświetlić wszystkie informacje

Serwer po odebraniu pakietów od klientów może je wyświetlić. W tym celu należy wprowadzić polecenie *packets*. Efektem tego polecenia jest niżej zamieszczony zrzut.

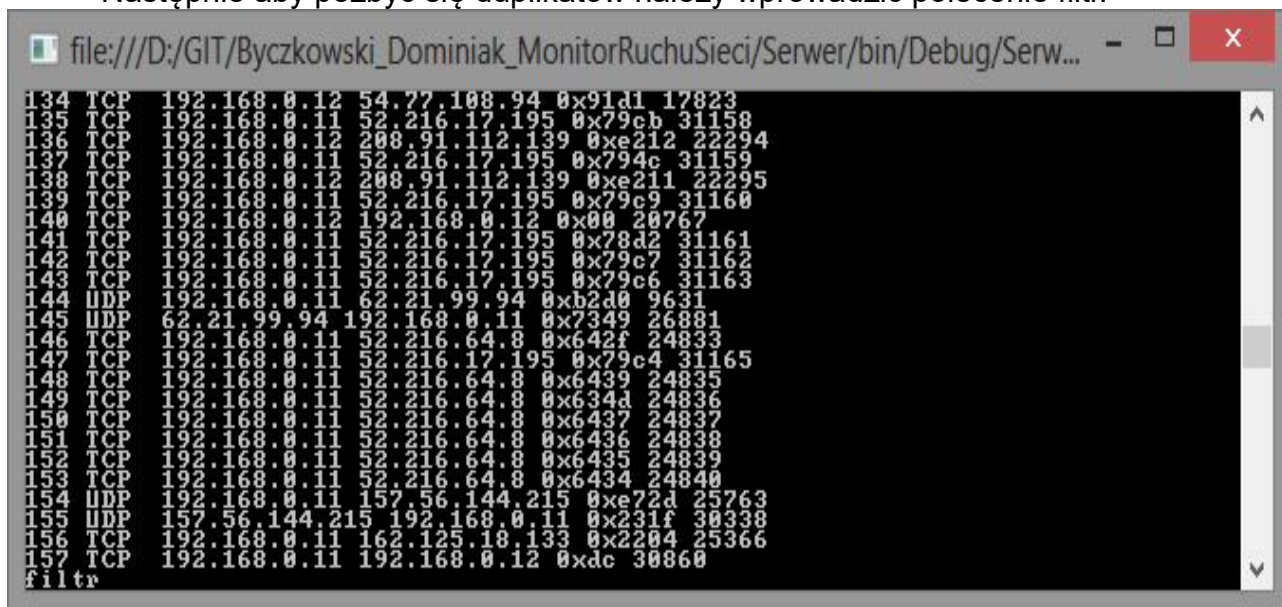


```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...
134 TCP 192.168.0.12 54.77.108.94 0x91d1 17823
135 TCP 192.168.0.11 52.216.17.195 0x79eb 31158
136 TCP 192.168.0.12 208.91.112.139 0xe212 22294
137 TCP 192.168.0.11 52.216.17.195 0x794c 31159
138 TCP 192.168.0.12 208.91.112.139 0xe211 22295
139 TCP 192.168.0.11 52.216.17.195 0x79c9 31160
140 TCP 192.168.0.12 192.168.0.12 0x00 20767
141 TCP 192.168.0.11 52.216.17.195 0x78d2 31161
142 TCP 192.168.0.11 52.216.17.195 0x79c7 31162
143 TCP 192.168.0.11 52.216.17.195 0x79c6 31163
144 UDP 192.168.0.11 62.21.99.94 0xb2d0 9631
145 UDP 62.21.99.94 192.168.0.11 0x7349 26881
146 TCP 192.168.0.11 52.216.64.8 0x642f 24833
147 TCP 192.168.0.11 52.216.17.195 0x79c4 31165
148 TCP 192.168.0.11 52.216.64.8 0x6439 24835
149 TCP 192.168.0.11 52.216.64.8 0x634d 24836
150 TCP 192.168.0.11 52.216.64.8 0x6437 24837
151 TCP 192.168.0.11 52.216.64.8 0x6436 24838
152 TCP 192.168.0.11 52.216.64.8 0x6435 24839
153 TCP 192.168.0.11 52.216.64.8 0x6434 24840
154 UDP 192.168.0.11 157.56.144.215 0xe72d 25763
155 UDP 157.56.144.215 192.168.0.11 0x231f 30338
156 TCP 192.168.0.11 162.125.18.133 0x2204 25366
157 TCP 192.168.0.11 192.168.0.12 0xdc 30860
```

Rys. 26 Wyświetlenie odebranych pakietów przez serwer zaraz po odebraniu pakietów od klientów



