

AKADEMIA GÓRNICZO - HUTNICZA

IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI AUTOMATYKI
INFORMATYKI I ELEKTRONIKI**

KATEDRA ELEKTRONIKI



Praca dyplomowa

Temat pracy:

**Przetwarzanie sygnałów akustycznych w
rozproszonych sieciach TCP/IP.**

Autor: **Daniel Mroczka**

Kierunek: **Elektronika**

Ocena:

Promotor:

dr inż. Roman Rumian

Recenzent:

Kraków 2003

SPIS TREŚCI

1. CEL	8
2. ZAGADNIENIA CYFROWEGO PRZETWARZANIA SYGNAŁÓW	9
2.1 HISTORIA I WSTĘP DO CYFROWEGO PRZETWARZANIA SYGNAŁÓW	9
2.2. ZASTOSOWANIA DSP	10
2.2.1. PODZIAŁ ZASTOSOWAŃ DSP	10
2.2.2. ZASTOSOWANIE W TELEKOMUNIKACJI	11
2.2.3. ZASTOSOWANIE W PRZETWARZANIU SYGNAŁÓW AUDIO	12
2.3. DYSKRETNA TRANSFORMACJA FOURIERA (DFT)	13
2.4. TRANSFORMATA ODWROTNA	15
2.5. WŁASNOŚCI DFT	15
2.5.1. LINIOWOŚĆ	15
2.5.2. PRZESUNIĘCIE CYKLICZNE	16
2.5.3. SYMETRIA	16
2.6. SZYBKI ALGORYTM OBLICZANIA DYSKRETNEJ TRANSFORMATY FOURIERA	17
2.6.1. PODZIAŁ W DZIEDZINIE CZASU – DIT	18
2.6.2. PODZIAŁ W DZIEDZINIE CZĘSTOTLIWOŚCI – DIF	21
2.6.3. KIEDY STOSOWAĆ METODĘ DIT, A KIEDY METODĘ DIF?	24
2.7. SPLOT	24
2.7.1. SPLOT LINIOWY	25
2.7.2. SPLOT KOŁOWY	27
2.8. ALGORYTM SEKCJONOWANEGO SZYBKIEGO SPLOTU SYGNAŁÓW DYSKRETNYCH	28
2.8.1. METODA „OVERLAP-SAVE”	29
2.8.2. METODA „OVERLAP-ADD”	30
2.9. INNE ZASTOSOWANIA DFT	35
3. WYBRANE ZAGADNIENIA PROTOKOŁU SIECIOWEGO TCP/IP	36
3.1. WSTĘP	36
3.2. BUDOWA PAKIETU IP	37
3.3. BUDOWA PAKIETU TCP	38
3.4. NAWIĄZANIE POŁĄCZENIA TCP	39
3.5. ZAKOŃCZENIE POŁĄCZENIA TCP	40
3.6. UTRATA PAKIETU, RETRANSMISJA	41
3.7. UDP	44
3.8. ADRESACJA	46
3.9. TRYBY ROZSYŁANIA DANYCH	47
3.9.1. UNICASTING	47
3.9.2. BROADCASTING – ROZGLĄSZANIE DANYCH	47
3.9.3. MULTICASTING - PRZESYŁANIE GRUPOWE	47

4. WYBRANE ELEMENTY SYSTEMÓW ROZPROSZONYCH	48
4.1. WSTĘP I HISTORIA	48
4.2. ZASTOSOWANIA	48
4.3. PODSTAWOWE WŁASNOŚCI SIECI ROZPROSZONYCH	49
4.3.1. DZIELENIE ZASOBÓW	49
4.3.2. OTWARTOŚĆ	49
4.3.3. WSPÓLBIEŻNOŚĆ	49
4.3.4. SKALOWALNOŚĆ	50
4.3.5. TOLEROWANIE USZKODZEŃ	50
4.3.6. PRZEźRO CZYSTOŚĆ	50
4.4. PRZYKŁADY PROJEKTÓW ZWIĄZANE Z OBLICZENIAMI ROZPROSZONYMI W SIECI.	51
4.5. NARZĘDZIA DO TWORZENIA OBIEKTOWYCH PROGRAMÓW ROZPROSZONYCH W ŚRODOWISKACH SIECIOWYCH.	52
4.6. PODSUMOWANIE	53
5. ARCHITEKTURA SYSTEMU	55
5.1. OGÓLNY SCHEMAT ARCHITEKTURY SYSTEMU	55
5.1.1. WSTĘP	55
5.1.2. SERWER – ZARZĄDCA	56
5.1.3. KLIENT	57
5.1.4. BUDOWA PAKIETU UŻYWANEGO DO KOMUNIKACJI POPRZECZ SIEĆ	58
5.1.5. SCHEMAT NAWIĄZANIA POŁĄCZENIA	61
5.1.6. SCHEMAT PRZYGOTOWANIA DO PRACY	62
5.1.7. SCHEMAT WYMIANY DANYCH	63
5.2. OPIS PAKIETÓW STOSOWANYCH W TRANSMISJI	64
5.2.1. ROZSYŁANIE INFORMACJI O SERWERZE W TRYBIE BROADCAST	64
5.2.2. PROCES NAWIĄZANIA POŁĄCZENIA	64
5.2.3. WYSŁANIE PARAMETRÓW TRANSMISJI	65
5.2.4. WYSŁANIE INFORMACJI OD KLIENTA	65
5.2.5. ŻĄDANIE WSPÓŁCZYNNIKÓW	66
5.2.6. WYSŁANIE WSPÓŁCZYNNIKÓW	67
5.2.7. POTWIERDZENIE PRZYJĘCIA WSPÓŁCZYNNIKÓW	69
5.2.8. ŻĄDANIE PODZIAŁU ZADAŃ	70
5.2.9. PODZIAŁ ZADAŃ	70
5.2.10. POTWIERDZENIE PRZYJĘCIA PODZIAŁU ZADAŃ	71
5.2.11. ROZSYŁANIE PRÓBEK	71
5.2.12. WYSŁANIE WYNIKÓW OBLICZEŃ	72
5.2.13. KONTROLA AKTYWNOŚCI KLIENTÓW	72
5.3. ZASADA DZIAŁANIA	73
5.3.1. SCHEMAT BLOKOWY	73
5.3.1. MECHANIZM WIELOWĄTKOWY	74
5.3.2. BUFORY	74
5.3.3. POBIERANIE PRÓBEK DŹWIĘKOWYCH	75
5.3.4. GNIAZDA SIECIOWE TCP I UDP	76
5.3.5. MECHANIZM KOLEJKOWANIA PAKIETÓW	77

5.3.6. MECHANIZM PODZIAŁU PRACY DLA KLIENTÓW	79
5.3.7. POZOSTAŁE ELEMENTY SYSTEMU	81
6. ŚRODOWISKO PROGRAMISTYCZNE DELPHI	82
6.1. DELPHI JAKO ŚRODOWISKO PROGRAMISTYCZNE TYPU RAD	82
6.2. CHARAKTERYSTYKA ŚRODOWISKA	82
6.3. JĘZYK PROGRAMOWANIA PASCAL	83
6.4. ŚRODOWISKO PROGRAMISTYCZNE: DELPHI	83
6.5. ZALETY JĘZYKA PASCAL	84
6.6. WADY JĘZYKA PASCAL	84
6.7. PODSUMOWANIE	84
7. REALIZACJA PROGRAMOWA WYBRANYCH ZAGADNIENÍ	85
7.1. TYPY DANYCH	85
7.2. PRZEBIEG DANYCH	88
7.3. ALGORYTM RADIX-2 Z PODZIAŁEM W CZASIE DIT	89
7.4. ALGORYTM RADIX-2 Z PODZIAŁEM W CZĘSTOTLIWOŚCI DIF	90
7.5. ALGORYTM PRZESTAWIANIA KOLEJNOŚCI PRÓBEK	91
7.6. ALGORYTM OVERLAP-SAVE	94
7.7. ALGORYTM PRZEPAKOWANIA W BUFORZE WEJŚCIOWYM KLIENTA	97
7.8. ALGORYTM PRZEPAKOWANIA W BUFORZE WYJŚCIOWYM KLIENTA	99
7.9. ALGORYTM KOLEJKOWANIA W BUFORZE WEJŚCIOWYM SERWERA	101
8. UŻYTKOWANIE SYSTEMU	104
8.1. OBSŁUGA APLIKACJI ZARZĄDCY	104
8.1.1. GŁÓWNY EKRAN	104
8.1.2. USTAWIENIA PRACY APLIKACJI	105
8.1.3. USTAWIENIA PARAMETRÓW PRÓBKOWANIA	106
8.1.4. USTAWIENIA WSPÓŁCZYNNIKÓW ODPOWIEDZI IMPULSOWEJ	107
8.1.5. ROZPOCZĘCIE TRANSMISJI	108
8.1.6. TRANSMISJA	109
8.1.7. KOŃCZENIE TRANSMISJI	110
8.1.8. URUCHOMIENIE / ZAKOŃCZENIE PRACY SERWERA	110
8.2. OBSŁUGA APLIKACJI KLIENTA	111
8.2.1. DODATKOWE FUNKCJONALNOŚCI	112
9. WNIOSKI I PROPOZYCJE ROZWOJU PROJEKTU	BŁĄD! NIE
<u>ZDEFINIOWANO ZAKŁADKI.</u>	
10. LITERATURA	116

<u>DODATEK A. ELEMENTY TEORII DŹWIĘKU</u>	<u>117</u>
<u>DODATEK B. KARTA MUZYCZNA I PRÓBKOWANIE DŹWIĘKU</u>	<u>118</u>
<u>DODATEK C. DŹWIĘK CYFROWY</u>	<u>120</u>
<u>DODATEK D RFC</u>	<u>125</u>
<u>DODATEK E PRZYKŁAD PAKIETU TCP</u>	<u>127</u>
<u>DODATEK F ANALIZA WYDAJNOŚCI UŻYTYCH ALGORYTMÓW RADIX-2</u>	<u>129</u>

SPIS ILUSTRACJI

RYS. 2.1. ZASTOSOWANIA CYFROWEGO PRZETWARZANIA SYGNAŁÓW	10
RYS. 2.2. PROCES PRZESUNIĘCIA CYKLICZNEGO SYGNAŁU	16
RYS. 2.3. SYMETRIA WIDMA SYGNAŁU	17
RYS. 2.4. STRUKTURA TZW. MOTYLKA W WERSJI PEŁNEJ	19
RYS. 2.5. STRUKTURA TZW. MOTYLKA W WERSJI UPROSZCZONEJ Z ZREDUKOWANĄ O POŁOWĘ ILOŚCIĄ	19
RYS. 2.6. SCHEMAT DIT FFT RADIX-2 DLA $N=8$	20
RYS. 2.7. ZMIANA KOLEJNOŚCI PRÓBEK W SYGNALE WEJŚCIOWYM	21
RYS. 2.8. SCHEMAT DIF FFT RADIX-2 DLA $N=8$	23
RYS. 2.9. PORÓWNANIE CZASU WYKONANIA SPLOTU METODĄ TRADYCYJNĄ I POPRZECZ TRANSFORMACJE FOURIERA.	25
RYS. 2.10. ZASADA DZIAŁANIA SPLOTU LINIOWEGO.	27
RYS. 2.11. ZASADA DZIAŁANIA SPLOTU KOŁOWEGO.	28
RYS. 2.12. OVERLAP – SAVE.	30
RYS. 2.13. OVERLAP – ADD.	32
RYS. 2.14. ZALEŻNOŚĆ WYDAJNOŚCI OBLICZENIOWEJ W ZALEŻNOŚCI OD DŁUGOŚCI TRANSFORMATY	34
RYS. 3.1. PROCES ENKAPSULACJI DANYCH	36
RYS. 3.2. BUDOWA NAGŁÓWKA IP	37
RYS. 3.3. BUDOWA PAKIETU TCP	38
RYS. 3.4. PAKIET ETHERNETOWY	38
RYS. 3.5. NAWIĄZANIE POŁĄCZENIA TCP	40
RYS. 3.6. ZAMYKANIE POŁĄCZENIA TCP	41
RYS. 3.7. SCHEMAT PRZESYŁANIA Z POTWIERDZENIAMI	42
RYS. 3.8. BUDOWA NAGŁÓWKA PAKIETU UDP	45
RYS. 3.9. PSEUDONAGŁÓWEK UDP	45
RYS. 3.10. KLASY ADRESÓW SIECIOWYCH	46
RYS. 3.11. ZAKRESY ADRESÓW SIECIOWYCH	46
RYS. 5.1. PRZYKŁADOWY SCHEMAT ARCHITEKTURY KLIENT-SERWER	55
RYS. 5.2. RODZAJE KOMUNIKACJI POMIĘDZY APLIKACJAMI	56
RYS. 5.3. SCHEMAT BUDOWY WZORCA PAKIETU UŻYWANEGO W SYSTEMIE DO KOMUNIKACJI W SIECI	58
RYS. 5.4. LISTA KOMEND STOSOWANYCH W SYSTEMIE	59
RYS. 5.5. SCHEMAT NAWIĄZANIA POŁĄCZENIA	61
RYS. 5.6. SCHEMAT PRZYGOTOWANIA DO PRACY	62
RYS. 5.7. SCHEMAT WYMIANY DANYCH	63
RYS. 5.8. SCHEMAT PAKIETU DLA KOMENDY NR 5 - ROZSYŁANIE ADRESU IP	64
RYS. 5.9. SCHEMAT PAKIETU DLA KOMENDY NR 27 – TYP TRANSMISJI	65
RYS. 5.10. SCHEMAT PAKIETU DLA KOMENDY NR 28 – TYP KLIENTA	66
RYS. 5.11. SCHEMAT PAKIETU DLA KOMENDY NR 30 – ŻĄDANIE WSPÓŁCZYNNIKÓW	67
RYS. 5.12. TRANSMISJA WSPÓŁCZYNNIKÓW – BUDOWA PIERWSZEGO PAKIETU	68
RYS. 5.13. TRANSMISJA WSPÓŁCZYNNIKÓW – BUDOWA POŚREDNIEGO PAKIETU	68
RYS. 5.14. TRANSMISJA WSPÓŁCZYNNIKÓW – BUDOWA OSTATNIEGO PAKIETU	69
RYS. 5.15. SCHEMAT PAKIETU DLA KOMENDY NR 32 – POTWIERDZENIE PRZYJĘCIA WSPÓŁCZYNNIKÓW	69
RYS. 5.16. SCHEMAT PAKIETU DLA KOMENDY NR 35 – PRYZNANIE PODZIAŁU	70

RYS. 5.17. SCHEMAT PAKIETU DLA KOMENDY NR 36 – POTWIERDZENIE PRYZNANIA PODZIAŁU	71
RYS. 5.18. SCHEMAT PAKIETU DLA KOMENDY NR 11/13 – ROZSYŁANIE PRÓBEK	72
RYS. 5.19. SCHEMAT PAKIETU DLA KOMENDY NR 12/14 – WYNIKI OBLICZEŃ	72
RYS. 5.20. SCHEMAT PAKIETU DLA KOMENDY NR 99/100 – „PING-PONG”	72
RYS. 5.22. SCHEMAT ZAŁADOWANIA DANYCH DO BUFORA WYJŚCIOWEGO	75
RYS. 5.23. PROCES LICZENIA SPLOTU DLA PIERWSZEGO KOMPUTERA	78
RYS. 5.24. PROCES LICZENIA SPLOTU DLA PIERWSZEGO KOMPUTERA	78
RYS. 5.25. PROCES SKŁADANIA WYNIKÓW	79
RYS. 7.1. REPREZENTACJA CIĄGU PRÓBEK W PAKIECIE PRZESYŁANYM PRZESIEĆ	85
RYS. 7.2 REPREZENTACJA CIĄGU PRÓBEK W PAKIECIE, NA KTÓRYM DOKONYWANE SĄ OBLICZENIA O DOWOLNYM N	86
RYS. 7.3. REPREZENTACJA ZBIORU LICZB ZESPOŁONYCH O ILOŚCI ELEMENTÓW N	86
RYS. 7.4. ZMIANA TABLICY 16BITOWEJ W 32BITOWĄ TABLICĘ LICZB ZESPOŁONYCH	87
RYS. 7.5. LISTING FUNKCJI ZMIENROZMWGORE	87
RYS. 7.6. LISTING FUNKCJI ZMIENROZMWDOŁ	87
RYS. 7.7. SCHEMAT „WĘDRÓWKI” DANYCH W PROCEDURACH OBLICZENIOWYCH KLIENTA	88
RYS. 7.8 LISTING FUNKCJI IMPLEMENTUJĄCEJ ALGORYTM FFT DIT	89
RYS. 7.9 LISTING FUNKCJI IMPLEMENTUJĄCEJ ALGORYTM FFT DIF	91
RYS. 7.10. ALGORYTM PRZESTAWIANIA KOLEJNOŚCI PRÓBEK	92
RYS. 7.11. LISTING FUNKCJI IMPLEMENTUJĄCEJ ALGORYTM PRZESTAWIANIA KOLEJNOŚCI PRÓBEK BITREVERSE	93
RYS. 7.12. SCHEMAT PRZEDSTAWIAJĄCY ALGORYTM OVERLAP-SAVE	94
RYS. 7.13 LISTING FUNKCJI OVERLAP-SAVE	96
RYS. 7.14. ZASADA DZIAŁANIA MECHANIZMU PRZEPAKOWUJĄCEGO NA WEJŚCIU	97
RYS. 7.15. LISTING FUNKCJI REALIZUJĄCEJ PRZEPAKOWANIE PAKIETÓW	98
RYS. 7.16. ZASADA DZIAŁANIA MECHANIZMU PRZEPAKOWUJĄCEGO NA WEJŚCIU	99
RYS. 7.17. LISTING FUNKCJI REALIZUJĄCEJ PRZEPAKOWANIE PAKIETÓW NA WEJŚCIU	100
RYS. 7.18. ZASADA DZIAŁANIA MECHANIZMU KOLEJKUJĄCEGO NA WEJŚCIU SERWERA	102
RYS. 7.19. LISTING ALGORYTMU REALIZUJĄCEGO MECHANIZM KOLEJKOWANIA NA WEJŚCIU SERWERA	103
RYS. 8.1. GŁÓWNY EKRAN APLIKACJI ZARZĄDCY	104
RYS. 8.2. USTAWIENIA PRACY APLIKACJI	105
RYS. 8.3. USTAWIENIA PARAMETRÓW PRÓBKOWANIA	106
RYS. 8.4. USTAWIENIA WSPÓŁCZYNNIKÓW	107
RYS. 8.5. MONIT INFORMUJĄCY O ZBYT NISKIEJ MOCY OBLICZENIOWEJ	108
RYS. 8.6. MONIT INFORMUJĄCY O SPEŁNIENIU KONIECZNEGO WARUNKU DO ROZPOCZĘCIA TRANSMISJI	108
RYS. 8.7. LISTA KLIENTÓW I INFORMACJA O ODPOWIEDZI Z ICH STRONY PO ROZESŁANIU PODZIAŁU ZADAŃ	109
RYS. 8.8. INFORMACJA O STANIE BIEŻĄCEJ TRANSMISJI	110
RYS. 8.9. EKRAN APLIKACJI KLIENTA	111
RYS. 8.10 EKRAN ZE WSPÓŁCZYNNIKAMI ODEBRANYMI OD SERWERA, ORAZ OBLICZONE WIDMO CZĘSTOTLIWOŚCI WYZNACZONEJ CZĘŚCI WSPÓŁCZYNNIKÓW	112
RYS. 8.11 PRZYKŁADOWY WYKRES PRZEBIEGU ODPOWIEDZI IMPULSOWEJ ZAŁADOWANEJ PRZESIEĆ KLIENTA	113

WYKAZ SKRÓTÓW

A/C	<i>Analog/Cyfra</i>
C/A	<i>Cyfra/Analog</i>
DFT	<i>Discrete Fourier Transform</i>
DSP	<i>Digital Signal Processing</i>
DIF	<i>Decimation In Frequency</i>
DIT	<i>Decimation in Time</i>
FFT	<i>Fast Fourier Transform</i>
IDFT	<i>Inverse Discrete Fourier Transform</i>
IFFT	<i>Inverse Fast Fourier Transform</i>
MCI	<i>Multimedia Communication Interface</i>
RAD	<i>Rapid Application Development</i>
Radix-N	<i>Podstawa - N</i>
RFC	<i>Request For Comments</i>
RTT	<i>Round-Trip Time</i>
TCP /IP	<i>Transmission Control Protocol/Internet Protocol</i>
UDP	<i>User Datagram Protocol</i>
VCL	<i>Visual Components Library</i>
SNR	<i>Signal to Noise Ratio</i>

1. Cel

Celem pracy jest zbudowanie systemu wykorzystującego moc obliczeniową komputerów znajdujących się w sieci komputerowej do przetwarzania sygnałów akustycznych. Pobierany sygnał akustyczny jest splatany z długą odpowiedzią impulsową filtru. Wykonanie takiej operacji potrzebuje bardzo dużej mocy obliczeniowej którą nie dysponuje pojedynczy komputer osobisty. Dlatego też zadanie jest rozłożone na większą ilość komputerów.

