POLITECHNIKA POZNAŃSKA

WYDZIAŁ ELEKTRYCZNY

Instytut Automatyki, Robotyki i Inżynierii Informatycznej

Marcin Jasiński

Sprawozdanie z zajęć laboratoryjnych Technologii Sieciowych

Protokoły binarne

7 grudnia 2018

1. Temat: Komunikacja pomiędzy klientem a serwerem (1:1), w oparciu o autorski protokół binarny.

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/widsock_klient/widsock_proba/widsock_proba.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/widsock_serwer/widsock_proba/widsock_proba.cpp

Protokół:

- połączeniowy,
- wszystkie dane przesyłane w postaci binarnej,
- pole operacji o długości 3 bitów,
- pola liczb o długości 32 bitów,
- pole statusu o długości 2 bitów,
- pole identyfikatora o długości 8 bitów,
- dodatkowe pola zdefiniowane przez programistę.

Funkcje oprogramowania:

- nawiązanie połączenia,
- uzgodnienie identyfikatora sesji,
- wykonywanie operacji matematycznych na dwóch argumentach
- wykonywanie operacji matematycznych na wielu argumentach:
- zakończenie połączenia.

Inne:

- gdy wartość wyniku wykracza poza zakres zmiennej, powinien zostać zwrócony kod błędu,
- identyfikator sesji powinien być przesyłany w trakcie komunikacji.

2. Opis protokołu (format komunikatu, zbiór komend i odpowiedzi).

Protokół binarny jest przeznaczony do bycia czytanym przez maszynę, raczej niż ludzkiego użytkownika, w przeciwieństwie do protokołów tekstowych. Protokoły binarne są zwięzłe, dzięki czemu zyskują na szybkości transmisji I interpretacji.

Przykłady protokołów binarnych: RTP, TCP, IP. W zadanym ćwiczeniu używany jest TCP.

Charakterystyka protokołu TCP: Jest on niezawodnym protokołem warstwy transportowej, działającym w trybie klient-serwer. Serwer po uruchomieniu oczekuje na nawiązanie połączenia na określonym porcie przez klienta. Ten w końcu inicjuje połączenie, po czym komunikacja odbywa się w dwu kierunkach, czyli zarówno od nadawcy do odbiorcy jak i od odbiorcy do nadawcy. Podstawowe funkcje protokołu: realizacja dwukierunkowej transmisji połączeniowej, buforowanie danych, zarządzanie przepływem, zapewnienie niezawodności.

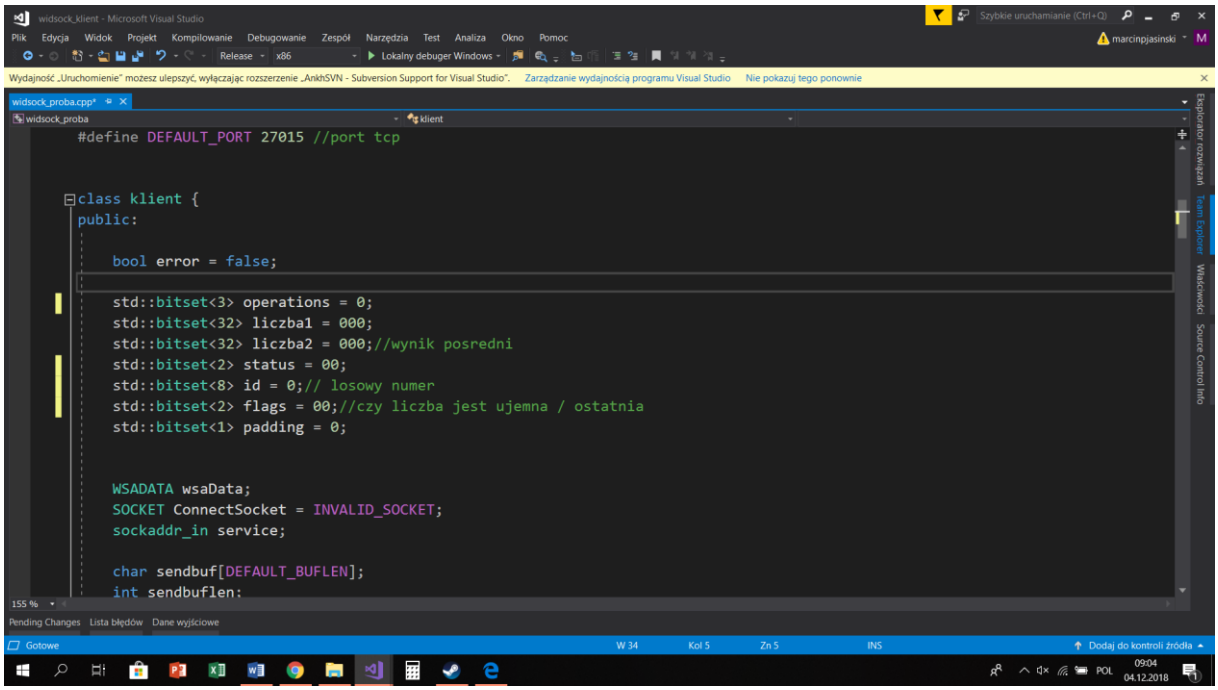
Moje programy są stworzone w języku c++, z użyciem biblioteki windsock2.