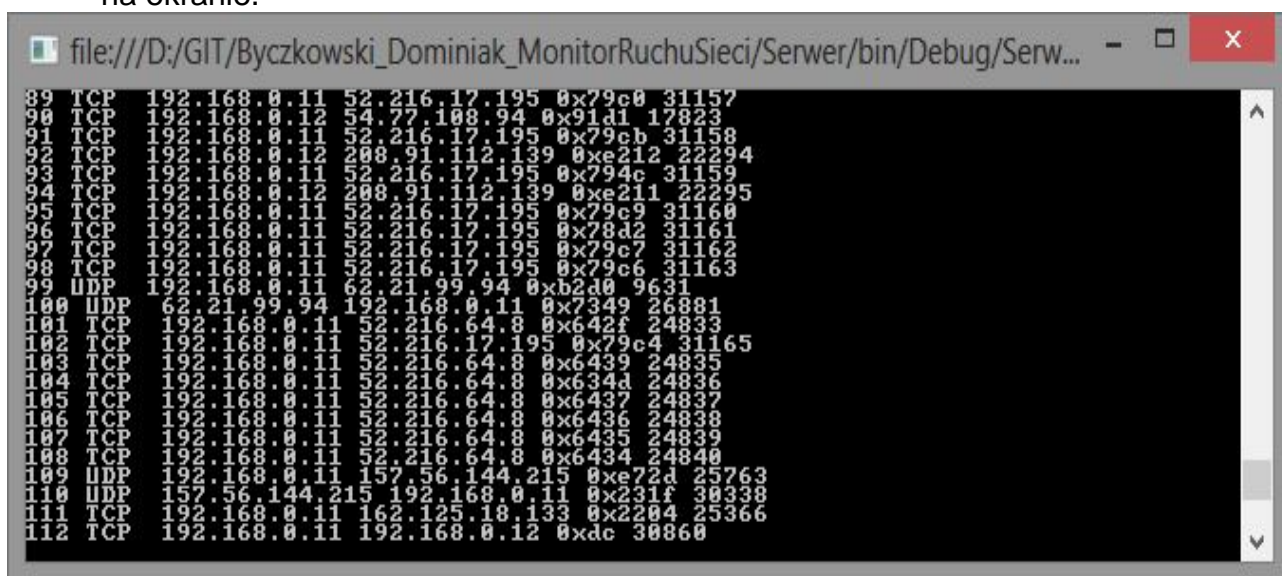
Następnie aby pozbyć się duplikatów należy wprowadzić polecenie filtr.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw... - □ ×
134 TCP 192.168.0.12 54.77.108.94 0x91d1 17823
135 TCP 192.168.0.11 52.216.17.195 0x79cb 31158
136 TCP 192.168.0.12 208.91.112.139 0xe212 22294
137 TCP 192.168.0.11 52.216.17.195 0x794c 31159
138 TCP 192.168.0.12 208.91.112.139 0xe211 22295
139 TCP 192.168.0.11 52.216.17.195 0x79c9 31160
140 TCP 192.168.0.12 192.168.0.12 0x00 20767
141 TCP 192.168.0.11 52.216.17.195 0x78d2 31161
142 TCP 192.168.0.11 52.216.17.195 0x79c7 31162
143 TCP 192.168.0.11 52.216.17.195 0x79c6 31163
144 UDP 192.168.0.11 62.21.99.94 0xb2d0 9631
145 UDP 62.21.99.94 192.168.0.11 0x7349 26881
146 TCP 192.168.0.11 52.216.64.8 0x642f 24833
147 TCP 192.168.0.11 52.216.17.195 0x79c4 31165
148 TCP 192.168.0.11 52.216.64.8 0x6439 24835
149 TCP 192.168.0.11 52.216.64.8 0x634d 24836
150 TCP 192.168.0.11 52.216.64.8 0x6437 24837
151 TCP 192.168.0.11 52.216.64.8 0x6436 24838
152 TCP 192.168.0.11 52.216.64.8 0x6435 24839
153 TCP 192.168.0.11 52.216.64.8 0x6434 24840
154 UDP 192.168.0.11 157.56.144.215 0xe72d 25763
155 UDP 157.56.144.215 192.168.0.11 0x231f 30338
156 TCP 192.168.0.11 162.125.18.133 0x2204 25366
157 TCP 192.168.0.11 192.168.0.12 0xdc 30860
filtr
```

Rys. 27 Wykonanie komendy filtr. Wyciągającej z odebranych pakietów tylko te unikalne

Komenda filtr usuwa duplikaty oraz wyświetla przefiltrowaną listę pakietów na ekranie.