Do podstawowych problemów projektu należy optymalny podział obowiązków pośród komputerów klienckich, odporność na wystąpienie awarii komputera – klienta, a także sieci, możliwość przekonfigurowania parametrów w trakcie pracy systemu oraz zbieranie i interpretacja wyników obliczeń.

Projekt ma na celu zebrać doświadczenia i wnioski związane z problemem dzielenia zadań na poszczególne procesy wykonywane na osobnych komputerach. Ze względu na bardzo obszerny przekrój zagadnień uwzględnionych w projekcie, niniejszą pracę należy postrzegać bardziej jako model doświadczalny służący do późniejszej analizy niż jako gotowy produkt finalny. Wiedząc o tym starano się stworzyć oprogramowanie przeznaczone na rozbudowę i skalowalność aplikacji.

2. Zagadnienia cyfrowego przetwarzania sygnałów

W rozdziale tym zostały zawarte podstawowe informacje o cyfrowym przetwarzaniu sygnałów konieczne do zrozumienia zasady działania projektu.

Omówiony zostanie problem Dyskretnej Transformacji Fouriera, a także jej szybkiej odmiany (FFT), opisane zostały także zagadnienia dotyczące splotu sygnałów cyfrowych.

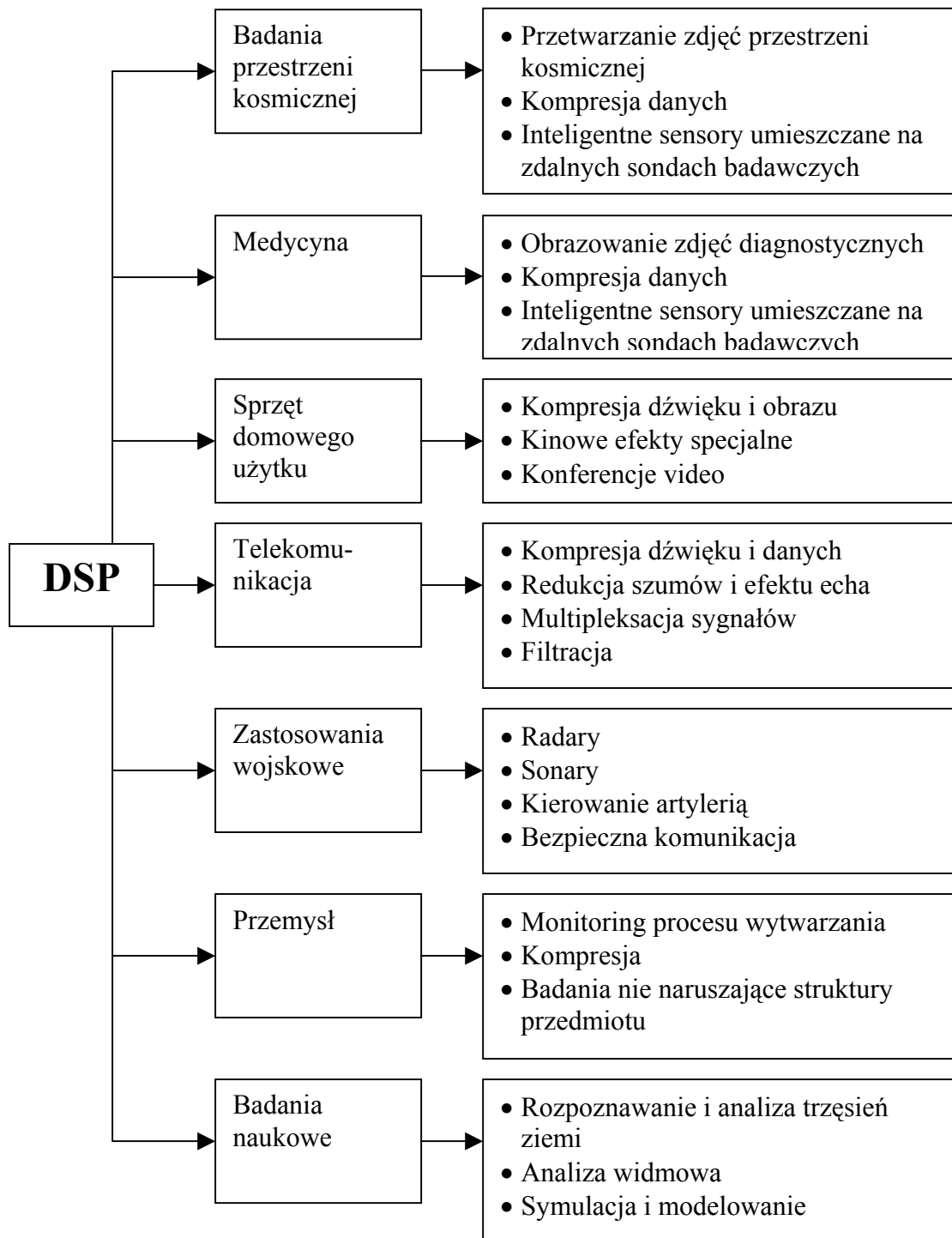
2.1 Historia i wstęp do cyfrowego przetwarzania sygnałów

Przetwarzanie sygnałów cyfrowych zwane potocznie DSP (ang. *Digital Signal Processing*) jest jedną z najbardziej dynamicznie rozwijających się technologii XXI wieku. Procesory sygnałowe analizują sygnał w celu wydobycia z niego informacji i mają swoje zastosowanie np. przy rozpoznawaniu mowy, diagnozy stanu pacjenta na podstawie sygnału elektrokardiogramu lub obrazu ultrasonograficznego, detekcji obiektów latających poprzez analizę sygnałów od nich odbitych, aż po te zastosowania spotykane w codziennym życiu: są wykorzystywane w telefonach komórkowych, sprzęcie audio i AGD. Procesory sygnałowe wykonują analizę i przekształcenia sygnałów wykorzystując specyficzne funkcje matematyczne.

Dziedzina analizy sygnałów ma długą historię, do wybitnych prekursorów należą XVIII i XIX wieczni matematycy tacy jak m.in. Fourier, Euler czy Laplace. Jednak dopiero upowszechnienie się pierwszych komputerów w drugiej połowie XX wieku było przełomem w tej dziedzinie. Kolejnym etapem była rewolucja w komputerach osobistych, kiedy to stały się one bardziej dostępne dla przeciętnego użytkownika, przestając być tylko i wyłącznie narzędziem naukowców i wojska.

2.2. Zastosowania DSP

2.2.1. Podział zastosowań DSP



Rys. 2.1. Zastosowania cyfrowego przetwarzania sygnałów

2.2.2. Zastosowanie w telekomunikacji

Telekomunikacja ma za zadanie przesłać informację z jednej lokalizacji do drugiej. Role informacji może spełniać zarówno rozmowa telefoniczna, sygnał telewizyjny, przesyłane pliki lub innego typu dane. Dla przesyłania danych potrzebny jest kanał transmisyjny. Jego rolę może pełnić para kabli, sygnał radiowy lub światłowód. Przetwarzanie sygnałów cyfrowych zrewolucjonizowało telekomunikację w wielu dziedzinach: w detekcji i sygnalizacji tonu, przesuwaniu pasma częstotliwości, usuwaniu niepożądanych składowych: szumów, zakłóceń (np. pochodzących od źródła zasilania).

Multipleksacja

Dotychczas centrale telefoniczne, aby realizować połączenia pomiędzy abonentami wymagały dla każdego z numerów osobnych kabli i przełączania w sposób mechaniczny. Dzięki zastosowaniu DSP w jednej parze kabli można transmitować wiele kanałów (np. T-carrier może transmitować jednocześnie 24 kanały), przez co prowadzenie kabli połączeniowych do kolejnych stacji stało się wielokrotnie tańsze i szybsze.

Kompresja

Umożliwia zastosowanie wydajnych algorytmów kompresji nieznacznie tylko pogarszających parametry odbieranego sygnału, zajmując znacznie mniejsze pasmo kanału transmisyjnego.

Redukcja efektu echa

Jest to poważny problem przy dalekich połączeniach nasilający się szczególnie przy połączeniach międzykontynentalnych. Rozmowa wówczas staje się wysoce utrudniona.

2.2.3. Zastosowanie w przetwarzaniu sygnałów audio

Dwa główne zmysły człowieka, czyli wzrok i słuch mają swoje odzwierciedlenie w postaci przetwarzania sygnałów akustycznych i wizyjnych.

Muzyka

Wielką rewolucją w dziedzinie audio stały się urządzenia hi-fi wyposażone w cyfrowy procesor dźwięku. Za jego pomocą można odtwarzać wybrane charakterystyki akustyczne pewnych pomieszczeń (np. hal koncertowych) oraz dźwięku w otwartej przestrzeni. Procesor dźwiękowy w tym momencie nie wykonuje nic innego jak splot sygnału ze współczynnikami odpowiedzi impulsowej zebranymi w danym pomieszczeniu. Wystąpi efekt echa tzn. kilkuset milisekundowych opóźnień, dających wrażenie przestrzennego sygnału dźwiękowego. Do wykonania tej operacji potrzebna jest spora moc obliczeniowa i tylko wyspecjalizowane procesory sygnałowe są w stanie zaoferować ich bogaty wachlarz możliwości.

Rozpoznawanie mowy

Jest to doskonały przykład rozbieżności w percepcji umysłu ludzkiego a możliwościami „martwej” maszyny obliczeniowej. Łatwo jest utworzyć program wykonujący niezwykle skomplikowane obliczenia, trudniej natomiast utworzyć algorytm, który doskonale rozpoznawałby mowę ludzką.

DSP radzi sobie z tym problemem następująco: wyodrębnia każde słowo i stara się analizować i porównywać z bankiem dostępnych wzorców słów. Pomimo wielu czynionych prac, w tej dziedzinie jest jeszcze wiele do zrobienia.

2.3. Dyskretna Transformacja Fouriera (DFT)

Dyskretna Transformacja Fouriera jest podstawowym narzędziem analizy częstotliwościowej sygnałów. Możemy uznać, że czas trwania przetwarzanego sygnału jest ograniczony. W przypadku sygnałów cyfrowych ich przetwarzanie ma charakter określonych obliczeń numerycznych. Jedną z najczęściej wykonywanych operacji jest obliczanie widma sygnału. Widmo dyskretnego sygnału $x(n)$ które powstało w wyniku próbkowania sygnału analogowego $x(t)$ z okresem T_0 , jest okresowe i przedstawia się następującą zależnością:

$$X_p(j\omega) = \frac{1}{T_0} \cdot \sum_{k=-\infty}^{\infty} X[j(\omega - k\Omega_0)]$$

gdzie Ω_0 to częstotliwość próbkowania. Zachowując warunki wynikające z twierdzenia o próbkowaniu (nośnik widma sygnału jest ograniczony oraz częstotliwość próbkowania jest większa bądź równa maksymalnej częstotliwości w sygnale próbkowanym), widmo to w ramach jednego okresu jest takie samo jak widmo sygnału analogowego $X(j\omega)$. Innym sposobem wyznaczenia widma sygnału dyskretnego jest:

$$X_p(j\omega) = \sum_{n=-\infty}^{\infty} x(n) \cdot \int_{-\Omega_0/2}^{\Omega_0/2} X_p(j\omega) e^{j\omega n T_0} d\omega$$

Kolejne próbki sygnału dyskretnego, to współczynniki rozwinięcia widma tego sygnału w wykładniczy szereg Fouriera:

$$x(n) = \frac{1}{\Omega_0} \cdot \int_{-\Omega_0/2}^{\Omega_0/2} X_p(j\omega) e^{j\omega n T_0} d\omega$$

Gdy częstotliwość próbkowania jest większa od częstotliwości Nyquista, powyższą zależność można sprowadzić do odwrotnej transformacji Fouriera widma sygnału analogowego:

$$x(n) = \frac{1}{2\pi} \cdot \int_{-\Omega_0/2}^{\Omega_0/2} X_p(j\omega) e^{j\omega n T_0} d\omega = \frac{1}{2\pi} \cdot \int_{-\infty}^{\infty} X(j\omega) e^{j\omega n T_0} d\omega$$

Założmy, że sygnał analogowy $x(t)$ jest sygnałem ograniczonym w czasie (τ) i że jest równocześnie sygnałem przyczynowym:

$$x(t) = 0 \Leftrightarrow t < 0 \text{ lub } t > \tau$$

Liczba niezerowych próbek sygnału $x(n)$ wynosi:

$$N = \left\lceil \frac{\tau}{T_0} \right\rceil$$

Transformacja Fouriera sygnału dyskretnego przyjmuje postać sumy skończonej:

$$X_p(j\omega) = \sum_{n=0}^{N-1} x(n)e^{-jn\omega T_0}$$

Aby odtworzyć sygnał dyskretny wystarczy znajomość wartości widma $X_p(j\omega)$ dla N dowolnie wybranych różnych wartości częstotliwości.

$$\begin{bmatrix} 1 & e^{-j\omega_0 T_0} & e^{-j2\omega_0 T_0} & \dots & e^{-j(N-1)\omega_0 T_0} \\ 1 & e^{-j\omega_1 T_0} & e^{-j2\omega_1 T_0} & \dots & e^{-j(N-1)\omega_1 T_0} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & e^{-j\omega_{N-1} T_0} & e^{-j2\omega_{N-1} T_0} & \dots & e^{-j(N-1)\omega_{N-1} T_0} \end{bmatrix} \cdot \begin{bmatrix} x(0) \\ x(1) \\ \dots \\ x(N-1) \end{bmatrix} = \begin{bmatrix} X(j\omega_0) \\ X(j\omega_1) \\ \dots \\ X(j\omega_{N-1}) \end{bmatrix} \quad (1)$$

Jest to układ N równań liniowych z N niewiadomymi, gdzie niewiadome to próbki sygnału dyskretnego. Wyznacznikiem macierzy głównej jest wyznacznik Vandermonde'a,. Jeśli do obliczeń wybierze się różne częstotliwości, wyznacznik ten będzie różny od zera i powyższy układ będzie posiadał rozwiązanie.

Rozwiązanie powyższego układu równań przedstawia się bardziej obrazowo, gdy próbki są rozmieszczone równomiernie wzdłuż przedziału $[0; \Omega_0)$:

$$\omega_k = \frac{2\pi}{T_0} \cdot \frac{k}{N}, \quad k = 0, 1, 2, \dots, N-1 \quad (2)$$

Widmo sygnału dyskretnego w powyższych punktach oznaczmy przez $X(k)$:

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{kn} \quad (3)$$

gdzie

$$W_N = e^{-j\frac{2\pi}{N}} \quad (4)$$

$$e^{-j\frac{2\pi}{N}} = \cos\left(\frac{2\pi}{N}\right) - j \sin\left(\frac{2\pi}{N}\right)$$

Natomiast samo rozwiązanie układu równań (1) przyjmuje postać:

$$x(n) = \frac{1}{N} \cdot \sum_{k=0}^{N-1} X(k) W_N^{-kn} \quad (5)$$

Wzór (3) określa dyskretną transformację Fouriera sygnału $x(n)$ DFT (ang. *Discrete Fourier Transform*). Natomiast wzór (5) określa odwrotną dyskretną transformatę Fouriera sygnału $X(k)$ – IDFT (ang. *Inverse Discrete Fourier Transform*).

2.4. Transformata odwrotna

$$x(n) = \frac{1}{N} \left[\sum_{k=0}^{N-1} X^*(k) W_N^{kn} \right]^*$$

Aby otrzymać oryginał sygnału na podstawie jego widma, należy dokonać transformaty jednak nie na samym widmie, lecz na jego sprzężonej postaci. Otrzymany wynik należy podzielić przez ilość punktów transformaty oraz ponownie dokonać operacji sprzężenia na otrzymanym wyniku.