Faktyczna budowa komunikatu w programie:

- pole operacji o długości 3 bitów,
  - 000 – dodawanie,
  - 001 – odejmowanie,
  - 010 – mnożenie,
  - 011 – dzielenie,
  - 100 - przyrównanie do podanej liczby
  - 101 - potęgowanie
  - 110 - zaokrąglanie do najbardziej znaczącego miejsca
  - 111 - przyrównanie do losowej liczby z podanego zakresu
- pole liczby o długości 32 bitów:
  - liczba1 to dane od klienta,
  - liczba2 to wynik od serwera
- pole statusu o długości 2 bitów, informacje o błędach
  - 1 bit informujący o przekroczeniu zakresu,
  - 1 bit informujący o niepoprawnym wystąpieniu 0 (np. dzielenie przez 0, 0 do potęgi 0)
- pole identyfikatora o długości 8 bitów,
- flagi wyniku o długości 2 bitów.
  - 1 bit znaku (równa się 1, gdy liczba jest mniejsza od zera, 0 w przeciwnym wypadku)
  - 1 bit informujący czy dany komunikat jest ostatni. (równa się 1, gdy jest ostatni, 0 w przeciwnym wypadku)
- Padding o długości 1 bit. Nie pełni żadnej funkcji, oprócz byciem dopełnieniem protokołu, by ten mógł być podzielny przez 8.