```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw... - □ ×
89 TCP 192.168.0.11 52.216.17.195 0x79c0 31157
90 TCP 192.168.0.12 54.77.108.94 0x91d1 17823
91 TCP 192.168.0.11 52.216.17.195 0x79cb 31158
92 TCP 192.168.0.12 208.91.112.139 0xe212 22294
93 TCP 192.168.0.11 52.216.17.195 0x794c 31159
94 TCP 192.168.0.12 208.91.112.139 0xe211 22295
95 TCP 192.168.0.11 52.216.17.195 0x79c9 31160
96 TCP 192.168.0.11 52.216.17.195 0x78d2 31161
97 TCP 192.168.0.11 52.216.17.195 0x79c7 31162
98 TCP 192.168.0.11 52.216.17.195 0x79c6 31163
99 UDP 192.168.0.11 62.21.99.94 0xb2d0 9631
100 UDP 62.21.99.94 192.168.0.11 0x7349 26881
101 TCP 192.168.0.11 52.216.64.8 0x642f 24833
102 TCP 192.168.0.11 52.216.17.195 0x79c4 31165
103 TCP 192.168.0.11 52.216.64.8 0x6439 24835
104 TCP 192.168.0.11 52.216.64.8 0x634d 24836
105 TCP 192.168.0.11 52.216.64.8 0x6437 24837
106 TCP 192.168.0.11 52.216.64.8 0x6436 24838
107 TCP 192.168.0.11 52.216.64.8 0x6435 24839
108 TCP 192.168.0.11 52.216.64.8 0x6434 24840
109 UDP 192.168.0.11 157.56.144.215 0xe72d 25763
110 UDP 157.56.144.215 192.168.0.11 0x231f 30338
111 TCP 192.168.0.11 162.125.18.133 0x2204 25366
112 TCP 192.168.0.11 192.168.0.12 0xdc 30860
```

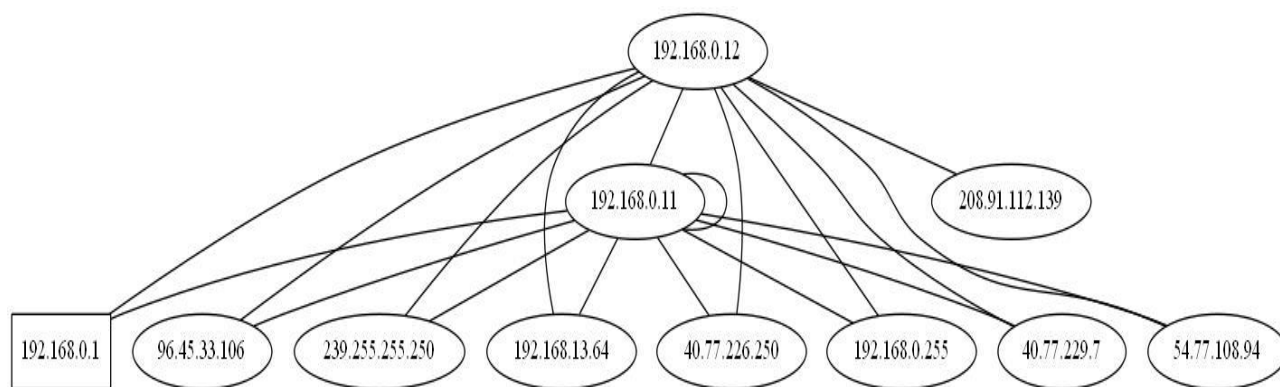
Rys. 28 Efektem jest zmniejszenie liczby przechowywanych pakietów w serwerze

Następnie wprowadzamy polecenie *graph*, które uruchamia narzędzie GraphViz, które rysuje połączenia między klientami.

```
file:///D:/GIT/Byczkowski_Dominiak_MonitorRuchuSieci/Serwer/bin/Debug/Serw...
89 TCP 192.168.0.11 52.216.17.195 0x79c0 31157
90 TCP 192.168.0.12 54.77.108.94 0x91d1 17823
91 TCP 192.168.0.11 52.216.17.195 0x79cb 31158
92 TCP 192.168.0.12 208.91.112.139 0xe212 22294
93 TCP 192.168.0.11 52.216.17.195 0x794c 31159
94 TCP 192.168.0.12 208.91.112.139 0xe211 22295
95 TCP 192.168.0.11 52.216.17.195 0x79c9 31160
96 TCP 192.168.0.11 52.216.17.195 0x78d2 31161
97 TCP 192.168.0.11 52.216.17.195 0x79c7 31162
98 TCP 192.168.0.11 52.216.17.195 0x79c6 31163
99 UDP 192.168.0.11 62.21.99.94 0xb2d0 9631
100 UDP 62.21.99.94 192.168.0.11 0x7349 26881
101 TCP 192.168.0.11 52.216.64.8 0x642f 24833
102 TCP 192.168.0.11 52.216.17.195 0x79c4 31165
103 TCP 192.168.0.11 52.216.64.8 0x6439 24835
104 TCP 192.168.0.11 52.216.64.8 0x634d 24836
105 TCP 192.168.0.11 52.216.64.8 0x6437 24837
106 TCP 192.168.0.11 52.216.64.8 0x6436 24838
107 TCP 192.168.0.11 52.216.64.8 0x6435 24839
108 TCP 192.168.0.11 52.216.64.8 0x6434 24840
109 UDP 192.168.0.11 157.56.144.215 0xe72d 25763
110 UDP 157.56.144.215 192.168.0.11 0x231f 30338
111 TCP 192.168.0.11 162.125.18.133 0x2204 25366
112 TCP 192.168.0.11 192.168.0.12 0xdc 30860
graph
```

Rys. 29 Wykonanie komendy *graph* w serwerze

Efekt działania komendy *graph* przedstawia się następująco. Jest to jeszcze nie zoptymalizowana wersja grafu. W następnych iteracjach podjęty zostanie ten problem.