2.5. Własności DFT

2.5.1. Liniowość

Niech będą dane dwa sygnały o takiej samej liczbie próbek: $x_1(n)$ oraz $x_2(n)$, to wynikiem ich kombinacji liniowej jest sygnał $x_3(n)$.

$$x_3(n) = \alpha_1 \cdot x_1(n) + \alpha_2 \cdot x_2(n)$$

Obrazem kombinacji widm sygnałów jest natomiast:

$$X_3(k) = \alpha_1 \cdot X_1(k) + \alpha_2 \cdot X_2(k)$$

Gdy sygnały $x_1(n)$ i $x_2(n)$ zawierają różną liczbę próbek, np.: N_1 , N_2 to własność jest dalej prawdziwa, należy wówczas wybrać większą z liczb:

$$N_3 = \max[N_1, N_2]$$

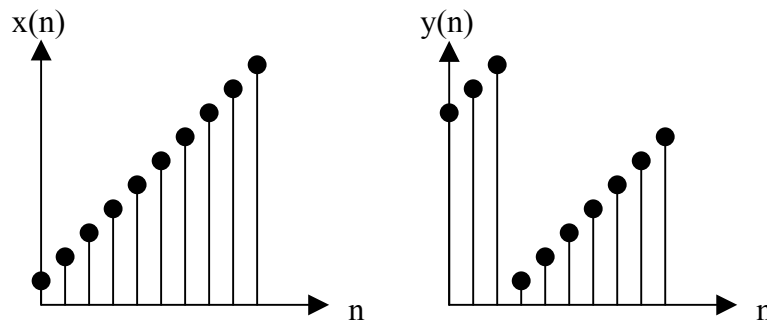
Oraz sygnał o mniejszej ilości próbek uzupełnić zerami do wartości N_3 , wówczas możemy zapisać:

$$X_1(k) = \sum_{n=0}^{N_1-1} x_1(n) \cdot W_{N_3}^{kn}$$
$$X_2(k) = \sum_{n=0}^{N_2-1} x_2(n) \cdot W_{N_3}^{kn}$$

2.5.2. Przesunięcie cykliczne

Jeżeli sygnał $x(n)$ składający się z N próbek zostanie poddany przesunięciu cyklicznemu o jedną próbkę w prawo, wówczas próbka zerowa zostanie przesunięta na miejsce próbki pierwszej, próbka pierwsza zostanie przesunięta na miejsce próbki drugiej... i tak aż do próbki o numerze $N-1$, która zajmie miejsce próbki zerowej. Związek między sygnałem $x(n)$ a sygnałem przesuniętym $y(n)$ ma postać:

$$x(n) = y(\text{mod}(b + m, N))$$

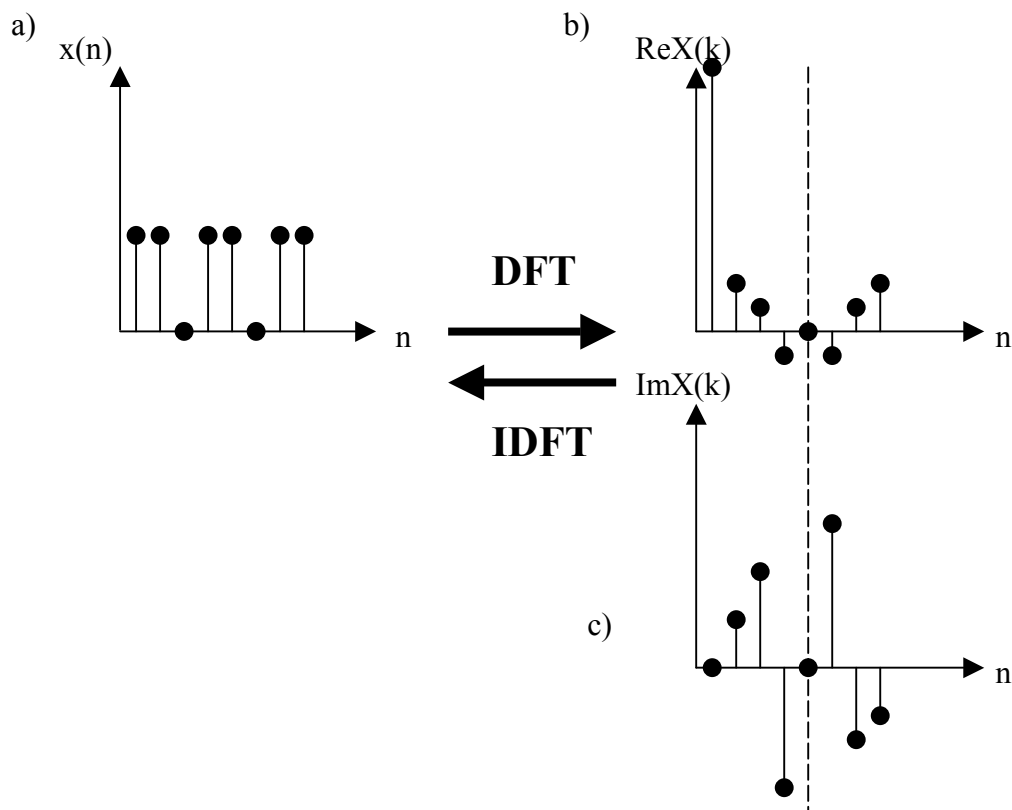


Rys. 2.2. Proces przesunięcia cyklicznego sygnału składającego się z 10 próbek w prawo o trzy próbki. Z lewej sygnał przed przesunięciem, z prawej sygnał po przesunięciu

2.5.3. Symetria

Dla sygnału o wartościach rzeczywistych charakterystyka amplitudowa jest funkcją parzystą, natomiast charakterystyka fazowa może zostać przedstawiona za pomocą funkcji nieparzystej. Ta własność transformaty przenosi się na DFT.

$$X(k) = X(\text{mod}(N - k, N)), \quad k = 0, 1, 2, \dots, N - 1$$



Rys. 2.3. Symetria widma sygnału. a) sygnał w czasie rzeczywistym b) część rzeczywista widma sygnału, sygnały są do siebie symetryczne względem osi przechodzącej przez punkt wyznaczony przez położenie próbki piątej ($k=4$), próbka pierwsza o indeksie $k=0$ nie ma swojego punktu symetrycznego gdyż $X(0)=X(\text{mod}(N-0, N))=X(\text{mod}(N,N))=X(0)$; c) część urojona widma sygnału, sygnały są do siebie nieparzyste względem próbki $k=4$

2.6. Szybki algorytm obliczania dyskretnej transformaty Fouriera

Aby obliczyć DFT sygnału $x(n)$ o długości N , należy wykonać $4 \cdot N^2$ mnożeń liczb rzeczywistych oraz $2N \cdot (2N-1)$ sumowań liczb rzeczywistych. Ilość potrzebnych obliczeń, a zarazem czas ich wykonania jest proporcjonalny do N^2 . W przypadku ciągłego wykonywania obliczeń czas trwania obliczeń staje się zbyt długi, aby umożliwić w czasie rzeczywistym liczenie długich splotów.

Algorytm szybkiej transformaty Fouriera – FFT (ang. *Fast Fourier Transform*) został po raz pierwszy opisany przez amerykańskich matematyków J.W. Cooleya i J.W. Tukeya w 1965 roku. W zależności od sposobu podziału próbek wyróżniamy dwie metody: podziału w dziedzinie czasu - DIT (ang. *Decimation in Time*) oraz metodę podziału w dziedzinie częstotliwości - DIF (ang. *Decimation In Frequency*).

2.6.1. Podział w dziedzinie czasu – DIT

W tej metodzie dokonuje się podziału próbek transformowanego sygnału na parzyste i nieparzyste, licząc próbki rozpoczynając od indeksu zerowego. Na każdym takim zbiorze wykonuje się DFT, a następnie dokonuje się odtworzenia widma całego sygnału z dwóch widm cząstkowych.

Przy każdym takim podziale zyskuje się dwukrotnie mniejszą ilość wymaganych obliczeń, a taki podział można kontynuować dalej aż do otrzymania zbioru dwuelementowego.

Zakładamy, że liczba próbek sygnału $x(n)$ jest całkowitą potęgą liczby dwa.

$$N = 2^M$$

Korzystając ze wzoru na DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

Pogrupujmy składniki na parzyste i nieparzyste:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2kn} + \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{(2n+1)k}$$

Co z kolei można zapisać następująco:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) W_N^{2kn} + W_N^k \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) W_N^{2nk}$$

Z powyższej zależności możemy wyróżnić dwie funkcje:

$$X(k) = Y(k) + W_N^k Z(k)$$

gdzie:

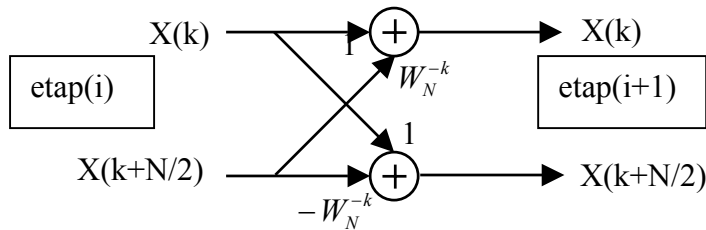
$$Y(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n) * (W_N^2)^{nk}$$

$$Z(k) = \sum_{n=0}^{\frac{N}{2}-1} x(2n+1) * (W_N^2)^{nk}$$

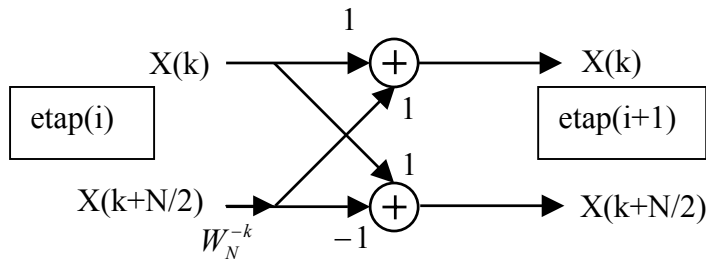
Funkcje $Y(k)$ i $Z(k)$ reprezentują dwie $N/2$ punktowe DFT. Wynika z tego, że N -punktową transformatę możemy wyliczyć jako sumę dwóch $N/2$ punktowych transformat. W ten sposób składa się widma dwuprążkowe w widma czteroprążkowe, czteroprążkowe

w ośmioprzędkowe aż do chwili odtworzenia widma N-przędkowego, czyli widma całego sygnału.

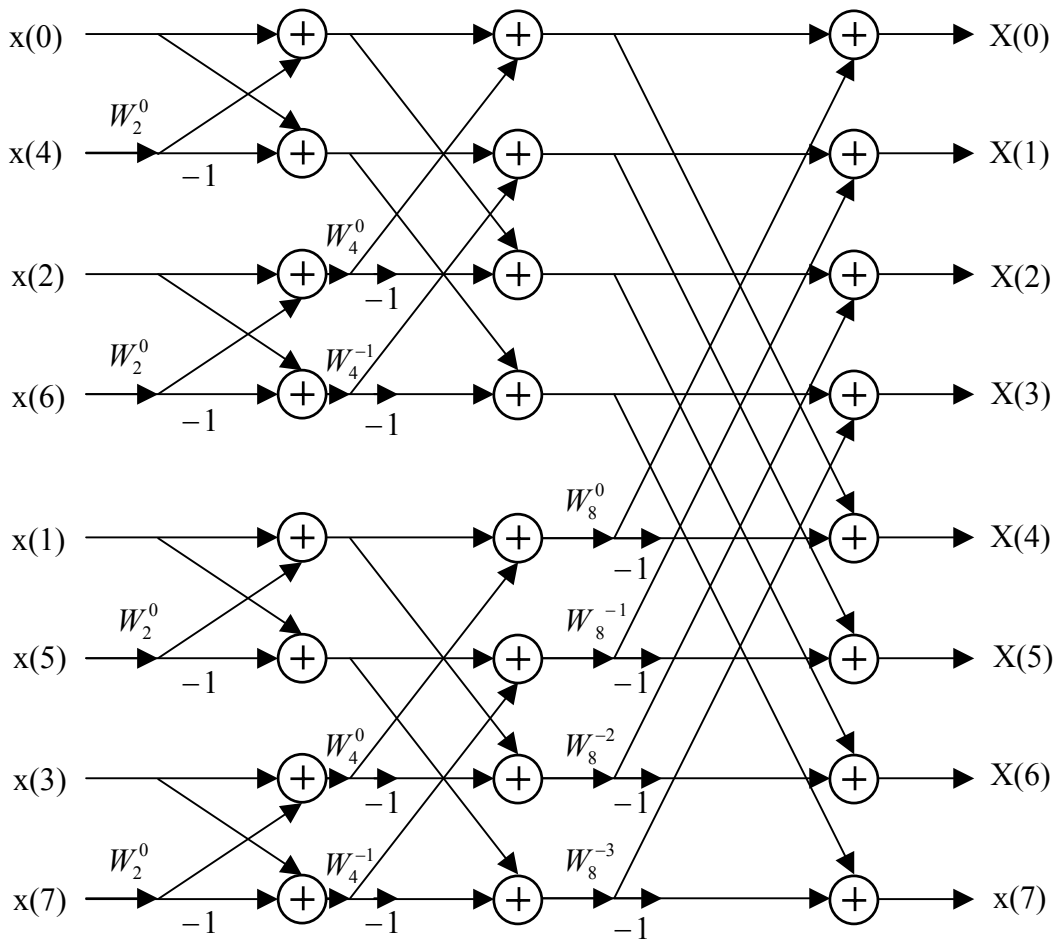
Dla danego N wykonujemy $\log_2 N$ etapów obliczeń. Dla przykładu dla $N=8$ ilość etapów wynosi $\log_2 8=3$, rozpoczynamy od wykonania czterech dwupunktowych DFT, potem ich widma składamy za pomocą DFT w dwa widma czteropunktowe, aby zakończyć na jednym ośmiopunktowym widmie. W każdym etapie wykonuje się N mnożeń i tyle samo dodawań zespolonych. Zatem koszt wykonania N punktowej FFT wynosi $N \log_2 N$ mnożeń i tyle samo dodawań zespolonych. Istnieje możliwość redukcji mnożeń o połowę w wersji uproszczonej (szybkiej).



Rys. 2.4. Struktura tzw. motylka w wersji pełnej



Rys. 2.5. Struktura tzw. motylka w wersji uproszczonej z zredukowaną o połowę ilością mnożeń korzystając w wyłączenia czynnika W_N^{-k} zgodnie z: $-W_N^{-k} = W_N^{-(k+N/2)}$



Rys. 2.6. Schemat DIT FFT radix-2 dla N=8

Bardzo ważną cechą charakterystyczną algorytmu DIT jest konieczność zmiany kolejności próbek w sygnale wejściowym. Zastosowano tu metodę numeracji o odwróconej kolejności bitów. Na pozycji o numerze n znajdujemy próbkę $x(\hat{n})$, gdzie wskaźnik \hat{n} powstaje przez odwrócenie zapisu binarnego liczby n .

N	Zapis binarny n	Odwrócony zapis binarny n	\hat{n}
0	000	000	0
1	001	001	4
2	010	010	2
3	011	011	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Rys. 2.7. Zmiana kolejności próbek w sygnale wejściowym

2.6.2. Podział w dziedzinie częstotliwości – DIF

Opierając się na podobnych założeniach co przy metodzie DIT, a mianowicie że N jest potęgą liczby 2 oraz potrzebą wyznaczenia k-tego prążka $X(k)$ widma Fouriera możemy decymować zamiast próbek sygnału $x(n)$ prążki widma $X(k)$. Innymi słowy wyznaczymy prążki o indeksach parzystych i nieparzystych.

Korzystając ze wzoru na DFT:

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}$$

Zapiszmy ten sam wzór rozdzielając na dwie sumy:

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{kn} = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=\frac{N}{2}}^{N-1} x(n) W_N^{nk} = \\ &= \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n + \frac{N}{2}\right) W_N^{(n+N/2)k} \end{aligned}$$

Ponieważ $W_N^{(n+\frac{N}{2})k} = W_N^{nk} * W_N^{\frac{N}{2}k}$ oraz $W_N^{\frac{N}{2}k} = (-1)^k$, powyższy wzór można przekształcić do postaci:

$$X(k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_N^{nk} + (-1)^k \sum_{n=0}^{\frac{N}{2}-1} x\left(n + \frac{N}{2}\right) W_N^{nk} = \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (-1)^k x\left(n + \frac{N}{2}\right) \right\} W_N^{nk}$$

Teraz możemy rozdzielić próbki DFT na parzyste i nieparzyste:

$$\begin{aligned} X(2r) &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (-1)^{2r} x\left(n + \frac{N}{2}\right) \right\} W_N^{2nr} = \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + x\left(n + \frac{N}{2}\right) \right\} W_{\frac{N}{2}}^{nr}, r = 0, \dots, \frac{N}{2} - 1 \end{aligned}$$

$$\begin{aligned} X(2r+1) &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ x(n) + (-1)^{2r+1} x\left(n + \frac{N}{2}\right) \right\} W_N^{(2r+1)n} = \\ &= \sum_{n=0}^{\frac{N}{2}-1} \left\{ [x(n) - x\left(n + \frac{N}{2}\right)] W_N^n \right\} W_{\frac{N}{2}}^{nr}, r = 0, \dots, \frac{N}{2} - 1 \end{aligned}$$

Podstawiając:

$$X_1(r) = X(2r), r = 0, \dots, \frac{N}{2} - 1$$

$$x_1(n) = x(n) + x\left(n + \frac{N}{2}\right), n = 0, \dots, \frac{N}{2} - 1$$

otrzymujemy:

$$X_1(r) = \sum_{n=0}^{\frac{N}{2}-1} x_1(n) W_{\frac{N}{2}}^{nr}$$

Taką N/2 punktową transformatę możemy podzielić na dwie sumy:

$$X_1(r) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x_1(n) + (-1)^r x_1\left(n + \frac{N}{4}\right) \right\} W_{\frac{N}{2}}^{nr}$$

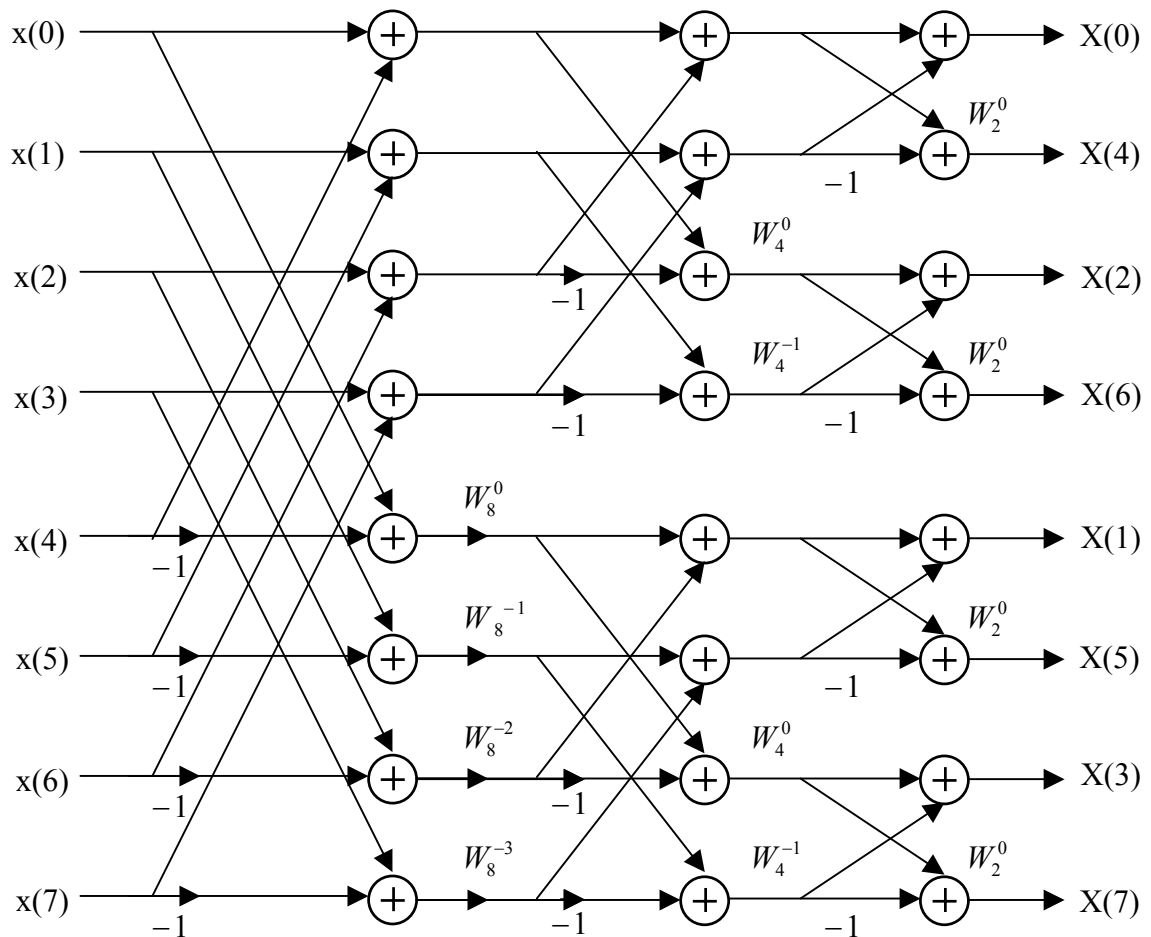
Dla próbek parzystych (niech $r=2s$) otrzymujemy:

$$X_1(2s) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ x_1(n) + x_1\left(n + \frac{N}{4}\right) \right\} W_{\frac{N}{4}}^{sn}$$

a dla próbek nieparzystych ($r=2s+1$):

$$X_1(2s+1) = \sum_{n=0}^{\frac{N}{4}-1} \left\{ [x_1(n) - x_1\left(n + \frac{N}{4}\right)] W_N^{2n} \right\} W_{\frac{N}{4}}^{sn}$$

Analogicznie $X(2r+1)$ możemy podzielić na dwie sumy, tak jak zostało to zrobione z $X(2r)$ dzieląc na $X_1(2s)$ i $X_1(2s+1)$. W rezultacie otrzymamy do policzenia cztery $N/4$ punktowe DFT. Kolejnym krokiem jest podzielenie tych transformat na osiem $N/8$ punktowych transformat. Podziałów dokonuje się tak długo aż otrzymamy dwupunktową DFT.