Dane mają stałą długość. Łączna ilość bitów: 48

Ilość przesyłanych skompresowanych bajtów: 6

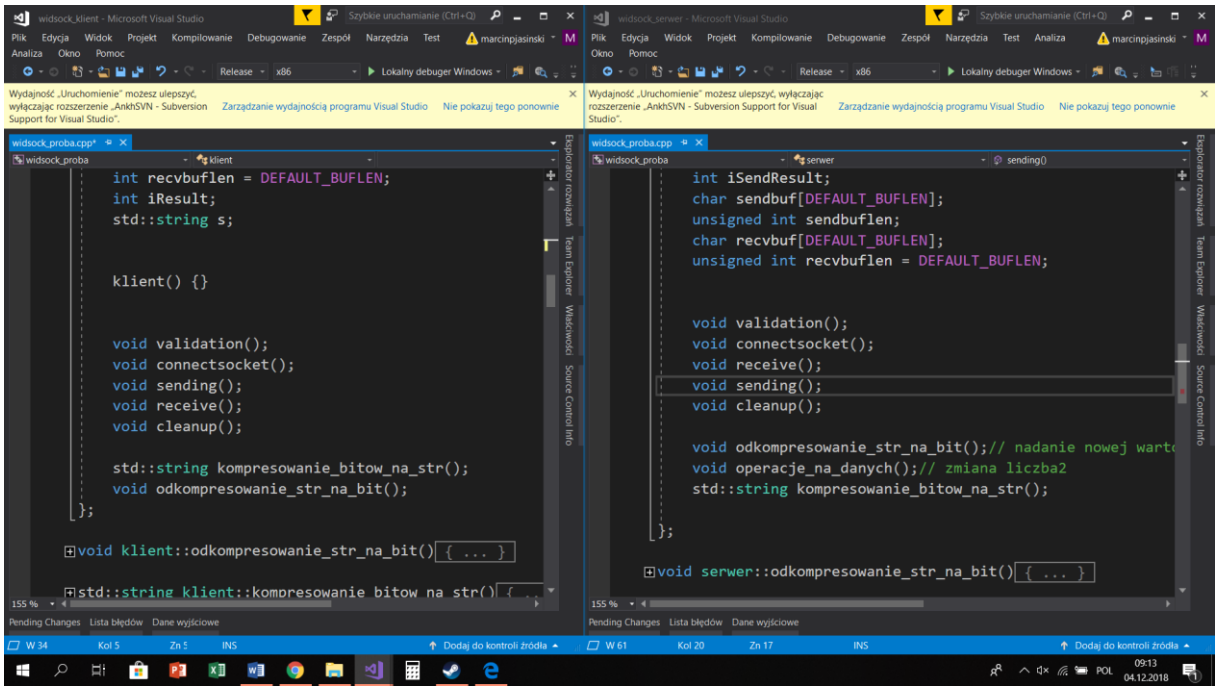


Zrzut ekranu nr 1 – pola komunikatu w programie

3. Aplikacja użytkownika oraz aplikacja serwera (zależnie od wariantu) w wersji źródłowej z komentarzami (w szczególności dotyczącymi fragmentów programów związanych z transmisją).

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/widsock_klient/widsock_proba/widsock_proba.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/widsock_serwer/widsock_proba/widsock_proba.cpp



Zrzut ekranu nr 2 – nazwy utworzonych funkcji klienta (po lewej) i serwera (po prawej)

Opis działania aplikacji klienta:

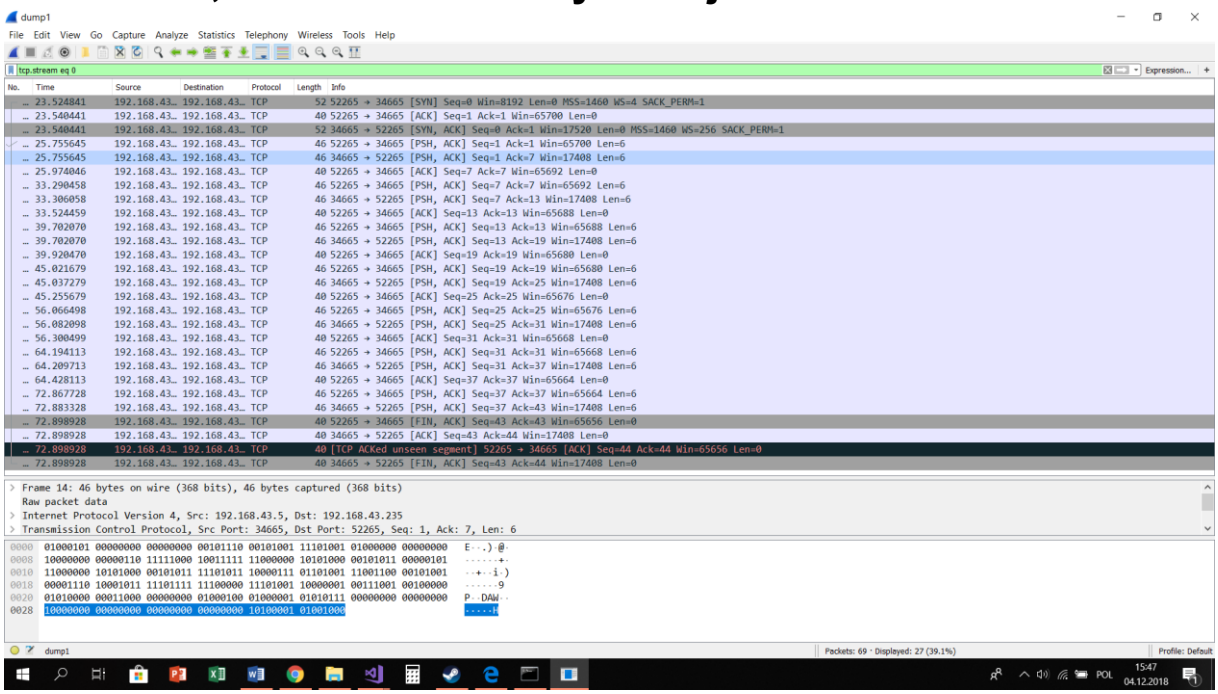
- Klient po zainicjowaniu próbuje się połączyć z serwerem o danym adresie IP. (`funkcje klient::validation() i klient::connectsocket()`)
- Po udanym połączeniu, serwer ustala identyfikator połączenia, a klient ustala początkową wartość serwera. (funkcja `int main()`)
- Klient podaje dane komunikatu: operacje, liczba1, flaga informująca czy dany komunikat będzie ostatni.(funkcja `int main()`)
- Klient wysyła serwerowi komunikat (funkcja `klient::sending()`)
- Klient odbiera odpowiedź serwera, wyświetla wyniki częściowe i ewentualnie kody statusu, informacje o błędach.(funkcje `klient::receive()` i `main()`)
- Jeśli nie ma więcej komunikatów, połączenie jest zamykane (klient::cleanup())