Rys. 30. Otrzymany graf na podstawie posiadanych informacji

## 4.5 Iteracja piąta

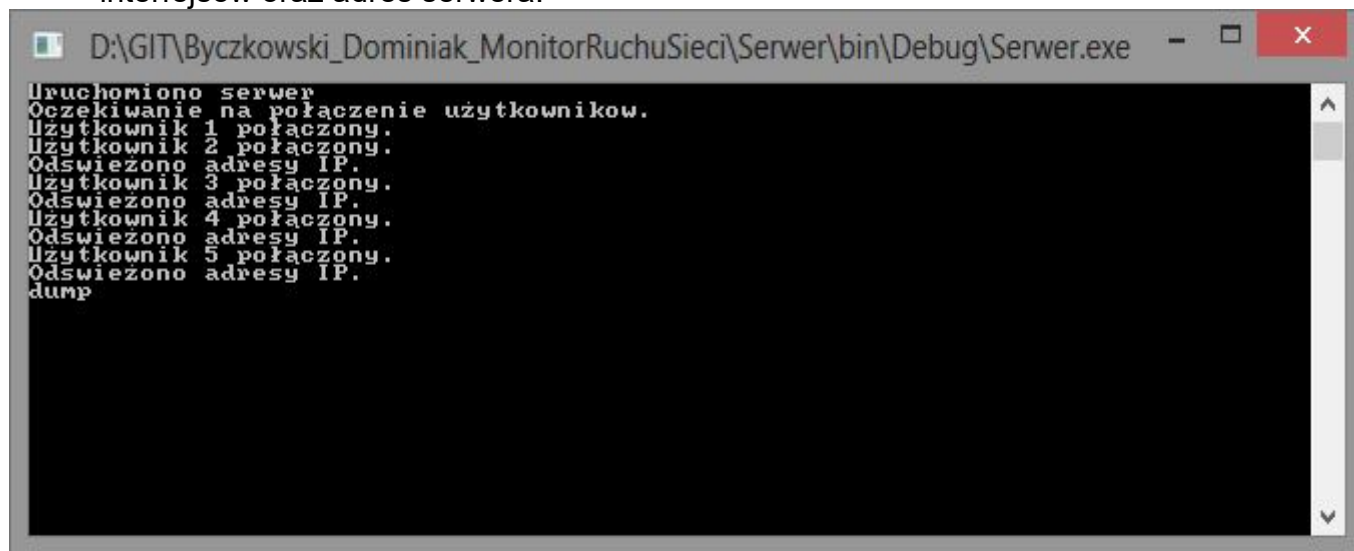
Ostatnia faza projektu skupiała się na testowaniu aplikacji na większej ilości urządzeń oraz na poprawie wyglądu generowanego grafu. Pierwszy etap okazał się być czasochłonny. Uruchomiono aplikacje klienckie na pięciu odrębnych maszynach, w tym na dwóch wirtualnych. Przy każdej próbie trzeba było uruchomić aplikacje, odczekać na przesłanie pakietów do serwera, co łącznie dla wszystkich urządzeń zajmowało od jednej do trzech minut. Następnie trzeba było wyłączać każdą aplikację z osobna i ponawiać proces. Za każdym razem sprawdzany był generowany graf.

GraphViz wydawał się być prostym narzędziem, który w łatwy sposób miał generować grafy. Założenie to sprawdzało się jednak tylko dla małej ilości wierzchołków grafu. Im więcej wierzchołków tym bardziej skomplikowany był problem i tym bardziej nieprzewidywalny był rezultat w postaci grafu. Próbowano manipulować informacjami wpisywanymi do pliku tekstowego na różne sposoby i żadna z prób nie przyniosła rezultatów jakie oczekiwano.

Zakładano, że routery znajdują się u góry grafu, adresy klientów na środku, natomiast wszystkie pozostałe na dole. Niestety graf stał się nie do opanowania przy dużej sieci połączeń. Dlatego zaprzestano ingerencji w położenie węzłów i oznaczono te najważniejsze kolorami dla zwiększenia czytelności. Efekt wydaje się być najlepszy w stosunku do pozostałych prób.

**Poniżej przedstawiono test systemu na koniec piątej iteracji oraz kilka z wygenerowanych grafów.**

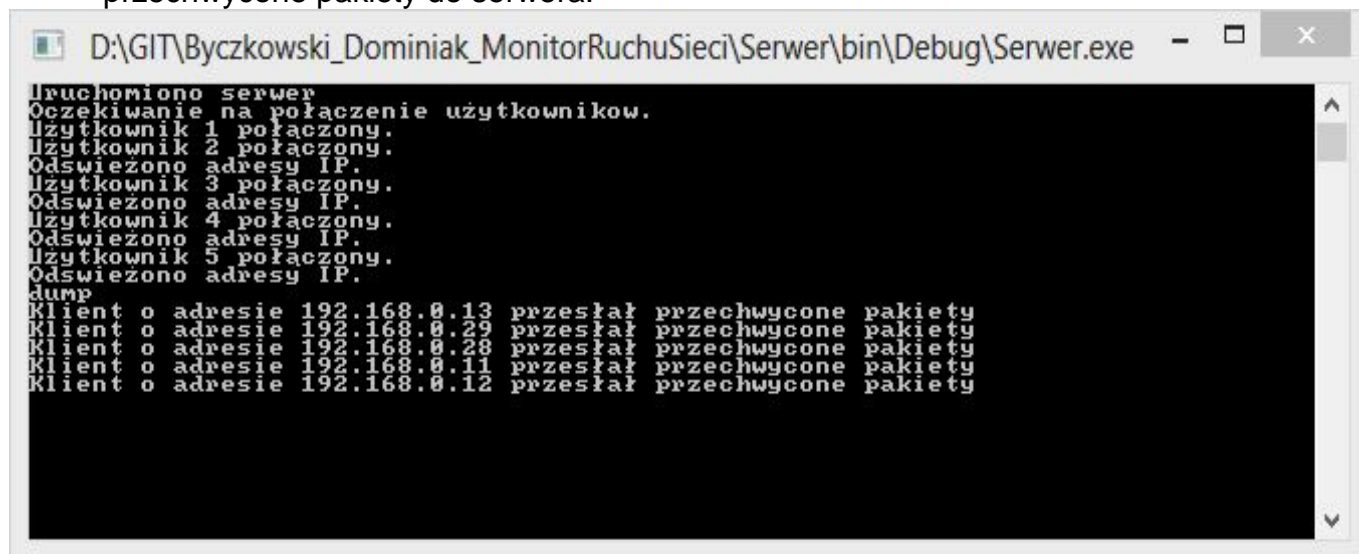
Połączenie się klientów do serwera oraz wykonanie komendy dump. Klienci łączą się w obecnej iteracji z pięciu różnych maszyn, pobierają adresy IP z własnych interfejsów oraz adres serwera.