Rys. 2.8. Schemat DIF FFT radix-2 dla $N=8$

2.6.3. Kiedy stosować metodę DIT, a kiedy metodę DIF?

Z wydajnościowego punktu widzenia obie metody reprezentują takie same rezultaty.

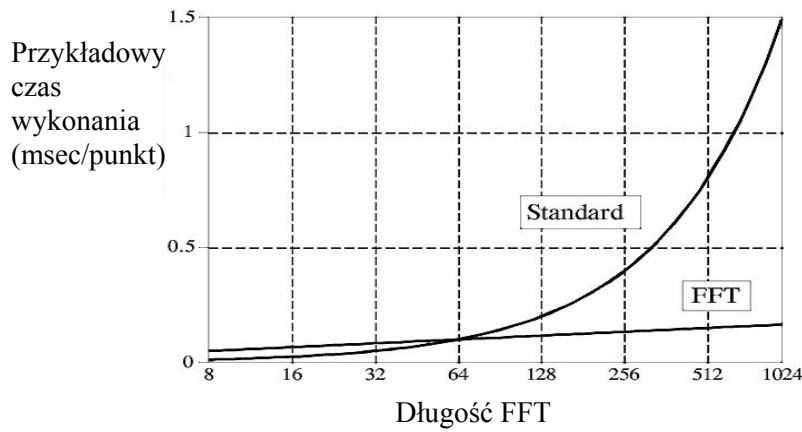
Zatem w przypadku liczenia widma sygnału tylko raz jest obojętne która z metod zostanie użyta, należy tylko pamiętać o operacji odwracania kolejności próbek na początku operacji, bądź po jej zakończeniu.

Jednak w przypadku liczenia splotu poprzez FFT (punkt 2.7) możemy w ogóle zrezygnować z konieczności odwracania kolejności próbek. Wystarczy, aby sygnał rzeczywisty został wpierv poddany transformacji metodą DIF, a wsteczna transformacja została wykonana metodą DIT. W ten sposób zarówno na wejściu, jak i na wyjściu obowiązywać będą poprawne kolejności próbek.

2.7. Splot

Splot jest połączeniem dwóch sygnałów dyskretnych dających w rezultacie trzeci sygnał, będący wynikiem szeregu mnożeń i dodawań sygnałów wejściowych. Splot opisuje operacje filtracji jednego sygnału przez drugi. Sygnały rozróżnia się pod względem spełnianych funkcji na filtrowany i filtrujący.

Splot można liczyć na dwa sposoby. Pierwszym sposobem jest wykonanie szeregu mnożeń i sumowań (metoda tradycyjna) oraz poprzez mnożenie widm sygnałów splatanych, a następnie wykonaniu na nich odwrotnej transformaty. Na rysunku poniżej zademonstrowano przypadki dla, których w zależności od długości sygnałów uzasadnione staje się użycie konkretnego sposobu liczenia splotu. W naszym przypadku projektowym długość ta jest wielokrotnie większa od 64, a zatem opisane zostaną tylko metody dla liczenia splotu poprzez FFT.



Rys. 2.9. Porównanie czasu wykonania splotu metodą tradycyjną i poprzez transformację Fouriera.
Rysunek z [2]

2.7.1. Splot liniowy

$$y(n) = x_1(n) * x_2(n) = \sum_{k=-\infty}^{\infty} x_1(k) \cdot x_2(n-k)$$

Widmo splotu sygnałów jest iloczynem widm poszczególnych sygnałów:

$$Y(j\omega) = X_1(j\omega) \cdot X_2(j\omega)$$

Dowód:

$$Y(j\omega) = \sum_{n=-\infty}^{\infty} e^{-jn\omega T_0} \sum_{k=-\infty}^{\infty} x_1(k)x_2(n-k) = \sum_{k=-\infty}^{\infty} x_1(k)e^{-jk\omega T_0} \sum_{n=k}^{\infty} e^{-j(n-k)\omega T_0} x_2(n-k)$$

Niezależnie od wartości zmiennej k sumowanie po zmiennej n prowadzi do widma sygnału $x_2(n)$:

$$Y(j\omega) = X_2(j\omega) \cdot \sum_{k=-\infty}^{\infty} x_1(k)e^{-jk\omega T_0}$$

Ponieważ:

$$X_1(j\omega) = \sum_{k=-\infty}^{\infty} x_1(k)e^{-jk\omega T_0}$$

$$Y(j\omega) = X_2(j\omega) \cdot X_1(j\omega)$$

Jeżeli oba sygnały $x_1(n)$ i $x_2(n)$ są ograniczone w czasie i składają się z N próbek o numerach od 0 do $N-1$, to sygnał $y(n)$ składa się z $2N-1$ próbek o numerach od 0 do $2N-2$, których wartość obliczymy ze wzoru:

$$y(n) = \sum_{k=\max(0, n-N+1)}^{\min(n, N-1)} x_1(n) \cdot x_2(n-k)$$

Jest to tzw. spłot liniowy sygnałów $x_1(n)$ i $x_2(n)$. DFT sygnału $y(n)$, składa się on z $2N-1$ próbek widma ciągłego. Zgodnie ze wzorem:

$$Y(j\omega) = X_1(j\omega) \cdot X_2(j\omega)$$

Próbki te wiążą się z próbkami widma sygnałów $x_1(n)$ i $x_2(n)$ w sposób następujący:

$$Y(k) = X_1(k) \cdot X_2(k)$$

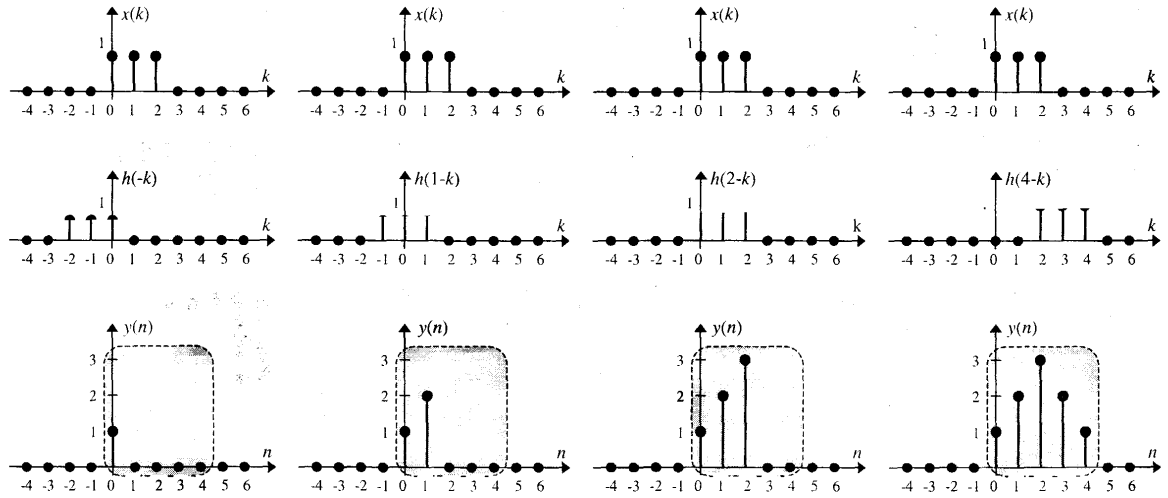
Zauważmy, że próbki widma tworzące DFT dla sygnału $y(n)$ pobierane są z krokiem $\Delta\omega = \Omega_0 / (2N - 1)$:

$$Y(k) = \sum_{n=0}^{2N-2} y(n) \cdot W_{2N-1}^{nk}$$

Oznacza to, że próbki sygnałów $x_1(n)$ i $x_2(n)$ pobierane są z takim samym krokiem:

$$X_1(k) = \sum_{n=0}^{N-1} x_1(n) \cdot W_{2N-1}^{nk}$$
$$X_2(k) = \sum_{n=0}^{N-1} x_2(n) \cdot W_{2N-1}^{nk}$$

Wynika z powyższych wzorów, że sygnały $x_1(n)$ i $x_2(n)$ uzupełniane są zerami do $2N-1$ próbek, liczone jest DFT (FFT) tak uzupełnionych sygnałów i dopiero wtedy iloczyn ciągów $X_1(k)$ i $X_2(k)$ liczony wyraz po wyrazie i poddany odwrotnej transformacie Fouriera tworzy wynik operacji splatania $y(n) = x_1(n) * x_2(n)$



Rys. 2.10. Zasada działania splotu liniowego. Rysunek z [1]

2.7.2. Splot kołowy

Założmy, że mamy dwa sygnały dyskretne $x_1(n)$ i $x_2(n)$ składające się z N próbek. Iloczyn DFT tych sygnałów:

$$Y(k) = X_1(k) \cdot X_2(k) = \left\{ \sum_{n=0}^{N-1} x_1(n) \cdot W_N^{nk} \right\} \times \left\{ \sum_{n=0}^{N-1} x_2(n) \cdot W_N^{nk} \right\}$$

to nic innego jak DFT sygnału składającego się także z N próbek, zwanego splotem kołowym sygnałów $x_1(n)$ i $x_2(n)$:

$$Y(k) = \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} x_1(n) \cdot x_2(m) \cdot W_N^{(n+m)k}$$

Należy zauważyć, że suma wskaźników $\nu = n + m$ przyjmuje wartości od 0 do $2N-2$, ale czynnik $W_N^{\nu k}$ przyjmuje tylko N różnych wartości:

$$W_N^{(\nu+N)k} = e^{-j \frac{2\pi(\nu+N)k}{N}} = e^{-j \frac{2\pi\nu k}{N}} e^{-j 2\pi k} = W_N^{\nu k}$$

Splot kołowy sygnałów $x_1(n)$ i $x_2(n)$ przepiszemy w postaci sumy N składników, z których każdy jest proporcjonalny do czynnika $W_N^{\nu k}$:

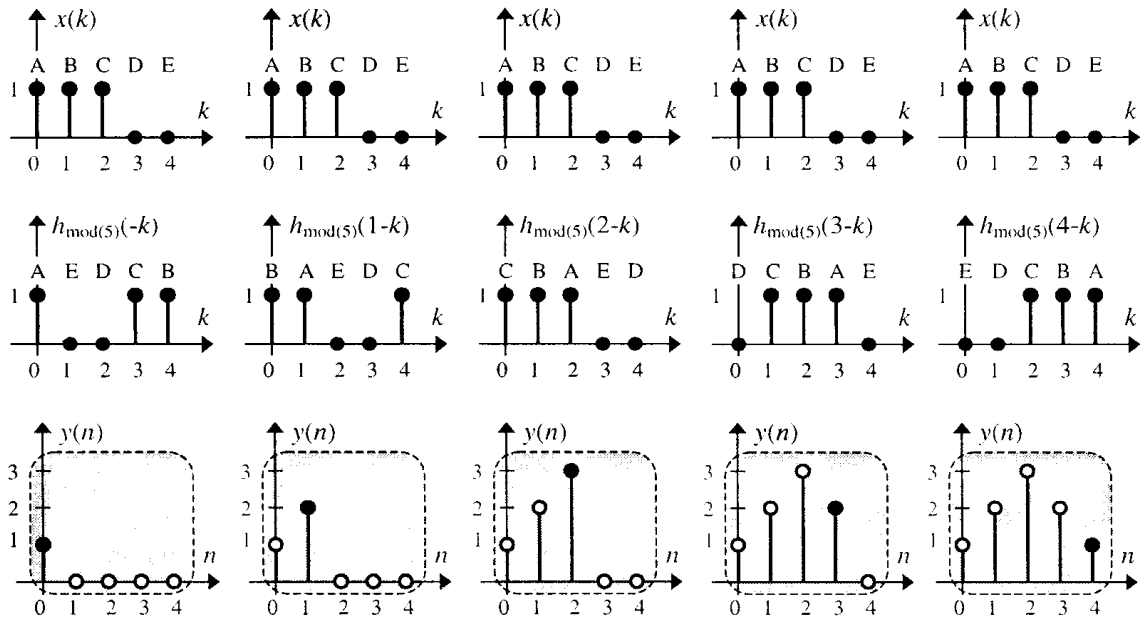
$$Y(k) = \sum_{\nu=0}^{N-1} \left(\sum_{\{0 \leq n, m \leq N-1 \wedge \text{mod}(n+m, N) = \nu\}} x_1(n) \cdot x_2(m) \right) \cdot W_N^{\nu k}$$

A zatem transformowany sygnał dany jest wzorem:

$$y(\nu) = \sum_{\{0 \leq n, m \leq N-1 \wedge \text{mod}(n+m, N) = \nu\}} x_1(n) \cdot x_2(m)$$

$$y(\nu) = \sum_{n=0}^{N-1} x_1(n) \cdot x_2(\text{mod}(N + \nu - n, N))$$

I powyższy wzór statecznie określa spłot kołowy sygnałów $x_1(n)$ i $x_2(n)$.



Rys. 2.11. Zasada działania spłotu kołowego Rysunek z [1]

2.8. Algorytm sekcjonowanego szybkiego spłotu sygnałów dyskretnych

Podstawowe algorytmy liczenia spłotu zapewniają splatanie dwóch bloków danych, ale nie nadają się do zastosowania w sytuacji, gdy dane napływają na bieżąco. Rozwiązaniem na to jest dzielenie ciągu danych na mniejsze bloki i wykonywanie spłotu na nich, a następnie na umiejętnym ich łączeniu w całość. Do najpopularniejszych metod liczenia spłotu przy wyżej wymienionych warunkach należą metody overlap-save i overlap-add. Nazwy tych metod wiernie odzwierciedlają zasadę działania. Overlap (ang. zachodzące), zatem w pierwszej z tych metod zachodzące próbki są pamiętane dla

następnego bloku, natomiast w drugiej metodzie próbki nie są pamiętane, jednak zachodzące na siebie elementy należy następnie ze sobą zsumować.

2.8.1. Metoda „overlap-save”

W metodzie tej sygnał przetwarza się realizując splot kołowy. Działanie metody zostanie wyjaśnione na poniższym przykładzie. Niech długość filtru wynosi $M=7$, a długość transformaty FFT, za pomocą której wykonywane będą sekwencyjne obliczenia wynosi $N=16$. Wartość ta wynika z założenia, iż musi być ona najmniejszą liczbą będącą potęgą liczby 2 i większą od parametru M . Oczywiście transformata może być dłuższa niż N – punktowa, jednak należy pamiętać, że czas wykonania takiej transformaty oprócz wzrostu rośnie wprost proporcjonalnie do N , a także dodatkowo o $\log_2(N)$.

Tylko raz na samym początku $M=7$ elementów filtru $h(n)$ jest uzupełniana $N-M=9$ zerami w celu obliczenia 16 punktowej transformaty Fouriera.

Zatem:

$$H(k) = \text{FFT}(h(n));$$

Kolejnym krokiem jest podział sygnału wejściowego $x(n)$ na mniejsze bloki o długości wynikającej wprost z zastosowanej długości FFT. W naszym przykładzie sygnał $x(n)$ dzielimy na kolejne bloki, nazywając je: $x_1(n)$, $x_2(n)$, $x_3(n)$ itd.

Bloki te jednak nie są kolejnymi elementami sygnału $x(n)$. Mianowicie zachodzą na siebie $M-1 = 6$ próbkami. Na tak przygotowanych blokach wykonuje się kolejne N - punktowe FFT każdego kolejnego sygnału $x_i(n)$.

$$X_i(n) = \text{FFT}(x_i(n));$$

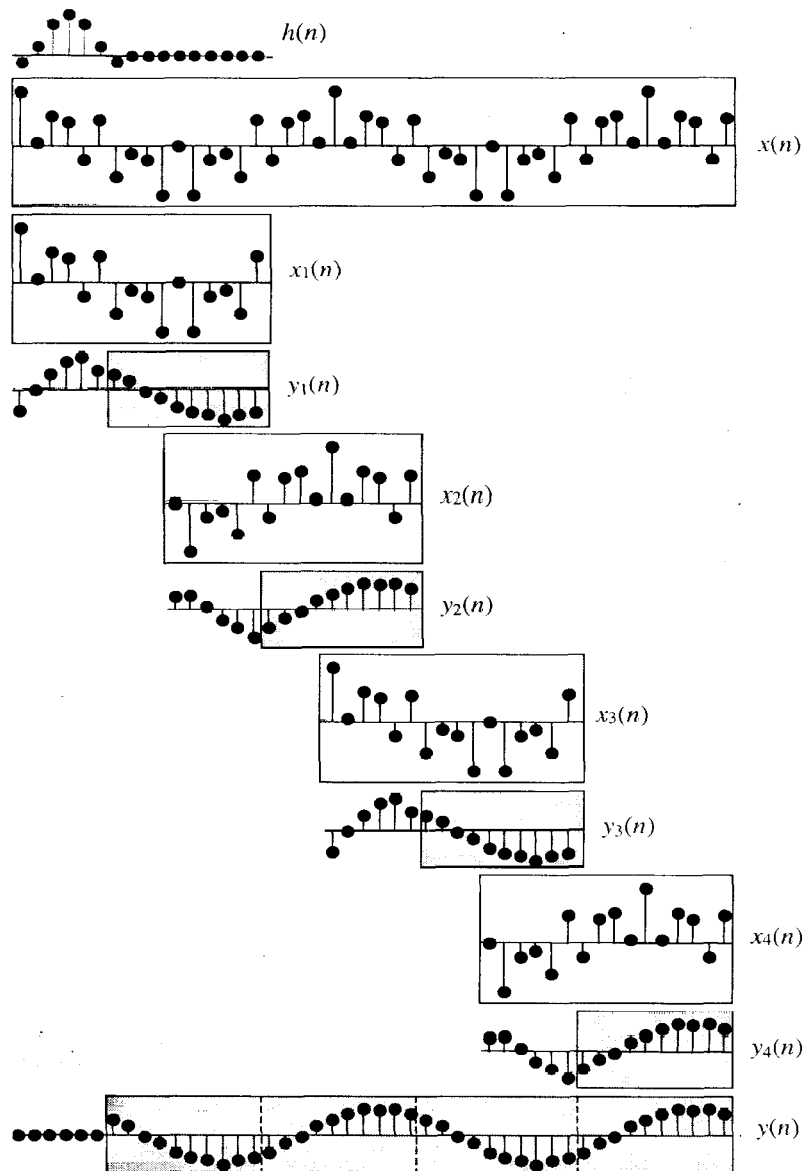
Otrzymane widmo $X_i(k)$ jest wymnażane z odpowiedzią częstotliwościową filtra $H(k)$.