Opis działania aplikacji serwera:

- Po udanych inicjalizacji i połączeniu, serwer ustala identyfikator połączenia (funkcje `serwer::validation(),serwer::connectsocket()` i `main()`)
- serwer odbiera komunikat od klienta (`serwer::receive()`)
- serwer operuje na danych. Zmienia wartość liczba2, jeśli nie pojawia się błąd. W przeciwnym wypadku, Ustawia status na odpowiednią wartość, a liczba2 się nie zmienia. (`serwer::operacje na danych()`)
- serwer wysyła odpowiedź (`serwer::sending()`)
- Jeśli nie ma więcej komunikatów, połączenie jest zamykane. (`serwer::cleanup()`)

Kompresowanie danych: W moich programach pakuję 48 bitów w 6 bajtów (każdy zawierający po 8 bitów). W przełożeniu na c++, konwertuję 6 danych typu `std::bitset` o długości 8 na tablicę znaków typu char, o długości 6. Ta tablica jest przesyłana pomiędzy nadawcą i odbiorcą. Po odbiorze jednak trzeba z powrotem odkompresować znaki na bity. Realizujące to funkcje `odkompresowanie_str_na_bit()` i `kompresowanie_bitow_na_str()` wywoływane są odpowiednio po otrzymaniu danych i przed wysłaniem danych.

4. Przebieg przykładowej sesji komunikacyjnej – opis słowny oraz obraz sesji zarejestrowany przez program Wireshark, wraz ze stosownymi objaśnieniami.