```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
Uruchomiono serwer
Oczekiwanie na połączenie użytkowników.
Użytkownik 1 połączony.
Użytkownik 2 połączony.
Odświeżono adresy IP.
Użytkownik 3 połączony.
Odświeżono adresy IP.
Użytkownik 4 połączony.
Odświeżono adresy IP.
Użytkownik 5 połączony.
Odświeżono adresy IP.
dump
```

Rys. 31. Uruchomienie serwera i połączenie do niego pięciu aplikacji klienckich

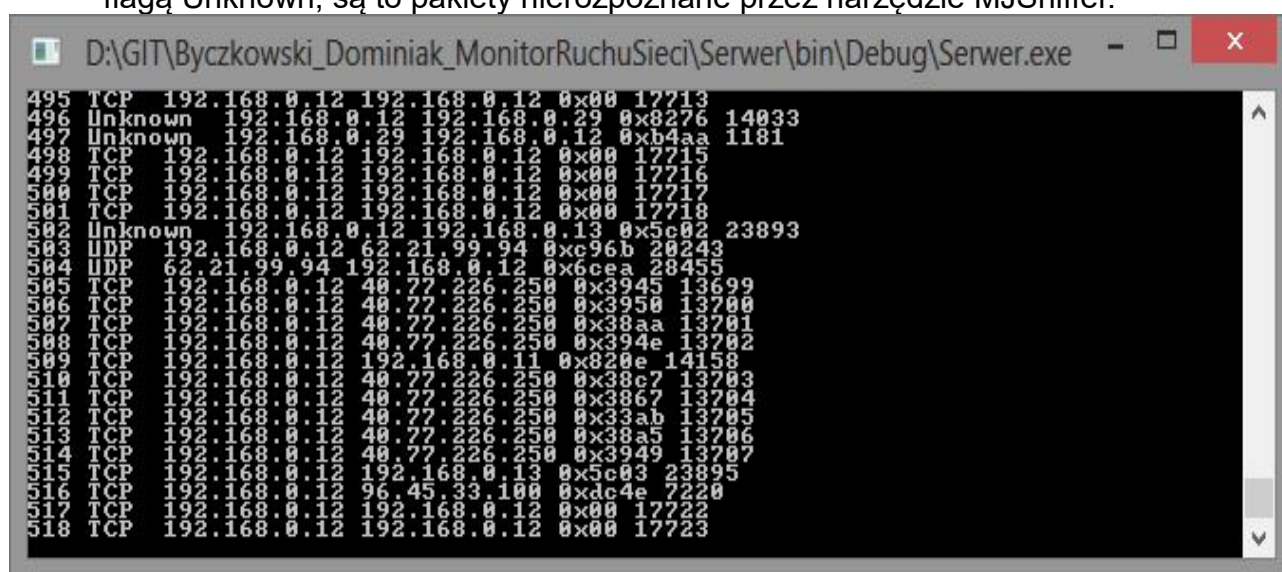
Po pewnym czasie, komenda dump została wykonana. Klienci przesyłają przechwycone pakiety do serwera.



```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
Uruchomiono serwer
Oczekiwanie na połączenie uzytkownikow.
Uzytkownik 1 połączony.
Uzytkownik 2 połączony.
Odswieazono adresy IP.
Uzytkownik 3 połączony.
Odswieazono adresy IP.
Uzytkownik 4 połączony.
Odswieazono adresy IP.
Uzytkownik 5 połączony.
Odswieazono adresy IP.
dump
Klient o adresie 192.168.0.13 przesłał przechwycone pakiety
Klient o adresie 192.168.0.29 przesłał przechwycone pakiety
Klient o adresie 192.168.0.28 przesłał przechwycone pakiety
Klient o adresie 192.168.0.11 przesłał przechwycone pakiety
Klient o adresie 192.168.0.12 przesłał przechwycone pakiety
```

Rys. 32. Komunikaty o otrzymaniu przechwyconych pakietów od klientów

Lista wszystkich pakietów przesłanych do serwera w liczbie 518. Lista ta zawiera duplikaty, które w następnym kroku należy usunąć. Widzimy też pakiety oznaczone flagą Unknown, są to pakiety nierozpoznane przez narzędzie MJSniffer.