Na tak przygotowanych widmach wykonuje się wsteczną transformatę i uzyskuje kolejne sygnały $y_i(n)$

$$y_i(n) = \text{IFFT}(X_i(k)H(k))$$

Ponieważ w każdym sygnale pierwsze $M-1=6$ próbek jest niepoprawnych, zostają one pomijane, a do bloku $y(n)$ dodawane jest $N-(M-1)=10$ próbek od indeksu 7 do 16.

Aby zapewnić poprawną pracę tej metodzie w pierwszej fazie dla próbki o $i=1$ nie należy wypełniać ważną treścią, a jedynie wypełnić je próbkami o wartości = 0.



Rys. 2.12. Overlap – save. Rysunek z [1]

2.8.2. Metoda „overlap-add”

W metodzie tej sygnał przetwarzany jest tym razem za pomocą splotu linowego. Weźmy przykład dla $M=7$ oraz $N=10$, gdzie wartość tego parametru będzie oznaczać ilość elementów jednorazowo pobieranych z bloku sygnału wejściowego $x(n)$. Długość transformaty FFT wynosić będzie $K=N+(M-1) = 16$. Podobnie jak w w/w metodzie sygnał $h(n)$ jest uzupełniany $K-M=9$ zerami i tak samo jak poprzednio tylko raz transformowany do dziedziny częstotliwości.

$$H(k)=\text{FFT}(h(n));$$

Kolejnym krokiem jest podział próbek sygnału wejściowego y_i na szereg mniejszych $x_1(n)$, $x_2(n)$, $x_3(n)$... itd., jednak z tą różnicą że poszczególne próbki nie będą zachodzić na siebie i będą miały długość 10 punktów.

Każdy kolejny sygnał $x_i(n)$ jest uzupełniany 9 zerami, aby zapewnić możliwość wykonania 16 punktowej FFT.

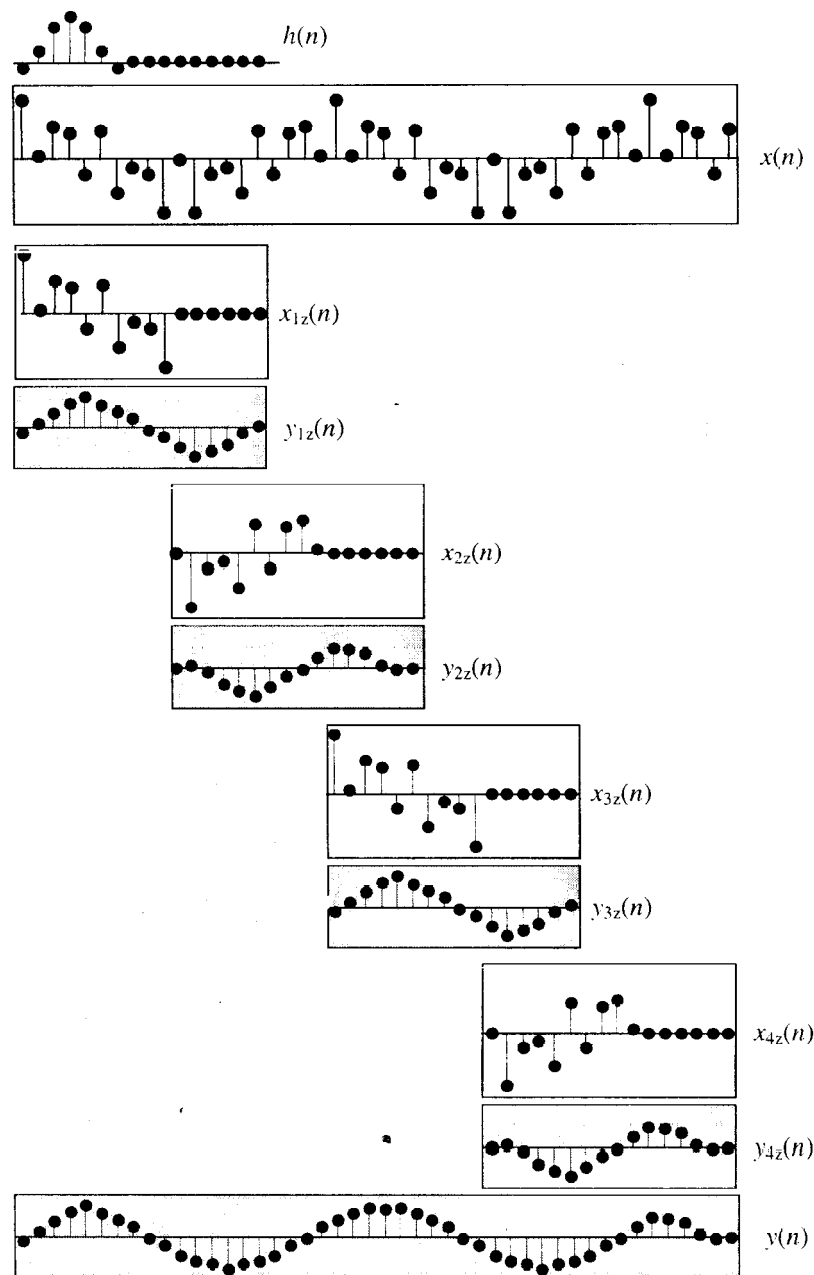
$$X_i(n) = \text{FFT}(x_i(n));$$

Otrzymane widmo $X_i(k)$ jest mnożone z odpowiedzią częstotliwościową filtra $H(k)$.

Na tak przygotowanych widmach wykonuje się wsteczną transformatę i uzyskuje kolejne sygnały $y_i(n)$

$$y_i(n) = \text{IFFT}(X_i(k)H(k))$$

Ponieważ sygnały zachodzą na siebie $K-N=6$ próbkami należy więc zachodzące próbki zsumować ze sobą.



Źródło: opracowanie własne autora, na podstawie publikacji [1].

Rys. 2.13. Overlap – Add. Rysunek z [1]

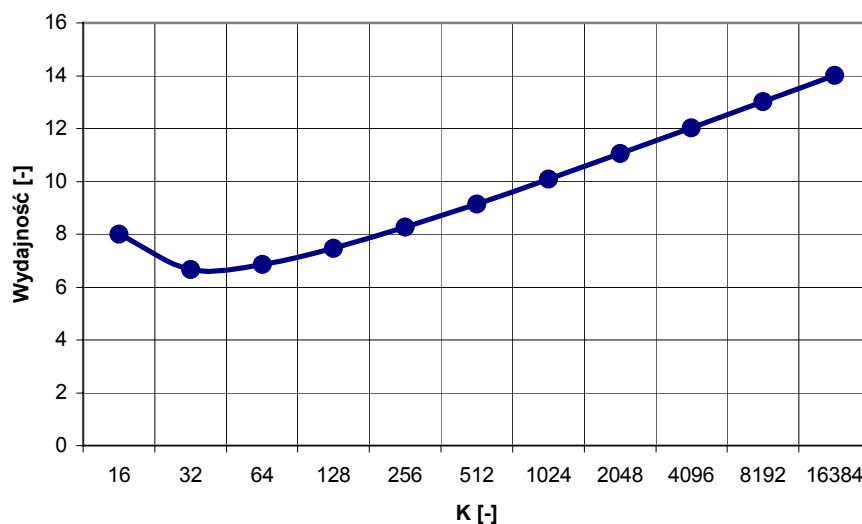
Do rozwiązania problemu w projekcie zastosowano algorytm overlap-save. Wybrano tę metodę, ponieważ posiada większą wydajność i mniejszą złożoność koniecznych operacji do wykonywania w trakcie pracy. W metodzie overlap-save próbki danych są pobierane blokowo bez modyfikacji w postaci uzupełniania zerami do długości użytej transformaty w odróżnieniu od metody Overlap-Add. Kolejną zaletą rozwiązania

jest to, że w metodzie tej wynik jest składany sekwencyjnie w kolejności porządkowej, w metodzie Overlap-Add zachodzące na siebie wyniki należy dodatkowo zsumować, co z kolei podraża dodatkowo zapotrzebowanie na moc obliczeniową. Jedyną wadą metody overlap-save jest to, że pierwszy blok należy odpowiednio przygotować, ale taka czynność jest wykonywana tylko raz w całym przebiegu wykonywania obliczeń.

Należy pamiętać o tym, iż w powyższych metodach blok jednorazowo przetwarzanych danych musi być większy od liczby próbek odpowiedzi impulsowej filtra. Wadą tego jest stosowanie opóźnień o tyle dłuższych na ile długa jest odpowiedź impulsowa. W naszym modelu najkorzystniejszym zatem przypadkiem jest wariant, w którym uczestniczy duża ilość komputerów o podobnych możliwościach obliczeniowych. W ten sposób np. przy stosowaniu długiej odpowiedzi o długości np. 256000 próbek i np. 20 komputerach o podobnych parametrach obliczeniowych jesteśmy w stanie wykorzystać 16k transformatę FFT, co przy założeniu częstotliwości próbkowania 44100 Hz powoduje, że opóźnienie może wynosić poniżej pół sekundy. Oczywiście jest to, że wystarczy jeden komputer wykorzystujący dłuższą transformatę, a osiągnięcie takiego rezultatu jest już niemożliwe, wewnętrzny bufor będzie zmuszony wymusić oczekiwanie na nadejście wyników obliczeń od tego komputera. Wniosek z tego rozumowania jest taki, że oprócz maksymalnego wykorzystania mocy procesora istotnym z e względu jakościowego może być dobieranie tak parametrów, aby długości FFT nie były zbyt zróżnicowane. Najbardziej racjonalnym wydaje się konfiguracja, w której wszystkie komputery korzystają z tej samej długości transformaty modyfikując jedynie część odpowiedzi impulsowej, jaką przydziela im się do obliczeń.

Bardzo istotnym problemem jest wybranie optymalnej wartości długości FFT. Korzystając z poprzedniego przykładu tzn. $M=9$ oraz $K=16$, czyli za każdym obliczeniem jesteśmy w stanie obliczyć efektywnie tylko 8 próbek (niezależnie od obranej metody).

M	K	K-M	Koszt = $K \cdot \log_2(K)$	Koszt jednostkowy
8	16	8	64	8
8	32	24	160	6.67
8	64	56	384	6.86
8	128	120	896	7.47
8	256	248	2048	8.26
8	512	504	4608	9.14
8	1024	1016	10240	10.08
8	2048	2040	22528	11.04
8	4096	4088	49152	12.02
8	8192	8184	106496	13.01
8	16384	16376	229376	14.01



Rys. 2.14. Zależność wydajności obliczeniowej w zależności od długości transformaty

Jak widać z prostej analizy przyjęta wartość $K=16$ nie jest wartością optymalną, jednakże w realnych warunkach projektowych wartość K przyjmuje wartości dużo większego rzędu, zatem nie ma zbyt dużej możliwości w doborze tego parametru.

2.9. Inne zastosowania DFT

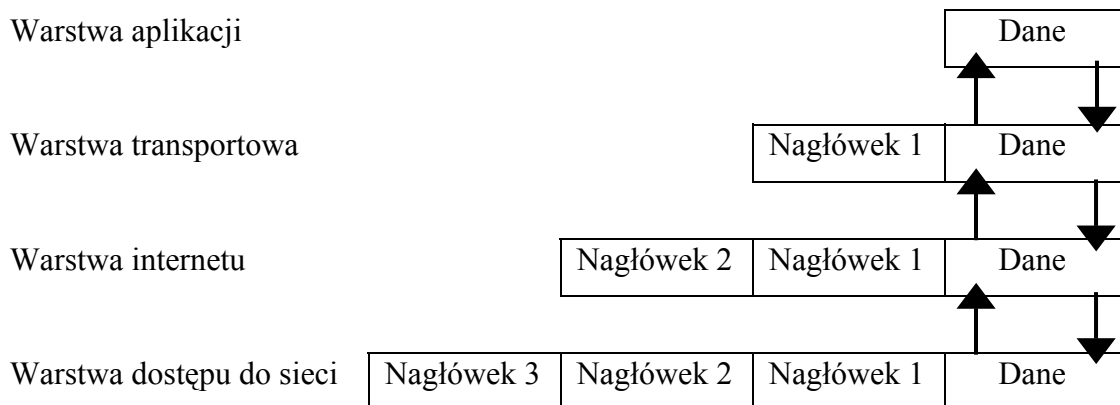
Oprócz cyfrowego przetwarzania sygnałów algorytm ten często jest wykorzystywany w informatyce. Wykorzystuje się go do działań na wielomianach oraz konstruując algorytmy arytmetyki wieloprecyzyjnej. W przypadku operacji na wielomianach najczęściej DFT stosuje się do Identyfikacji współczynników wielomianu oraz ich szybkiemu mnożeniu.

3. Wybrane zagadnienia protokołu sieciowego TCP/IP

3.1. Wstęp

Niniejszy rozdział stanowi zebranie najistotniejszych zagadnień związanych z protokołem TCP/IP, stanowiących minimum wiedzy potrzebnej do zrozumienia zasady działania przykładu projektowego związanego z transmisją sieciową. Po szersze wyjaśnienie tematu zaleca się do sięgnięcia do pozycji [4, 5, 6]. W dodatku D umieszczono spis ważniejszych dokumentów RFC (ang. Request For Comments), w których można poznać szczegółową zasadę działania poszczególnych elementów transmisji.

Warstwy OSI protokołu TCP/IP są ograniczone do czterech warstw. Poniżej zostały one wymienione wraz z zasygnalizowaniem sposobu wymiany danych. Wymiana danych przyjmuje postać tzw. „kopertowania” informacji tzn. dodawania przez kolejne warstwy specyficznych dla siebie nagłówków.



Rys. 3.1. Proces enkapsulacji danych

Zaletą protokołów TCP/IP jest to, że są odseparowane od techniki przesyłania danych, co umożliwia budowanie intersieci z wielu różnorodnych sieci i łączy danych.

3.2. Budowa pakietu IP

Protokół sieciowy IP jest podstawowym protokołem w zestawie TCP/IP. Wszystkie dane z TCP i UDP przesyłane są poprzez datagramy IP. Usługa dostarczania pakietów charakteryzuje się zawodnością i beipołączeniowością.

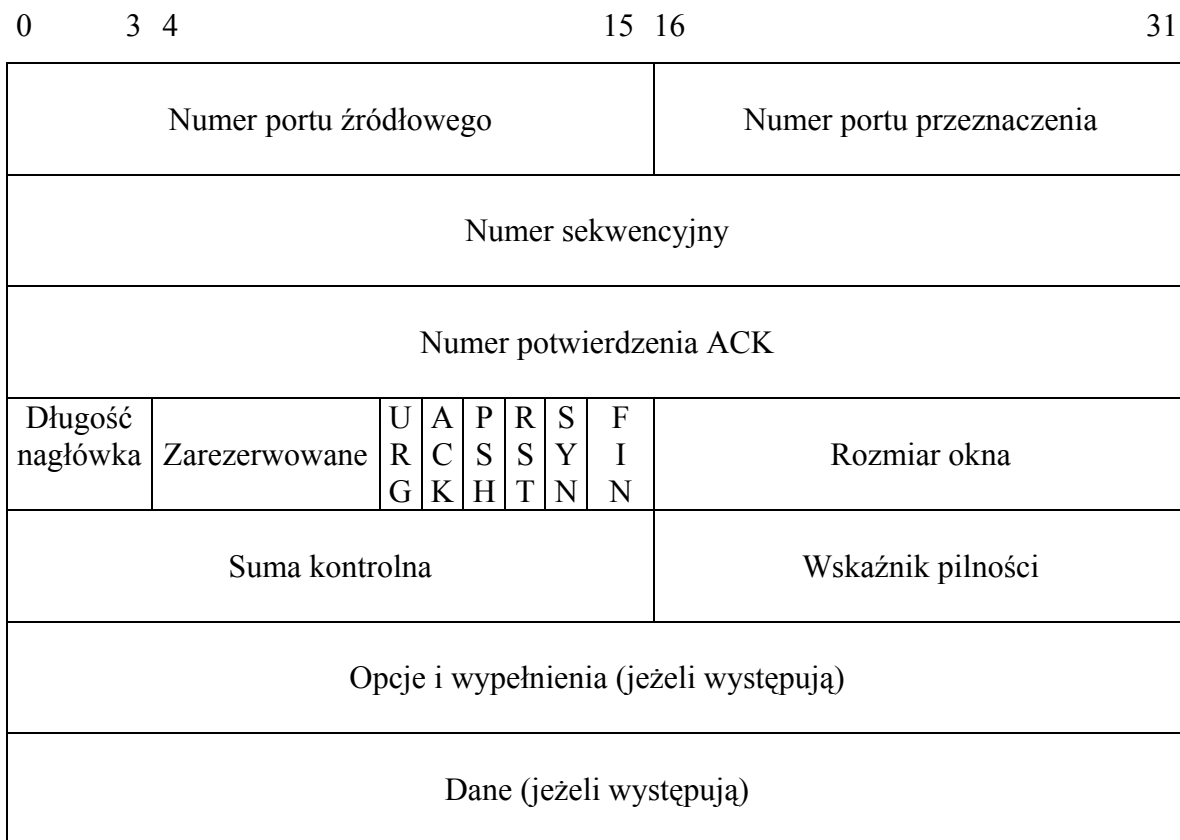
Zawodność oznacza, że nie ma żadnej gwarancji dostarczenia datagramu do odbiorcy. IP stara się przesyłać dane do odbiorcy, jednak jeżeli coś się nie uda (np. przepełnienie buforów), to datagram zostanie po prostu porzucony a IP uruchomi prostą sygnalizację wystąpienia błędu: wyśle zwrócić wiadomość ICMP, która jednak również nie musi dotrzeć w sposób pewny do nadawcy. Lepsze mechanizmy nadzorowania stanu transmisji muszą zapewniać protokoły wyższych rzędów (np. TCP).

Beipołączeniowość oznacza, że protokół IP nie zachowuje żadnej informacji o datagramach, które zostały już wysłane do odbiorcy. Poszczególne porcje informacji mogą więc docierać odmienną drogą, w innej kolejności niż porządek wysłania w sytuacjach awaryjnych, mogą być zniszczone lub powielone itp. Zwłaszcza kierunek wysyłania (z A do B lub z B do A) powoduje najczęstszą zmianę parametrów przesyłania informacji.

0	4	8	16	18	19	31
Wersja protokołu IP	Długość nagłówka n*4 bajty	Typ usługi TOS	Całkowita długość datagramu			
Identyfikacja			Znaczniki	Przesunięcie fragmentacji		
Czas życia datagramu TTL		Typ protokołu	Suma kontrolna nagłówka			
Adres źródłowy						
Adres docelowy						
Opcje nagłówka (jeżeli występują)						
Dane						

Rys. 3.2. Budowa nagłówka IP

3.3. Budowa Pakietu TCP



Rys. 3.3. Budowa pakietu TCP

Długość w bajtach	8	6	6	2	46 - 1500	4
Zawartość	Preambuła	Adres docelowy	Adres źródłowy	Długość	Dane	FCS (CRC)

Rys. 3.4. Pakiet ethernetowy

TCP jest protokołem działającym w warstwie transportowej wykorzystując warstwę sieci (IP) dostarcza usług aplikacjom. TCP, jest protokołem połączeniowym, co oznacza, iż dwie aplikacje chcące wymieniać dane pomiędzy sobą muszą uprzednio nawiązać połączenie.