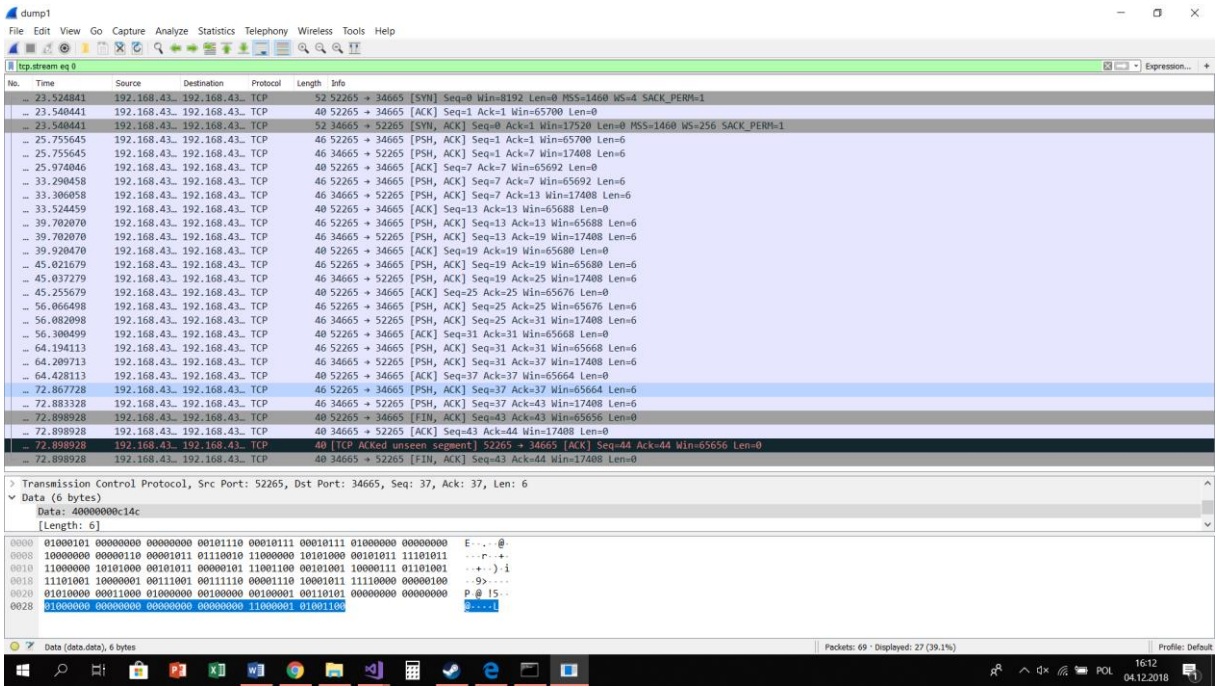


Zrzut ekranu nr 3 - odpowiedź serwera na przypisanie pod niego 5 na początku transmisji

Tabela 1: Podział przesyłanego komunikatu na zrzucie ekranu nr 3:

Część komunikatu	zawartość	interpretacja
pole operacji (3 bit),	100	przyrównanie
pole liczby (32 bit)	00000000 00000000 00000000 00000101	Wynik pośredni wynosi 5
pole statusu (2 bit)	00	Operacja przebiegła bez błędów
pole identyfikatora (8 bit)	00101001	Identyfikator sesji wynosi 41
flagi wyniku (2 bit)	00	Liczba jest nieostatnia i nieujemna