```
D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
495 TCP 192.168.0.12 192.168.0.12 0x00 17713
496 Unknown 192.168.0.12 192.168.0.29 0x8276 14033
497 Unknown 192.168.0.29 192.168.0.12 0xb4aa 1181
498 TCP 192.168.0.12 192.168.0.12 0x00 17715
499 TCP 192.168.0.12 192.168.0.12 0x00 17716
500 TCP 192.168.0.12 192.168.0.12 0x00 17717
501 TCP 192.168.0.12 192.168.0.12 0x00 17718
502 Unknown 192.168.0.12 192.168.0.13 0x5c02 23893
503 UDP 192.168.0.12 62.21.99.94 0xc96b 20243
504 UDP 62.21.99.94 192.168.0.12 0x6cea 28455
505 TCP 192.168.0.12 40.77.226.250 0x3945 13699
506 TCP 192.168.0.12 40.77.226.250 0x3950 13700
507 TCP 192.168.0.12 40.77.226.250 0x38aa 13701
508 TCP 192.168.0.12 40.77.226.250 0x394e 13702
509 TCP 192.168.0.12 192.168.0.11 0x820e 14158
510 TCP 192.168.0.12 40.77.226.250 0x38c7 13703
511 TCP 192.168.0.12 40.77.226.250 0x3867 13704
512 TCP 192.168.0.12 40.77.226.250 0x33ab 13705
513 TCP 192.168.0.12 40.77.226.250 0x38a5 13706
514 TCP 192.168.0.12 40.77.226.250 0x3949 13707
515 TCP 192.168.0.12 192.168.0.13 0x5c03 23895
516 TCP 192.168.0.12 96.45.33.100 0xdc4e 7220
517 TCP 192.168.0.12 192.168.0.12 0x00 17722
518 TCP 192.168.0.12 192.168.0.12 0x00 17723
```

Rys. 33. Wyświetlenie pakietów po ich odebraniu i zapisaniu na serwer



Po użyciu komendy filtr, liczba pakietów zmniejszyła się znacząco. Użycie tej komendy jest niezbędne dla poprawnego wyrysowania grafu oraz uniknięcia nadmiarowości.

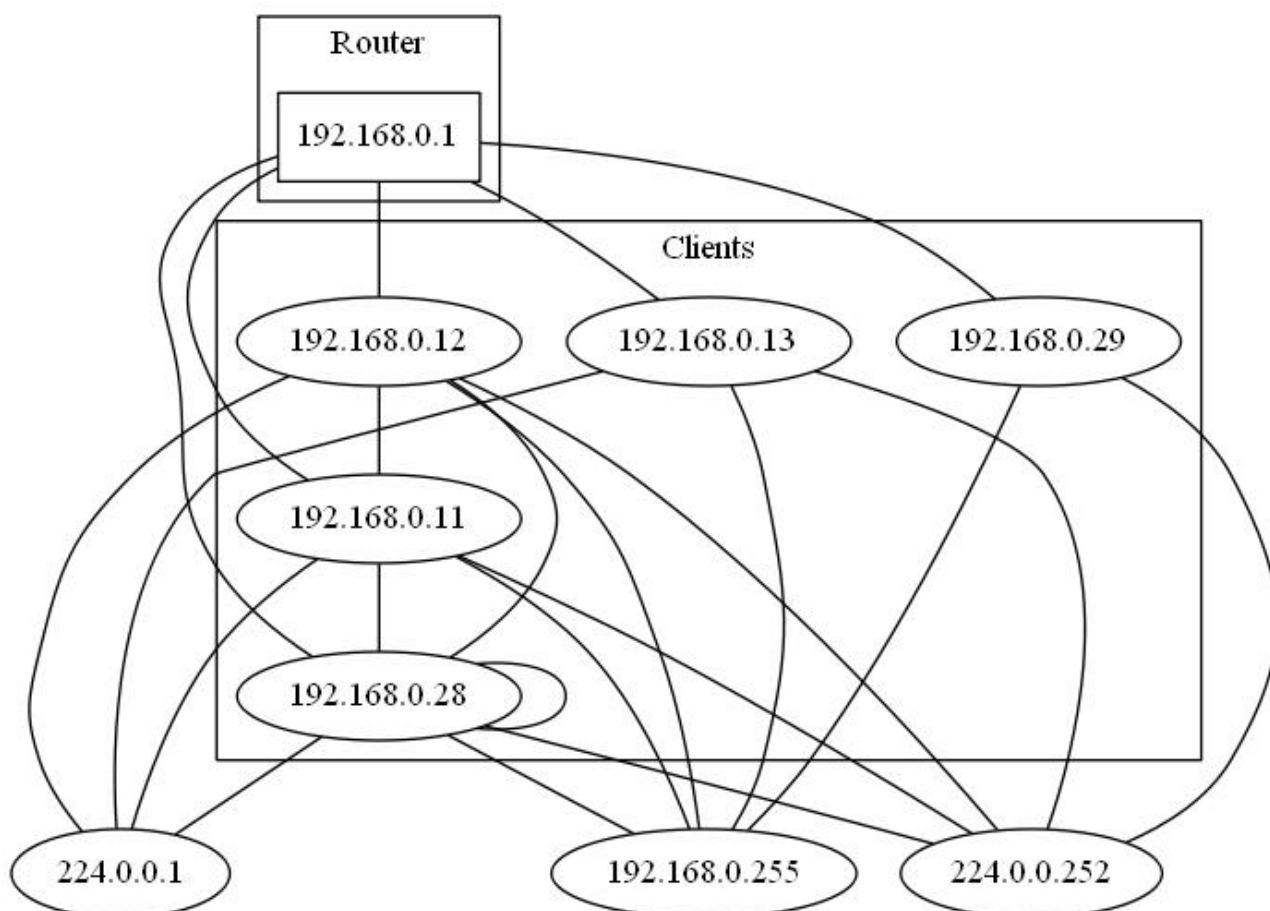
```

D:\GIT\Byczkowski_Dominiak_MonitorRuchuSieci\Serwer\bin\Debug\Serwer.exe
269 TCP 192.168.0.12 192.168.0.11 0x8222b 14141
270 TCP 192.168.0.12 192.168.0.11 0x8222a 14142
271 TCP 192.168.0.12 192.168.0.11 0x82229 14143
272 TCP 192.168.0.12 192.168.0.11 0x82228 14144
273 TCP 192.168.0.12 192.168.0.11 0x82227 14145
274 TCP 192.168.0.12 192.168.0.11 0x82226 14146
275 TCP 192.168.0.12 192.168.0.11 0x82225 14147