Niezawodność protokołu wynika między innymi z:

- Potwierdzania otrzymanych danych. Po otrzymaniu pakietu (lub w mechanizmie okna po otrzymaniu pakietów o ilości równej szerokości okna) odbiorca informuje nadawcę o otrzymaniu danych;
- Sprawdzania sum kontrolnych nagłówka datagramu i danych. W przypadku niezgodności pakiet jest uważany za nieważny (błędny) i niepotwierdzany, przez co nadawca nie otrzymując informacji zwrotnej od odbiorcy dokonuje retransmisji;
- Retransmisje niepotwierdzonych danych. Brak potwierdzenia w określonym czasie oznacza konieczność ponownego wysłania pakietu;
- Porządkowanie przychodzących pakietów. W TCP pakiety są przesyłane jako datagramy IP, które mogą być wysyłane w dowolnej kolejności, mogą także nadchodzić w innej kolejności niż je wysłano. W takim przypadku odbiorca porządkuje je;
- Odrzucaniu powielonych danych wynikających z niedoskonałości IP.

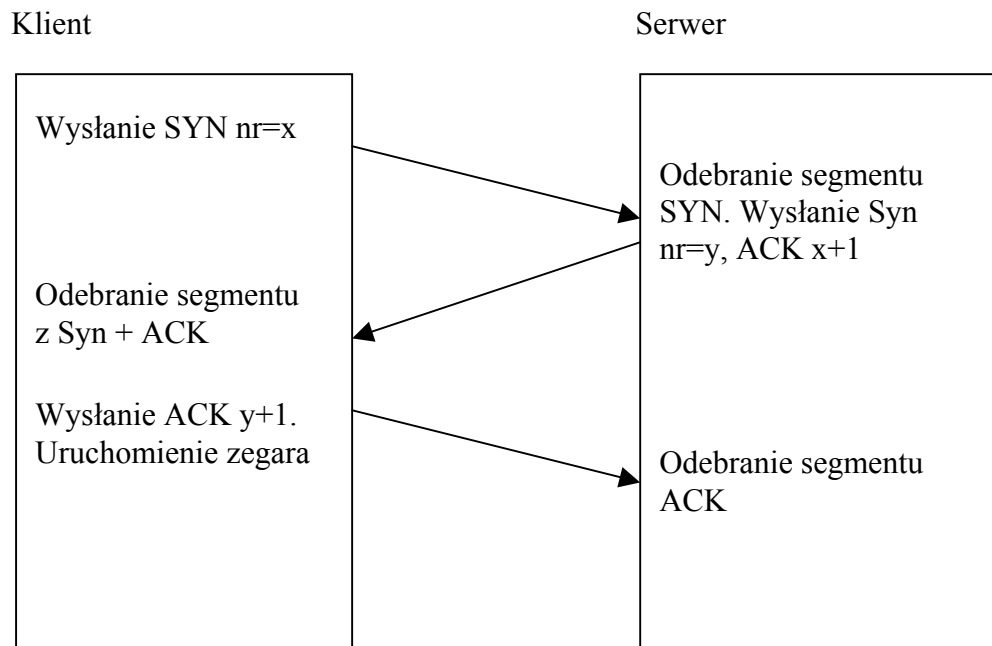
Segmentacja danych powoduje, iż w przypadku awarii lub kolizji retransmitowana jest tylko część całej informacji.

3.4. Nawiązanie połączenia TCP

Proces nawiązania połączenia składa się z następujących kroków:

- Strona, która wysyła zapytania (zwykle zwana klientem) nadaje segment SYN, określający numer portu serwera, z którym klient chce się połączyć, a także początkowy numer sekwencyjny klienta;
- Serwer odpowiada, wysyłając własny segment SYN zawierający początkowy numer sekwencyjny serwera. Ponadto serwer potwierdza odebranie segmentu SYN klienta, wysyłając (ACK) z nadesłanym przez klienta ISN plus jeden;
- Klient potwierdza nadesłany przez serwer segment SYN – wysyłając ACK z ISN serwera powiększony o jeden.

Wymiana tych trzech segmentów kończy ustalanie połączenia. Taki sposób nawiązywania połączenia jest często nazywany trójstronnym uzgodnieniem (three way handshake).

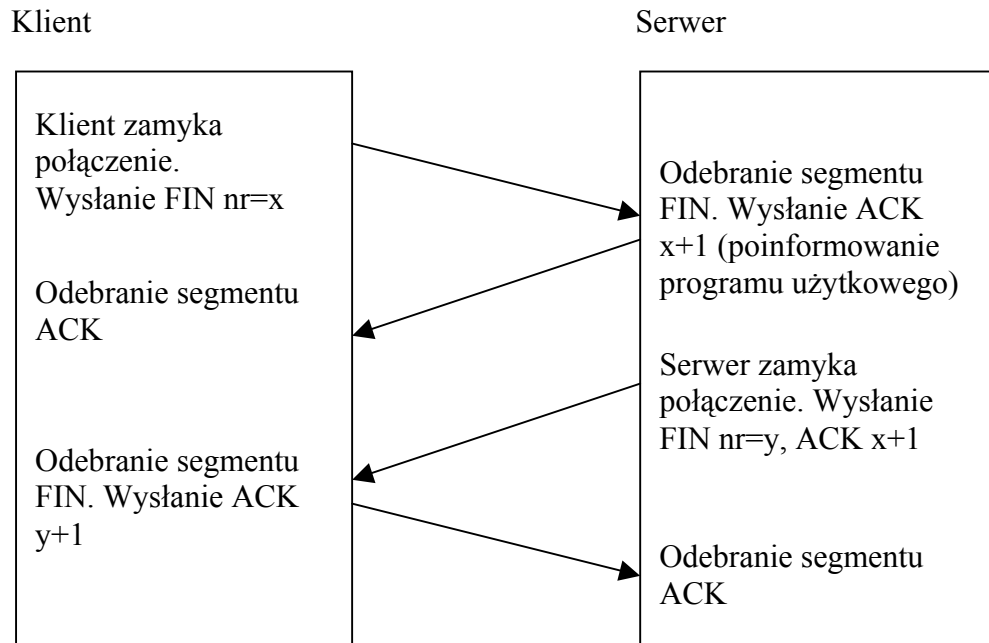


Rys. 3.5. Nawiązanie połączenia TCP

Strona, która wysyła pierwszy SYN wykonuje tak zwane aktywne otwarcie. Druga strona, która odbiera SYN i wysyła w odpowiedzi segment SYN, wykonuje tak zwane pasywne otwarcie.

3.5. Zakończenie połączenia TCP

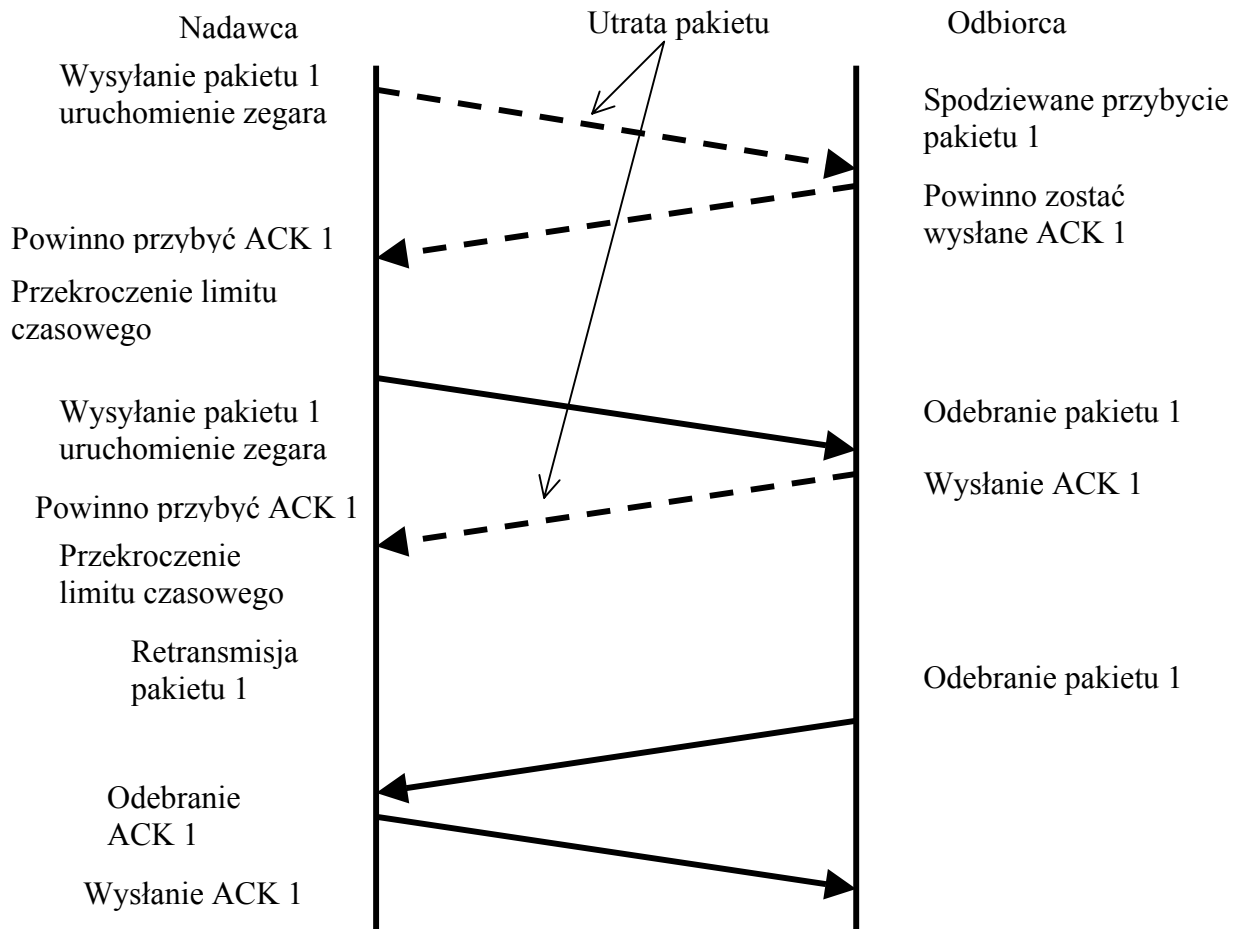
Ponieważ połączenie, TCP jest połączeniem full-duplex, to każdy z kierunków musi zostać zamknięty niezależnie. Zasadą jest, że jeden z końców wysyła pakiet FIN po zakończeniu przesyłania danych. Kiedy stacja odbierze segment FIN, musi powiadomić aplikację, że druga strona połączenia zakończyła ten kierunek przepływu danych. Odebranie FIN oznacza jedynie, że w tym kierunku połączenia nie będą płynęły już dane. TCP może nadal wysyłać dane po odebraniu FIN (połączenie półzamknięte). W celu pełnego zamknięcia połączenia druga strona musi wykonać podobną sekwencję operacji (FIN, oraz potwierdzenie ACK FIN).



Rys. 3.6. Zamykanie połączenia TCP

3.6. Utrata pakietu, retransmisja

Kiedy podczas transmisji w protokole TCP wysyłany jest segment, to jednocześnie uruchamiany jest zegar, który rozpoczyna oczekiwanie na potwierdzenie odebrania segmentu przez drugą stronę. Jeżeli potwierdzenie nie nadejdzie w określonym czasie segment jest wysyłany powtórnie.



Rys. 3.7. Schemat przesyłania z potwierdzeniami

Kiedy oprogramowanie TCP wysyła dane na drugi koniec połączenia, wtedy żąda od partnera przekazania połączenia. Jeżeli nie otrzyma potwierdzenia, to automatycznie ponownie wysyła dane i przez dłuższy czas czeka na potwierdzenie. Po pewnej liczbie ponownych transmisji oprogramowanie TCP zaniecha wysyłania danych, przy czym cały czas przeznaczony na próby wysyłania danych wynosi od 4 do 10 minut (zależnie od implementacji systemu). Protokół TCP zawiera algorytmy przeznaczone do dynamicznego oszacowania czasu powrotu (oznaczanego skrótem RTT, ang. *round-trip time*) informacji od klienta do serwera, zatem wie jak długo ma czekać na potwierdzenie. Na przykład czas powrotu w sieci lokalnej może być mierzony w milisekundach, podczas gdy w sieci rozległej można go wyrażać w sekundach. Ponadto oprogramowanie TCP może zmierzyć, że między pewnym klientem a serwerem wartość RTT wynosiła najpierw 1 s, a po upływie 30 s dla tego samego połączenia urosła do 2 s, ponieważ zmieniło się natężenie ruchu w sieci.

Protokół TCP ustala kolejność danych, przypisując numer kolejny każdemu wysłanemu bajtowi. Przypuśćmy na przykład, że pewien program użytkowy odsyła 2048 bajtów do gniazda TCP, powodując, że oprogramowanie TCP wysła dwa segmenty danych, z których pierwszy zawiera dane opatrzone numerami kolejnymi 1-1024, drugi zaś - numerami kolejnymi 1025-1048. (Segment jest to jednostka danych przekazywanych przez warstwę TCP do warstwy IP). Jeżeli segmenty przybywają w innej kolejności, to odbierające oprogramowanie TCP porządkuje je na podstawie numerów kolejnych zawartych w nich bajtów, zanim przekaże dane do pobierającego je programu użytkowego. Jeżeli oprogramowanie TCP otrzyma zdublowane dane od swego partnera (przypuśćmy, że partner uznał, że jeden z segmentów zaginął i przesłał go ponownie, podczas gdy w rzeczywistości ten segment nie zaginął tylko sieć była chwilowo przeciążona), to może wykryć, że dane zostały zdublowane (na podstawie numerów kolejnych) i usunąć niepotrzebne kopie danych.

Protokół TCP zapewnia sterowanie przepływem (ang. *flow control*). Oprogramowanie TCP zawsze informuje swego partnera o tym, ile bajtów danych chce od niego przyjąć. Nazywa to się oknem oferowanym (ang. *advertised window*). Rozmiar okna jest zawsze równy rozmiarowi wolnego miejsca w buforze odbiorcy, dzięki czemu nadawca nie może spowodować przepełnienia tego bufora. Rozmiar okna zmienia się dynamicznie podczas przesyłania danych: gdy dane napływają od nadawcy, wówczas rozmiar okna się zmniejsza, ale w miarę jak program użytkowy czyta dane z tego bufora, zwiększa się też rozmiar okna. Rozmiar okna może wynosić 0. Oznacza to, że przyjmujący dane bufor odbiorczy dla tego gniazda jest pełny i trzeba poczekać, aż program użytkowy pobierze dane z bufora, zanim będzie można przyjąć więcej danych od partnera.

I wreszcie, połączenie TCP jest także w pełni dwukierunkowe (ang. *full-duplex*), to znaczy, że przez jedno połączenie program użytkowy może równocześnie wysyłać i przyjmować dane. Oznacza to, że oprogramowanie TCP musi śledzić informacje dotyczące połączenia, takie jak numery kolejne i rozmiary okna, dla każdego kierunku przepływu danych: wysyłania i odbierania.

3.7. UDP

Protokół datagramów użytkownika UDP (ang. *User Datagram Protocol*) cechuje się małym obciążeniem transmisji danych w postaci datagramów. Jego podstawową cechą odróżniającą go od TCP jest to, że jest protokołem bezpołączeniowym, zatem nie istnieje etap nawiązywania/zakończenia połączenia. Protokół ten nie zapewnia prawidłowej kolejności odbierania pakietów i nie sygnalizuje zagubienia pakietu, jest jednak wydajniejszy od protokołu TCP i może być przydatny w przypadku przesyłania niewielkich porcji informacji lub w sieciach, w których prawdopodobieństwo wystąpienia błędu transmisji jest bardzo małe, na przykład w sieciach Ethernet i prywatnych.

Nadawca wysyła dane licząc na to, że odbiorca jest w stanie je odebrać, jednak nie ma pewności czy aplikacja jest uruchomiona, czy udało się odczytać te dane lub czy odczytane dane są poprawne. Z tego też powodu transmisje w trybie UDP można przeprowadzać w aplikacji, w której uprzednio zestawiono połączenie TCP, skonfigurowano opcje połączenia i inne potrzebne informacje. W tym momencie obie aplikacje wiedzą o poprawności połączenia i są gotowe do wysyłania i odbierania danych (próbek sygnałów jak ma to miejsce w projekcie) korzystając z protokołu UDP. Protokół ten jest nieoceniony, jeżeli chcemy wysyłać duże ilości danych do wszystkich komputerów w sieci (tzw. tryb Broadcast), w tym momencie niezależnie od ilości komputerów w sieci (czy też klasy sieci) informacje będą mogły odczytać wszystkie komputery rozpoznając siebie jako odbiorcę (w tym przypadku adres IP będący adresem Broadcast).

W przypadku protokołu TCP tryb Broadcast jest niemożliwy do wykonania, zawsze będą istnieć nieobsadzone adresy IP (wolne miejsca w sieci bądź najzwyczajniej komputery odłączone od sieci), które nie będą mogły potwierdzić danych. Mechanizm TCP w momencie wysłania informacji do odbiorcy, którym jest cała sieć nigdy nie będzie dysponował informacją o tym, że wszystkie komputery otrzymały dane.

Budowa nagłówka UDP jest o wiele prostsza niż nagłówka TCP i zajmuje tylko 8 bajtów. Jako, że UDP jest protokołem zawodnym, nie ma obowiązku liczenia sumy kontrolnej, wykonanie tej czynności jest opcjonalne. Protokół ten z zamierzenia miał być wydajnym, minimalnym protokołem transportowym. W strukturze nagłówka są zawarte tylko te informacje, które mówią jakiej aplikacji przekazać datagram oraz informacje niezbędne do minimalnej kontroli błędów.

UDP nie posiada mechanizmów mierzenia czasu sterowania przepływem, zarządzaniem przeciążeniem, potwierdzania, przyspieszonego dostarczania pilnych danych ani żadnych innych. Usługa ta, bardzo przypomina pocztę pneumatyczną. Wrzucamy coś do sieci, a UDP transportuje to od punktu A do B - przynajmniej mamy taką nadzieję. Jeśli przesyłka nie dotrze do celu, to nikt nie zapewnia próby ponownej transmisji.

0	15	16	31
Numer portu źródłowego		Numer portu docelowego	
Długość komunikatu		Suma kontrolna	
Dane			

Rys. 3.8. Budowa nagłówka pakietu UDP

Dodatkowe pola niezbędne do obliczenia sumy kontrolnej UDP tworzą tak zwany pseudonagłówek. Skład pseudonagłówka jest przedstawiony na rysunku poniżej:

0	8	16	24	31
Adres źródłowy IP				
Adres docelowy IP				
0	Protokół	Długość UDP		
Port źródłowy		Port docelowy		
Długość UDP		Suma kontrolna		
Dane				

Rys. 3.9. Pseudonagłówek UDP

3.8. Adresacja

W sieci Internet komputery macierzyste mają przydzielone adresy IP, będące 32 bitowymi identyfikatorami numerycznymi jednoznacznie określającymi podsieć oraz identyfikator komputera, który jednoznacznie określa interfejs danego komputera w owej sieci. Adres IP jest zatem zbudowany z dwóch części:

- Net_id – adresu sieci
- Host_id – adresu komputera host w owej sieci

Dopiero obydwie części tworzą unikalny adres. Adres zwykle jest zapisywany w sposób dziesiętny podzielony na oktety kropkami.