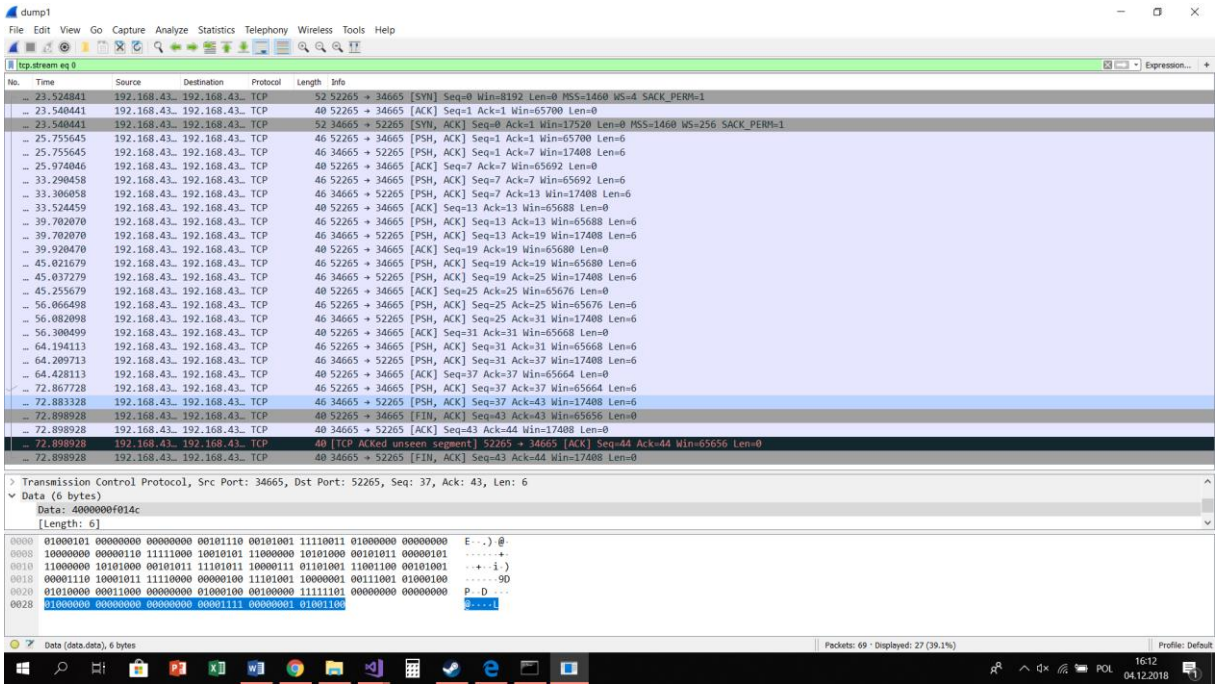
Padding (1 bit)	0	Dopełnienie do 8



Zrzut ekranu nr 4 - prośba klienta na wymnożenie obecnej wartości serwera (20) przez 6 pod koniec transmisji

Tabela 2: Podział przesyłanego komunikatu na zrzucie ekranu nr 4:

Część komunikatu	zawartość	interpretacja
pole operacji (3 bit),	010	mnożenie
pole liczby (32 bit)	00000000 00000000 00000000 00000110	Liczba wynosi 6
pole statusu (2 bit)	00	Nie wystąpił błąd operacji
pole identyfikatora (8 bit)	00101001	Identyfikator sesji wynosi 41
flagi wyniku (2 bit)	10	Liczba jest ostatnia i nieujemna
Padding (1 bit)	0	Dopełnienie do 8



Zrzut ekranu nr 5 - odpowiedź serwera. Wynik mnożenia 20*6 pod koniec transmisji

Tabela 3: Podział przesyłanego komunikatu na zrzucie ekranu nr 5:

Część komunikatu	zawartość	interpretacja
pole operacji (3 bit),	010	mnożenie
pole liczby (32 bit)	00000000 00000000 00000000 01111000	Wynik pośredni wynosi 120
pole statusu (2 bit)	00	Operacja przebiegła bez błędów
pole identyfikatora (8 bit)	00101001	Identyfikator sesji wynosi 41
flagi wyniku (2 bit)	10	Liczba jest ostatnia i nieujemna
Padding (1 bit)	0	Dopełnienie do 8

5. Odpowiedzi na pytania postawione w sekcji „Zadania szczegółowe”

1. **Przygotuj implementacje protokołu komunikacyjnego, aplikacji klienckiej oraz aplikacji serwerowej w dowolnym języku wysokiego poziomu.**

   Obie aplikacje zostały wykonane w języku c++. Kody źródłowe znajdują się na repozytorium o podanych linkach:

Kod aplikacji klienta:
https://github.com/mjaelo/repos1/blob/master/widsock_klient/widsock_proba/widsock_proba.cpp

Kod aplikacji serwera:
https://github.com/mjaelo/repos1/blob/master/widsock_serwer/widsock_proba/widsock_proba.cpp

2. **Przetestuj połączenie pomiędzy programami, rejestrując całość transmisji. Przeanalizuj przechwycone dane. Czy przesłane dane są w pełni binarne?**

   Aby transfer był możliwy, trzeba dane dopełnić do bajtów, co oznacza, że nie są w pełni binarne.

3. **Określ teoretyczną oraz rzeczywistą wielkość komunikatów. Czy rozmiar jest zależny od przesyłanych danych? Czy istnieje możliwość łatwej rozbudowy protokołu?**

   W stworzonym programie rozmiar komunikatu nie zależy od wartości przesyłanych danych, bo na dane dane jest poświęcona dana liczba bitów. TCP przesyła dane w formie bajtów, co oznacza, że trzeba dopełnić brakujące bity zerami. Jednak rozmiar się zmieni gdy rozbuduje się protokół do dając nowe dane(np. znacznik czasu). W programie można łatwo rozbudować protokół. Starczy zadeklarować nowy bitset, Zapełnić go danymi, dodać go do przesyłanych danych. Potem dodać trzeba jeszcze na nowo stworzonym paddingiem, bo liczba bajtów się zmieniła.