276 Unknown 192.168.0.12 192.168.0.29 0x82276 14033
277 Unknown 192.168.0.29 192.168.0.12 0xb4aa 1181
278 Unknown 192.168.0.12 192.168.0.13 0x5c02 23893
279 UDP 192.168.0.12 62.21.99.94 0xc96b 28243
280 UDP 62.21.99.94 192.168.0.12 0x6cea 28455
281 TCP 192.168.0.12 40.77.226.250 0x3945 13699
282 TCP 192.168.0.12 40.77.226.250 0x3950 13700
283 TCP 192.168.0.12 40.77.226.250 0x38aa 13701
284 TCP 192.168.0.12 40.77.226.250 0x394e 13702
285 TCP 192.168.0.12 192.168.0.11 0x820e 14158
286 TCP 192.168.0.12 40.77.226.250 0x38c7 13703
287 TCP 192.168.0.12 40.77.226.250 0x3867 13704
288 TCP 192.168.0.12 40.77.226.250 0x33ab 13705
289 TCP 192.168.0.12 40.77.226.250 0x38a5 13706
290 TCP 192.168.0.12 40.77.226.250 0x3949 13707
291 TCP 192.168.0.12 192.168.0.13 0x5c03 23895
292 TCP 192.168.0.12 96.45.33.100 0xdc4e 7220

```

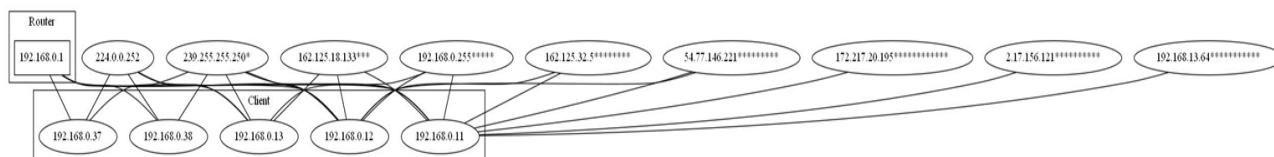
Rys. 33. Wyświetlenie pakietów po usunięciu duplikatów komendą filtr

Graf numer 1, wygenerowany dla pięciu klientów działających na pięciu różnych maszynach. Jak widać graf jest mało czytelny, dlatego podjęte zostały działania w celu jego optymalizacji.

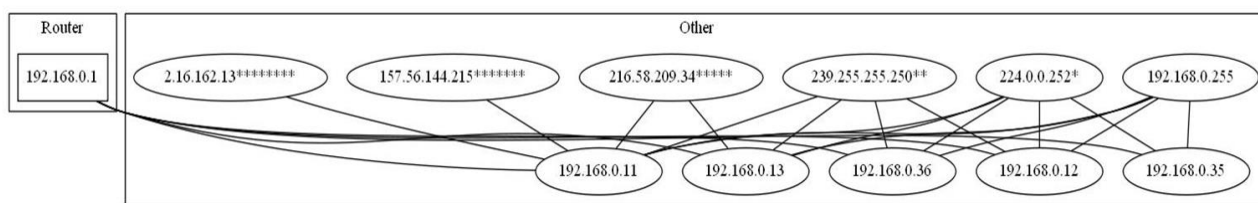


Rys. 34. Pierwszy wygenerowany graf dla pięciu klientów

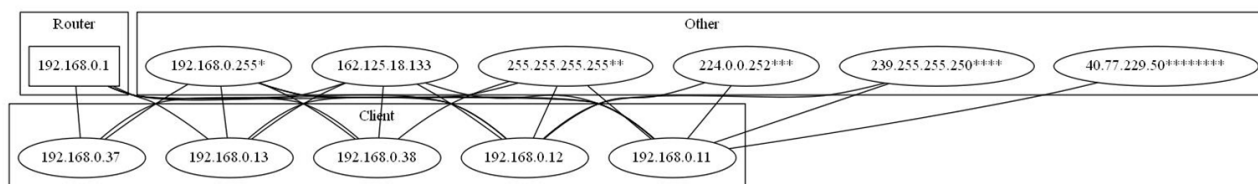
Kolejne próby optymalizacji grafu wynikowego przedstawiono poniżej.



Rys. 35. Próba dodania znaków \* do adresów spoza systemu aby ograniczyć plątaninę połączeń



Rys. 36. Próba dodania trzeciego subgraphu dla adresów spoza sieci spowodowała przyłączenie także subgraphu dla klientów



Rys. 37. Po opanowaniu tego problemu graf dalej był nieczytelny

Finalna wersja grafu w postaci stosunkowo łatwej do zrozumienia dla potencjalnego użytkownika (administratora sieci). Router oznaczono kolorem czerwony, klientów kolorami niebieskimi, reszta adresów z których korzystali klienci wyświetlają się na czarno.