Klasa	0	8	16	24	31
A	0	net_id	host_id		
B	10	net_id		host_id	
C	110	net_id			host_id
D	1110	adres grupowy			
E	11110	zarezerwowane dla przyszłych zastosowań			

Rys. 3.10. Klasy adresów sieciowych

Klasa	Zakres
A	1.0.0.0 – 126.0.0.0
B	128.1.0.0 – 191.254.0.0
C	192.0.1.0 – 223.255.254.0
D	224.0.0.0 – 239.255.255.254
E	240.0.0.0 – 255.255.255.255

Rys. 3.11. Zakresy adresów sieciowych

Klasa A ma zastosowanie w dużych sieciach, klasa D wykorzystywana jest do rozgłaszania wiadomości (multicast). Część host_id adresu nie może przyjmować samych zer, gdyż wówczas adres ten odnosi się do samej sieci, w przypadku samych jedynek oznacza wszystkie komputery w sieci.

3.9. Tryby rozsyłania danych

3.9.1. Unicasting

W tym trybie dane są wysyłane tylko do jednego komputera o niepowtarzalnym adresie korzystając z protokołu TCP

3.9.2. Broadcasting – rozgłaszanie danych

Pozwala na wysyłanie danych z jednej stacji roboczej do wielu innych stacji w lokalnej sieci komputerowej jednocześnie. Właściwość ta jest charakterystyczna dla protokołów bezpołączeniowych, ponieważ wszystkie komputery w lokalnej sieci komputerowej mogą odebrać i przetworzyć rozgłaszaną wiadomość. Wadą rozgłaszania jest to, że każdy komputer, nasłuchujący w sieci musi przetworzyć taką wiadomość. Załóżmy na przykład, że użytkownik rozgłasza wiadomość w sieci, a karta sieciowa każdego komputera podejmuje ją i przekazuje na stos sieciowy. Stos sprawdza następnie wszystkie aplikacje sieciowe, by stwierdzić, czy któraś z nich powinna otrzymać tę wiadomość. Zwykle zdecydowana większość komputerów w sieci nie jest nią zainteresowana i odrzuca te dane. Niestety, każdy komputer musi poświęcić czas na przetworzenie takiego pakietu i sprawdzenie, czy któraś z aplikacji go nie oczekuje. Dlatego zbyt intensywna transmisja rozgłoszeniowa może zablokować całą sieć.

3.9.3. Multicasting - przesyłanie grupowe

Polega na wysłaniu danych przez jeden proces i odbieraniu ich przez dowolną liczbę odbiorców. Metody nawiązywania sesji wielopołączeniowej przez proces różnią się między sobą w zależności od protokołu. Na przykład w protokole IP transmisja grupowa jest zmodyfikowaną wersją rozgłaszania. Transmisja grupowa IP wymaga, by wszystkie węzły zainteresowane wysyłaniem lub odbieraniem danych należały do specjalnej grupy. Gdy proces chce przyłączyć się do takiej grupy, w karcie sieciowej umieszczany jest filtr, dzięki, któremu jedynie dane związane z tym adresem grupowym będą pobierane przez sprzęt sieciowy i przekazywane na stos sieciowy w celu dalszego przetwarzania. Z transmisji grupowej korzystają często aplikacje do obsługi wideokonferencji.

4. Wybrane elementy systemów rozproszonych

4.1. Wstęp i historia

System rozproszony (ang. *distributed system*) jest zbiorem samodzielnych komputerów połączonych za pomocą sieci i wyposażonych w rozproszone oprogramowanie systemowe. Oprogramowanie systemu rozproszonego umożliwia komputerom koordynowanie ich działań oraz dzielenie zasobów systemu: sprzętu, oprogramowania i danych. Użytkownicy dobrze zaprojektowanego systemu rozproszonego powinni go odbierać jako jedno, zintegrowane środowisko obliczeniowe, pomimo że może ono być zrealizowane za pomocą wielu komputerów znajdujących się w różnych miejscach. Rozwój systemów rozproszonych nastąpił z chwilą pojawienia się lokalnych sieci komputerowych o dużej szybkości, na początku lat siedemdziesiątych. Zwiększenie mocy obliczeniowej komputerów osobistych spowodowało odejście od scentralizowanych komputerów wielostanowiskowych

4.2. Zastosowania

Znaczenie sieci rozległych zwiększyło się wraz ze wzrostem liczby przyłączonych do nich komputerów oraz poszerzeniem zakresu oprogramowania umożliwiającego ich użycie. W podpunkcie 4.4. postarano się omówić dwa z największych zastosowań mających realizację praktyczne. Należy jednak rozróżnić dwa przypadki obliczeń: Pierwszy, w którym dysponujemy bardzo dużą ilością informacji (np. próbki sygnałów akustycznych), dane są przetwarzane w czasie rzeczywistym i krytycznym parametrem jest przepustowość sieci, zatem takie obliczenia bardziej nadają się do stosowania w sieci lokalnej np. Ethernet, oraz obliczenia, które na podstawie ograniczonej porcji danych wyliczają wiele informacji (np. sieci neuronowe), w których obliczenia czasu rzeczywistego nie są konieczne, po prostu klient pobiera kolejne paczki danych w chwili, gdy zakończy obliczenia obecnych, wydłużenie czasu przetwarzania nie jest krytycznym parametrem ten rodzaj obliczeń bardziej nadaje się do obliczeń w sieci Internet.

4.3. Podstawowe własności sieci rozproszonych

Sieci rozproszone charakteryzują się sześcioma podstawowymi własnościami. Poniżej zostaną one wymienione i objaśnione.

4.3.1. Dzielenie zasobów

Termin zasobu charakteryzuje zbiór rzeczy, które zostają dzielone (wspólnie wykorzystywane) w systemie rozproszonym. Zasoby w systemie rozproszonym są fizycznie zamknięte w którymś z komputerów a dostęp do nich odbywa się za pomocą komunikacji. Tak też jest i w naszym projekcie. Wspólnym zasobem jest tu ciąg próbek pobieranych z karty dźwiękowej, a komputery pełniące role klientów, czyli maszyn obliczeniowych otrzymują ten zasób w postaci komunikatów.

Termin zarządca zasobów odnosi się do określenia modułu oprogramowania zarządzającego zbiorem zasobów. Jednym z najbardziej znanych i najszerzej adaptowanych modelem systemu jest model klient - serwer. W modelu tym wszystkie zasoby są utrzymywane i zarządzane przez procesy serwerów. Jeśli procesy klientów potrzebują dostępu do któregoś z zasobów zwracają się do serwera. Jeśli zamówienie jest poprawne serwer wykona żadaną czynność i odeśle odpowiedź do procesu klienta.

4.3.2. Otwartość

Przez otwartość rozumie się zdolność do rozszerzania różnymi sposobami. System może być otwarty lub zamknięty np. na rozszerzanie oprogramowania, czyli dodawania nowych właściwości do systemu operacyjnego, protokołów komunikacyjnych. Otwartość osiąga się przez określanie i dokumentowanie zasadniczych interfejsów programowych systemu rozproszonego i udostępnianie ich programistom.

4.3.3. Współbieżność

Jeżeli w jednym komputerze istnieje wiele procesów to znaczy, że są one wykonywane współbieżnie (równolegle). Gdy komputer jest komputerem wyposażonym w jeden procesor wówczas współbieżność osiąga się poprzez przeplatanie porcji poszczególnych procesów w czasie. W przypadku zastosowania N-procesorów można wtedy wykonać aż N-procesów jednocześnie zyskując na przepustowości obliczeniowej. Jeżeli przyjmiemy, że system składa się z M-komputerów jednoprocessorowych, wówczas możemy powiedzieć że działa aż M-procesorów równolegle co prawda na różnych komputerach.

4.3.4. Skalowalność

Systemy rozproszone powinny działać skutecznie i wydajnie przy różnej ilości jednostek obliczeniowych (oczywiście przy spełnieniu kryterium minimalnej mocy obliczeniowej, jeśli takie jest wymagane). Najmniejszy system rozproszony może składać się z dwu stacji roboczych i serwera.

Powiększanie skali systemu nie powinno pociągać za sobą konieczności wykonywania w nich zmian. Projektując system powinniśmy planować np. fakt, iż ze wzrostem ilości komputerów w sieci wzrośnie częstotliwość kontaktu z plikami, ilości wymienianych danych przez sieć itp. Musi istnieć możliwość dodawania kolejnych serwerów do autonomicznej sieci rozproszonej tak, aby odciążać coraz bardziej obciążony serwer, należy też pamiętać o bardzo istotnym kryterium, jakim jest sieć. Zbyt duża ilość komputerów o bardzo małych mocach obliczeniowych znacznie bardziej obciąża sieć (naddatki komunikatów, informacji...) niż mniejsza ilość komputerów, ale o większej mocy obliczeniowej z osobna, i pomimo podobnego rezultatu obliczeniowego w pierwszym przypadku szybciej dojdzie do sytuacji, w którym kryterium wydajności stanie się medium transmisyjne.

4.3.5. Tolerowanie uszkodzeń

Niestety należy zawsze przewidzieć i taki wariant pracy systemu. Aby stworzyć systemy tolerujące awarie sprzętu do jednego zastosowania stosuje się dwa połączone ze sobą komputery, z których jeden działa jako maszyna rezerwowa. Jest to jednak drogie rozwiązanie podwajające koszt sprzętu. Należy też uwzględnić awarie sprzętu po stronie klienckiej, w tym celu należy zawsze przyjmować kryterium minimalnej mocy obliczeniowej z pewnym zapasem tak, aby awaria przynajmniej jednego stanowiska nie powodowała potrzeby zatrzymania pracy systemu ze względu na brak mocy obliczeniowej. Niestety awaria medium transmisyjnego powoduje konieczność zakończenia obliczeń, pomimo tego warto zaimplementować mechanizmy wykrywające ten stan i sygnalizujące awarie.

4.3.6. Przeźroczystość

Jest to ukrywanie przed użytkownikiem, a także i programistą oddzielności składowych w systemie rozproszonym tak, że w rezultacie system postrzegany jest jak całość a nie złożenie serwera-zarządcy ze zbiorem klientów. Przeźroczystość systemu rozproszonego możemy podzielić na następujące części:

Przeźroczystość dostępu – umożliwia dostęp do lokalnych i odległych obiektów za pomocą identycznych działań;

Przeźroczystość położenia – umożliwia dostęp do obiektów bez wiedzy o ich lokalizacji.

Wraz z przeźroczystością dostępu są najważniejszymi z cech przeźroczystości;

Przeźroczystość współbieżności – wiele procesów może w sposób niezakłócony działać współbieżnie z użyciem wspólnych obiektów informacji;

Przeźroczystość zwielokrotniania – umożliwia użycie wielu kopii obiektów informacji w celu zwiększenia niezawodności i wydajności bez wiedzy użytkowników i programów użytkowych o zwielokrotnieniach;

Przeźroczystość awarii – umożliwia ukrywanie uszkodzeń pozwalając użytkownikom i programom użytkowym na dokończenie zadań pomimo awarii sprzętu lub składowych oprogramowania;

Przeźroczystość wędrówki - pozwala na przemieszczenie obiektów informacji w obrębie systemu bez wpływu na działanie użytkowników lub programów użytkowych;

Przeźroczystość wydajności – pozwala na rekonfigurowanie systemu w celu poprawy działania przy zmianie obciążenia;

Przeźroczystość skalowania – umożliwia systemowi i jego zastosowaniom rozszerzanie skali bez zmiany struktury systemów lub algorytmów użytkowych.

4.4. Przykłady projektów związane z obliczeniami rozproszonymi w sieci.

a) Projekt SETI@Home - The Search for ExtraTerrestrial Intelligence.

Projekt ten jest przedsięwzięciem naukowym, wykorzystującym moc obliczeniową komputerów podłączonych do Internetu do poszukiwań sztucznie wytworzonych sygnałów pochodzących od cywilizacji pozaziemskich w emisji radiowej kosmosu.

W projekcie może uczestniczyć każdy użytkownik Internetu. Wzięcie udziału w Projekcie wymaga jedynie zainstalowania na komputerze programu klienta SETI@Home i utworzenia indywidualnego konta uczestnika.

b) Distributed.Net .

Distributed.Net jest organizacją koordynującą poprzez sieć internet działania komputerów na całym świecie w celu uzyskania jak największej mocy obliczeniowej.

Podejmują się zadań, z którymi nawet najszybsze komputery na świecie musiałyby zajmować się setkami lat. Jednym z zadań jest projekt RC5. Zapoczątkowany został przez firmę RSA, która ogłosiła nagrodę dla osoby, która odczyta wiadomość zaszyfrowaną ich kodem z 64-bitowym kluczem. Distributed.net posiadająca już doświadczenia w tego typu zadaniach (19.10.1997 r. rozwiązała poprzednie zadanie RSA - kod z 56-bitowym kluczem w 250 dni) podjęła się i tego zadania. Równolegle z projektem RC5 prowadzone są przez distributed.net inne projekty, jak złamanie kodu DES-56 (niecałe 24 godziny), złamanie kodu CSC-56 w konkursie ogłoszonym przez CS Communications, wyliczenie optymalnych linijek golombowskich.. Sposób liczenia jest prosty. Klient ściąga z serwera bloki kluczy do przeliczenia. Po przeliczeniu-sprawdzeniu czy któryś z nich rozkodowuje zaszyfrowaną wiadomość. A kluczy jest do sprawdzenia 2^{64} kombinacji

c) Inne projekty

Powstało wiele różnych innych projektów wykorzystujących sieć do obliczeń rozproszonych, np. obliczanie liczby Pi czy też szczepionki na raka.

4.5. Narzędzia do tworzenia obiektowych programów rozproszonych w środowiskach sieciowych.

Mechanizm gniazdek (ang. sockets) jest interfejsem do przekazywania danych między procesami działającymi na różnych komputerach. Wymiana danych następuje między gniazdkami, które mogą wysyłać lub odczytywać dane.

Komunikacja może odbywać się dwoma sposobami: z tworzeniem połączenia (TCP) lub bez połączenia (UDP). O typie komunikacji trzeba zdecydować już w chwili tworzenia gniazdka - oba końce (nadawca i odbiorca) muszą mieć ten sam typ. Przed wysłaniem pierwszych danych należy wykonać następujące czynności:

- utworzyć gniazdko (po stronie komputera pełniącego rolę serwera),
- przydzielić mu wolny numer portu (możliwe jest utworzenie gniazda o tym samym numerze portu dla protokołu TCP i UDP oddzielnie),
- utworzyć połączenie (jeśli ma to być komunikacja z połączeniem) inicjowane ze strony komputera – klienta.

Mechanizm gniazdek jest tylko interfejsem, którego można używać do przesyłania danych, w miarę niezależnie od protokołu używanego w niższych warstwach. Nie wszystko jednak jest realizowane niezależnie od protokołu. Przykładem mogą być adresy. Adres gniazdka składa się z adresu komputera i numeru portu. Adresy maszyn wyglądają różnie w przypadku różnych protokołów. Z tego względu adres komputera składa się z pola określającego formę (rodzinę) adresów oraz z adresu używanego przez dany protokół. Taka struktura zwykle jest rzutowana na strukturę `sockaddr` używaną przez funkcje biblioteki gniazdek. Przy tworzeniu gniazdka należy podać formę adresów oraz protokół używany przez dane gniazdko. Dwa gniazdko będą mogły wymieniać dane tylko wtedy, gdy będą używały tego samego protokołu. Biblioteka gniazdek umożliwia w stosunkowo prosty i efektywny sposób realizację wymiany danych między procesami wykonywanymi przez jeden lub kilka różnych komputerów. Dzięki niemal identycznej implementacji, w różnych środowiskach możliwe jest pisanie oprogramowania przenośnego.. Te zalety zdecydowały o moim wyborze

4.6. Podsumowanie

Systemy rozproszone stały się normą w organizowaniu możliwości obliczeniowych. Mogą być użyte do realizacji interakcyjnych systemów komputerowych ogólnego przeznaczenia, w stylu systemu UNIX, oraz do wspomagania szerokiej gamy handlowych i przemysłowych zastosowań komputerów. Coraz częściej stosuje się je jako bazę do nowych zastosowań w dziedzinach takich, jak sieciowe usługi informacyjne i zastosowania multimedialne, w których podstawowym wymaganiem jest komunikacja.

We wszystkich tych funkcjach systemy rozproszone są w stanie zaoferować swoim użytkownikom znaczące korzyści. Wyróżniliśmy ich podstawowe własności, identyfikując je za pomocą następujących pojęć: dzielenie zasobów, otwartość, współbieżność, skalowalność, tolerowanie uszkodzeń i przezroczystość. Przynoszone przez nie korzyści nie są osiągane automatycznie, lecz zależą od starannego zaprojektowania składowych systemu. Sposoby realizacji tych funkcji są głównym tematem następnego rozdziału.

Dzielenie zasobów jest podstawową cechą systemów rozproszonych. Podtrzymuje ono inne właściwości omówione w tym rozdziale i silnie oddziałuje na architektury oprogramowania występujące w systemach rozproszonych. Zasoby mogą być jednostkami danych, oprogramowania lub składowymi sprzętowymi. Zasoby takie odróżniają się od jednostek

danych zarządzanych wewnątrz poszczególnych procesów potrzebą wspólnego ich użytkowania przez kilka procesów. Powoduje to konieczność zewnętrznego zarządzania nimi w stosunku do procesów, które z nich korzystają. W systemach klient-serwer przybiera to formę procesu serwera lub - ogólniej - usługi dostarczanej przez kilka współpracujących procesów. Systemy oparte na obiektach mogą dostarczać elastyczniejszego modelu zarządzania wspólnymi zasobami, jednak najlepsza forma projektu takiego modelu pozostaje przedmiotem badań.

Otwartość ma aspekty zarówno techniczne, jak i komercyjne. Skupiamy się przede wszystkim na aspekcie technicznym, przybierającym kształt zapotrzebowania na dostępność dobrze zdefiniowanych interfejsów do zarządców zasobów. W systemach rozproszonych dostęp do interfejsów odbywa się za pomocą komunikacji międzyprocesowej. Interfejsy w systemie otwartym są publikowane i muszą być ogólnego przeznaczenia, aby uniknąć potrzeby powielania zasobów i zarządców zasobów dla różnych aplikacji.

Współbieżność przynosi korzyści w postaci większej wydajności. W systemach rozproszonych ogólnego przeznaczenia można korzystać ze współbieżności między procesami klientów i między procesami serwerów, natomiast wewnątrz jednego zastosowania współbieżności nie da się wykorzystać dopóty, dopóki program użytkowy nie zostanie skonstruowany w postaci systemu współbieżnych procesów.

Skalowalność stanowiła dominujące zagadnienie w projektowaniu systemów rozproszonych w ostatniej dekadzie i jest nim nadal. Zwielokrotnianie danych oraz dystrybucja obciążeń między serwerami są kluczowymi technikami jej wdrażania.

Tolerowanie uszkodzeń może być skuteczniej realizowane w systemach rozproszonych aniżeli w systemach o bardziej scentralizowanych architekturach. Aby zagwarantować, że w przypadku awarii jednego komputera ważne zadania zostaną przydzielone innemu komputerowi, stosuje się redundancję w sprzęcie. Niemniej jednak usuwanie bez utraty danych skutków awarii sprzętowych i programowych wymaga starannego projektowania.