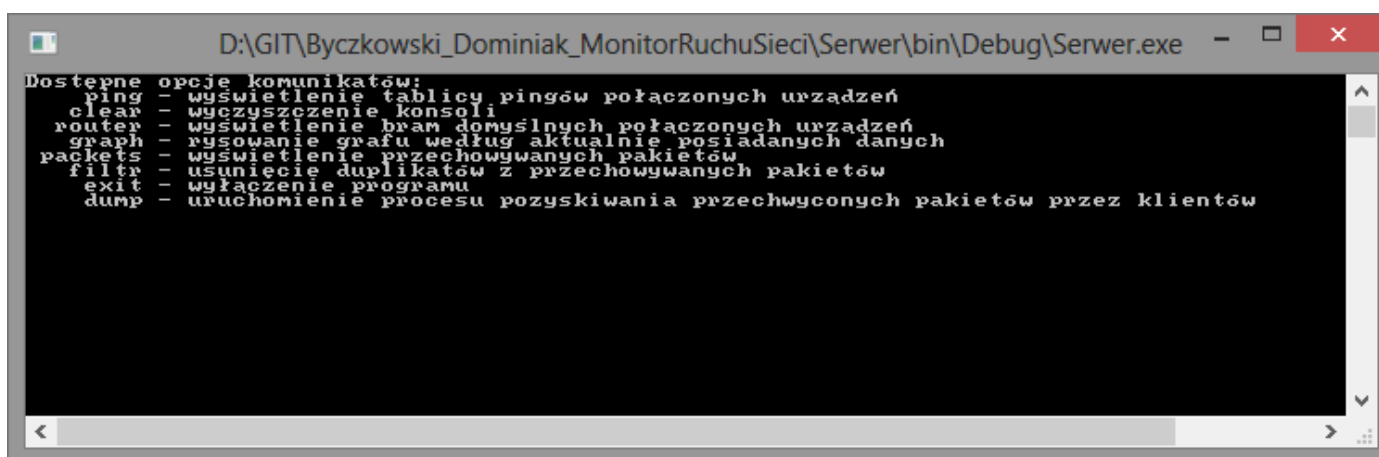


Rys. 38. Finalna wersja grafu – adres routera oznaczono przez czerwony ośmiokąt, a adresy klientów przez niebieski prostokąt.

## 5. Instrukcja użytkownika aplikacji

W tej sekcji zawarto instrukcję dla użytkownika aplikacji, krok po kroku przeprowadzono użytkownika przez proces monitorowania sieci.

1. Pobierz oba projekty z repozytorium GitHub.
2. Przed uruchomieniem aplikacji klienckiej należy uruchomić aplikację serwera jako administrator i pozostawić ją uruchomioną.
3. W aplikacji klienta wpisz komendę `help` i zapoznaj się z możliwościami programu. Po wpisaniu polecenia wyświetlana zostaje lista komend aplikacji, dostęp do pomocy możliwy jest z każdego miejsca w aplikacji.



Rys. 39. wyświetlone informacje po wywołaniu komendy `help`.

4. Uruchom co najmniej dwie aplikacje klienckie na różnych maszynach jako administrator.
5. Aby zwiększyć ilość przechwyconych pakietów można uruchomić kilka stron w przeglądarkach internetowych na maszynach z aplikacjami klienckimi.
6. Następnie w aplikacji serwera wpisz komendę `dump`.
7. Oczekaj chwilę, aż pojawią się komunikaty o przesłanych pakietach od klientów.
8. Wyświetl odebrane pakiety wpisując komendę `packets`.
9. Wpisz komendę `filtr` aby usunąć z przechowywanych pakietów duplikaty, następnie ponownie je wyświetl.
10. Wpisz komendę `graph` aby wygenerować graf topologii sieci i adresów z którymi łączyli się użytkownicy.

## 6. Plany możliwego dalszego rozwoju

W tej sekcji znajdują się plany i kierunki możliwego rozwoju aplikacji oraz elementy, które zostaną zaimplementowane w przyszłości.

- Zliczanie ilości pakietów przesyłanych między poszczególnymi węzłami i wizualizacja tego w grafie np. przez stopniowe pogrubianie linii łączących adresy wraz ze wzrostem ilości przesyłanych informacji.
- Dodanie innego narzędzia do wizualizacji grafu
- Utworzenie lepszego buforu między aplikacją kliencką a snifferem, który przechowywałby dane. W tej chwili służy do tego plik tekstowy i czasami pojawia się błąd, gdy w tym samym momencie sniffer i klient próbują otrzymać do niego dostęp.
- Rozbudowana analiza zawartości pakietów- w tej chwili mamy możliwość ich wyświetlenia oraz na ich podstawie obrazowany jest transfer danych
- Obsługa przypadku gdy klient rozłączyłby się w trakcie trwania transmisji danych do serwera- aktualnie serwer nie otrzymałby komunikatu o przerwaniu połączenia
- Wersja okienkowa aplikacji serwera
- Wersja mobilna aplikacji klienckiej



## 7.Podsumowanie

Projekt został zrealizowany zgodnie z założeniami ustalonymi na początku projektu. W trakcie trwania prac nad aplikacją grupa zdobyła nowe doświadczenia w temacie gniazd sieciowych, komunikacji między urządzeniami w sieci oraz nabyto umiejętności w posługiwaniu się narzędziem GraphViz. Aplikacje mogą posłużyć do celów edukacyjnych, adminstartorom oraz osobom interesującym się sieciami komputerowymi.