Przezroczystość dotyczy spełniania oczekiwań użytkowników i programistów aplikacji, że zbiór połączonych siecią komputerów można będzie uważać za system zintegrowany, ukrywający rozproszoną naturę zasobów służących do wykonywania zadań użytkownika.

Obecny stan wiedzy na temat systemów rozproszonych wywodzi się z bogactwa prac badawczych i rozwojowych. W ostatnich dwudziestu latach wykonano wiele fascynujących i pasjonujących projektów badawczych.

5. Architektura systemu

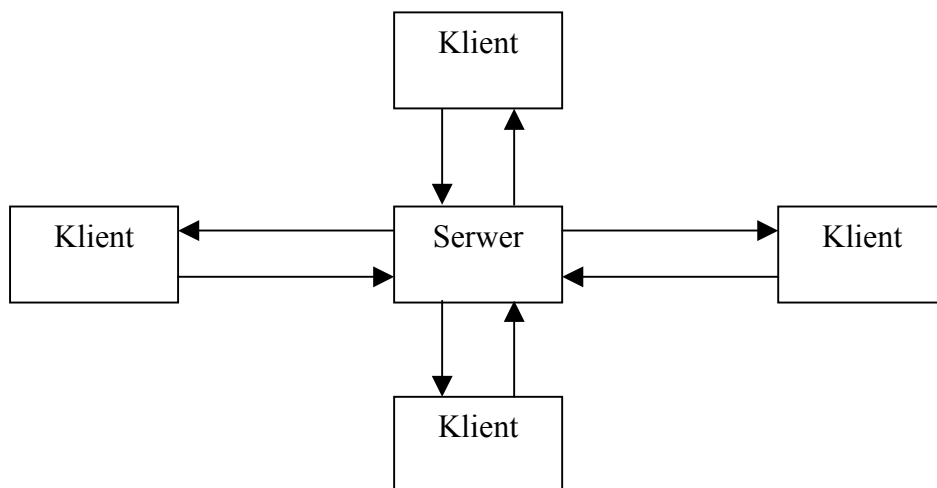
5.1. Ogólny schemat architektury systemu

5.1.1. Wstęp

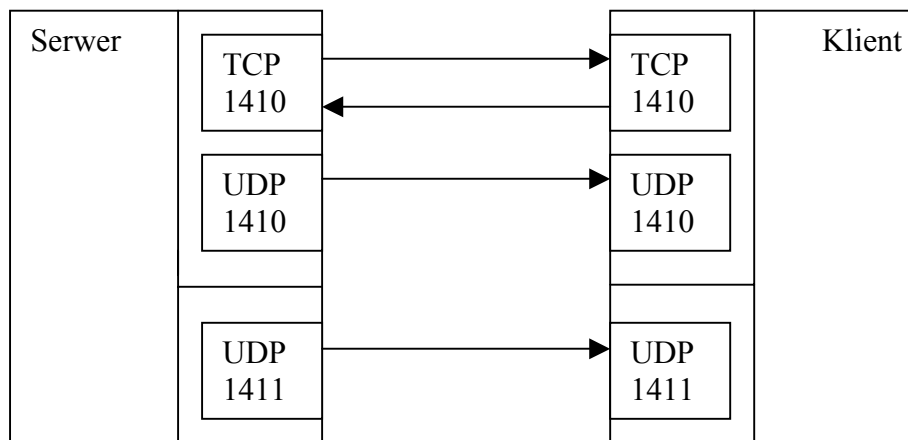
Aplikacja działa w architekturze klient-serwer. Budowa takiego systemu składa się z wydzielonego komputera głównego (serwera) oraz grupy komputerów klienckich połączonych w sieć.

System korzysta z protokołu TCP/IP w fazie etapu łączenia się klientów z serwerem, wymiany komunikatów sterujących i informujących o stanie pracy poszczególnych klientów a także do przesyłania wyników obliczeń pomiędzy klientami a serwerem.

Do rozgłaszania informacji dotyczących pobranych próbek z karty dźwiękowej użyto protokołu UDP. Wykorzystano w ten sposób transmisję w trybie Broadcast, przez co system stał się bardziej skalowalny ze względu na ilość komputerów. Informacja o próbkach jest przesyłana w sposób ciągły, a każda próbka transmitowana jest tylko jeden raz.



Rys. 5.1. Przykładowy schemat architektury klient-serwer



Rys. 5.2. Rodzaje komunikacji pomiędzy aplikacjami

5.1.2. Serwer – zarządca

Do głównych zadań komputera pełniącego rolę serwera należy:

- zarządzanie procesem logowania klientów, dystrybucją parametrów transmisji,
- decyzja o optymalnym i efektywnym podziale zadań na poszczególne komputery,
- pobieranie próbek sygnałów akustycznych z karty dźwiękowej,
- odbieranie i sygnalizowanie wszelkich niepoprawnych zachowań w tym wykrywanie i sprawdzanie poprawności działania sieci, wystarczającej ilości mocy obliczeniowej komputerów itp.,
- zbieranie wyników obliczeń z komputerów klienckich, formowanie kolejności próbek a następnie odtwarzanie wyniku obliczenia (dźwięk, postać binarna).

Ponieważ serwer zajmuję się pobieraniem próbek dźwięku, zatem konieczne jest, aby był wyposażony w kartę dźwiękową. W fazie projektu założono, iż serwer powinien współpracować z stereofoniczną kartą 16 bitową. Jako źródło dźwięku użyte jest wejście mikrofonu lub wejście liniowe, a dodatkowo umożliwiające odtwarzanie dźwięku z zasobów systemowych np. CD-ROM lub z aplikacji umożliwiającej odtwarzanie plików typu WAV.

5.1.3. Klient

Klientem może być dowolny komputer osobisty pracujący pod kontrolą systemu operacyjnego rodziny Microsoft Windows. Co prawda sam projekt nie wymaga stosowanego kryterium, ale przyjęto jako minimalną konfigurację komputer z procesorem Pentium 100. Z efektywnego punktu widzenia najkorzystniejszymi warunkami do pracy systemu jest jednorodna konfiguracja sprzętu komputerowego. Doskonałym przykładem jest np. uczelniana pracownia komputerowa, gdzie zwykle znajdują się komputery o podobnych konfiguracjach sprzętowych.

Komputer powinien być włączony w sieć lokalną za pomocą sieci Ethernet i posiadać zainstalowaną obsługę protokołu TCP. Aplikacja kliencka działa dopiero po zalogowaniu użytkownika do systemu, jednak nie wymaga żadnych specjalnych uprawnień.

Zadania klienta:

- nawiązanie połączenia z serwerem, etap inicjalizacji pracy aplikacji,
- odczyt próbek sygnałów,
- dokonanie obliczeń,
- wysyłanie wyników odpowiedzi do serwera.

W trakcie pracy nie zaleca się używania innych aplikacji absorbujących pracę procesora lub pamięci, ponieważ w ten sposób można zakłócić płynną pracę aplikacji i doprowadzić do zaburzenia całego procesu transmisji.

5.1.4. Budowa pakietu używanego do komunikacji poprzez sieć

W projekcie dla uproszczenia implementacji, a także dla zwiększenia niezawodności zdecydowano się na stosowanie jednego wzorca pakietu wymiany informacji. Wzorzec zawiera wszystkie informacje niezbędne do wymiany koniecznych informacji przyjmując usystematyzowany format przedstawiony na rysunku:

PID	0..65535	2B
CID	0..255	1B
Command	0..255	1B
Size	0..511	2B
Data		0..511B

Rys. 5.3. Schemat budowy wzorca pakietu używanego w systemie do komunikacji w sieci

Znaczenie pól w pakiecie:

PID (Packet Identifier) – Dwubajtowe pole z zakresu od 0 – 65535 zawiera informacje o liczbie porządkowej pakietu w trakcie transmisji. Pole to przyjmuje wartość 0 tylko raz w przebiegu całej transmisji i ma to miejsce na jej początku. Wewnętrzny licznik mechanizmu generującego tą wartość w momencie, gdy dojdzie do maksymalnego zakresu ustawiany jest na wartość 1. Przyjmując częstotliwość próbkowania = 44100 Hz, szerokość danych = 16 bitów oraz dwa kanały daje to szybkość strumienia wynoszącą 176400 bajtów / sekundę, co daje ok. 345 pakietów na sekundę, a zatem częstotliwość powtórzenia się tego samego numeru PID zdarza się co około 190 sekund. Jest to wartość na tyle duża, że nie dopuszczalne jest, aby doszło do takiego opóźnienia. Zatem nie ma żadnych obaw, iż pakiet może zostać niejednoznacznie zinterpretowany.

CID (Client Identifier) – Bajtowe pole określające identyfikator klienta na liście klientów znajdującej się na serwerze. Zwykle jest to ostatni oktet adresu IP komputera klienta. Wartość 0 – jest zarezerwowana w momencie, gdy serwer rozgłasza komunikat w trybie Broadcast, wówczas stosowanie identyfikatora jest niecelowe. Najczęstszym zastosowaniem tego pola jest wysyłanie informacji o kliencie lub też wyników obliczeń, aby serwer mógł jednoznacznie interpretować dane, od kogo pochodzą. Serwer jest w stanie zinterpretować dane bez potrzeby odczytu tego pola jednakże najczęściej dane te są buforowane podczas odczytu.

Zatem do bufora i tak musiałaby trafić dodatkowa informacja o nadawcy. Klient na podstawie tego pola jest w stanie zinterpretować odbiorcę pakietu.

Command – Pole komendy służy do interpretacji typu komunikatu. Zarówno klient jak i serwer po otrzymaniu pakietu na podstawie tego pola decydują, co należy zrobić z tym pakietem. Pakiet może być zadaniem, potwierdzeniem lub nieść ze sobą dane lub wyniki obliczeń.

Numer komendy	Opis komendy	Kierunek	Protokół
5	Rozsyłanie adresu IP serwera w trybie Broadcast, aby umożliwić klientom identyfikację adresu serwera	S → K	UDP
7	Wysłanie żądania rozłączenia klienta	S → K	TCP
11	Rozsyłanie próbek dla kanału lewego. W przypadku sygnału monofonicznego próbki są wysyłane z tą właśnie komendą	S → K	UDP
13	Rozsyłanie próbek dla kanału prawego	S → K	UDP
14	Wysyłanie wyniku obliczeń od klienta	S ← K	TCP
27	Wysyłanie informacji o typie transmisji	S → K	TCP
28	Informacje od klienta: moc obliczeniowa, rodzaj systemu operacyjnego, zalogowany użytkownik	S ← K	TCP
30	Żądanie wysłania współczynników	S ← K	TCP
31	Wysyłanie współczynników	S → K	TCP
32	Potwierdzenie przyjęcia współczynników	S ← K	TCP
34	Żądanie ponownego przeliczenia zadań	S ← K	TCP
35	Otrzymano zadania. Klient gotowy do obliczeń	S → K	TCP

Rys. 5.4. Lista komend stosowanych w systemie

Size – pole określające ile elementów w polu DATA jest ważnych, jeśli wartość wynosi zero oznacza to, że wszystkie elementy w tym polu są istotne i należy je odczytać. Gdy wartość tego pola zawiera się w przedziale (0, 511) oznacza to, że tylko tyle pierwszych elementów należy brać pod uwagę jako ważne dane, resztę należy uznać

za dane nieważne. W ten sposób można prawidłowo zinterpretować ostatnie pakiety w ciągu transmisji (współczynniki, próbki, odpowiedzi) .

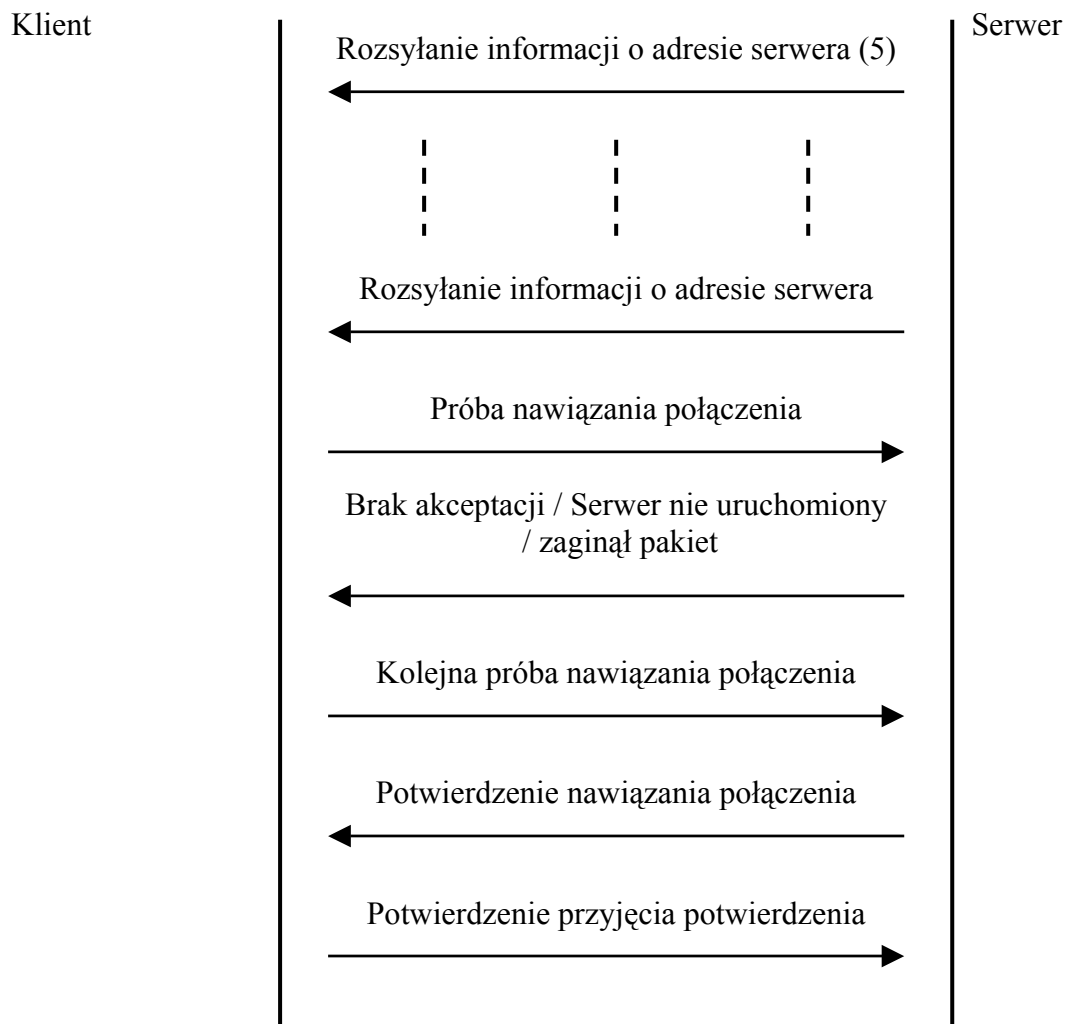
Data – pole niosące informacje w postaci próbek sygnałów, wyników obliczeń, parametrów lub innych informacji liczbowych. Jest to tablica 512 elementów typu całkowitego ze znakiem o szerokości 16 bitów zawierającego wartości z zakresu $\langle -32767 \dots 32768 \rangle$.

Dane przekraczające wartości $\langle -32767 \dots 32768 \rangle$ są dzielone na dwie części LSB i MSB. W części LSB znajduje się reszta z dzielenia, czyli wartość modulo 32767, z kolei w MSB znajduje się wynik dzielenia wartość / 32767

Zatem $Wartosc = MSB * 32767 + LSB$

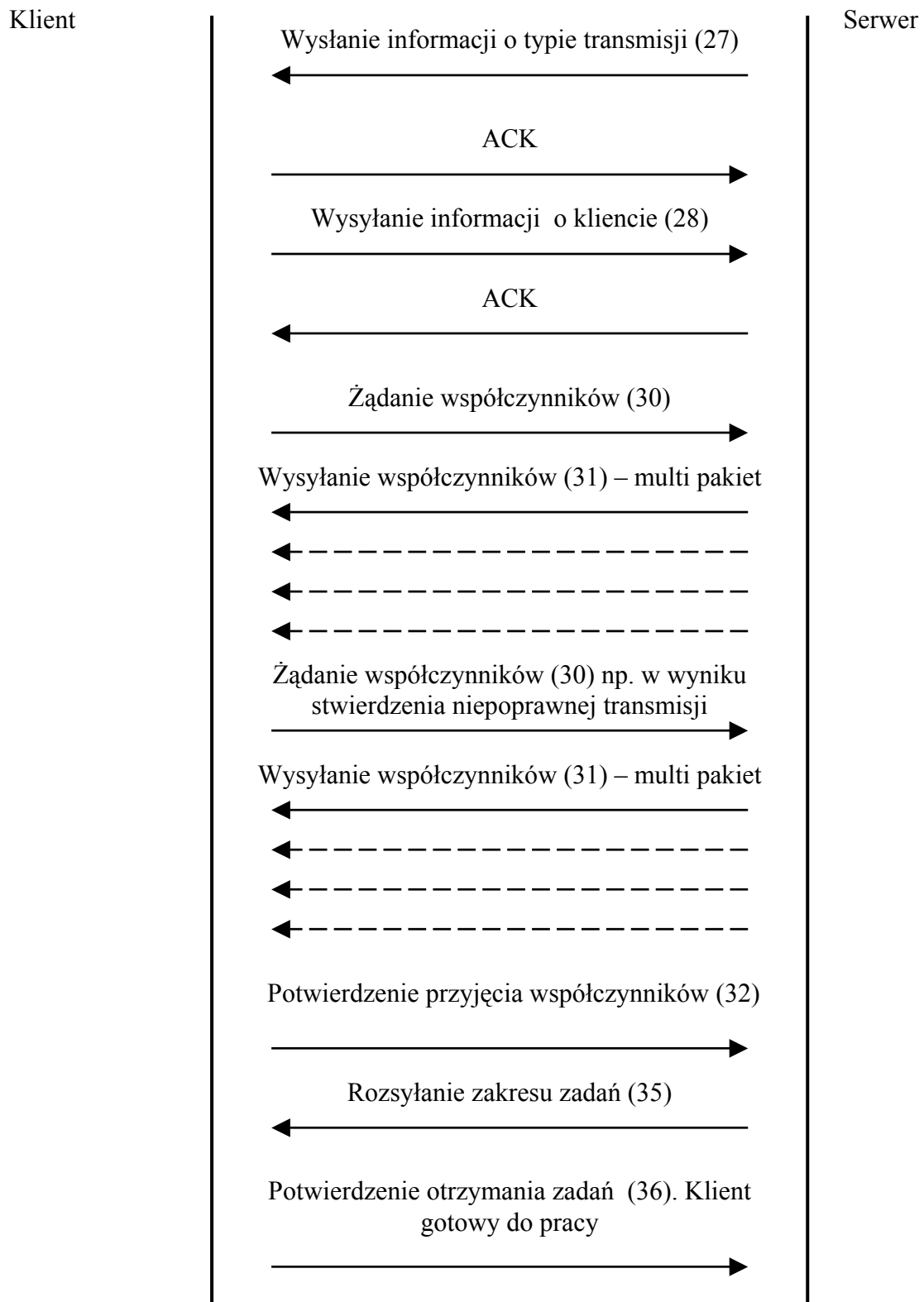
Numery komend zostały tak podzielone, iż komendy o numerach nieparzystych oznaczają komunikacje od serwera do klienta, natomiast numery parzyste oznaczają komunikacje w drugą stronę.

5.1.5. Schemat nawiązania połączenia



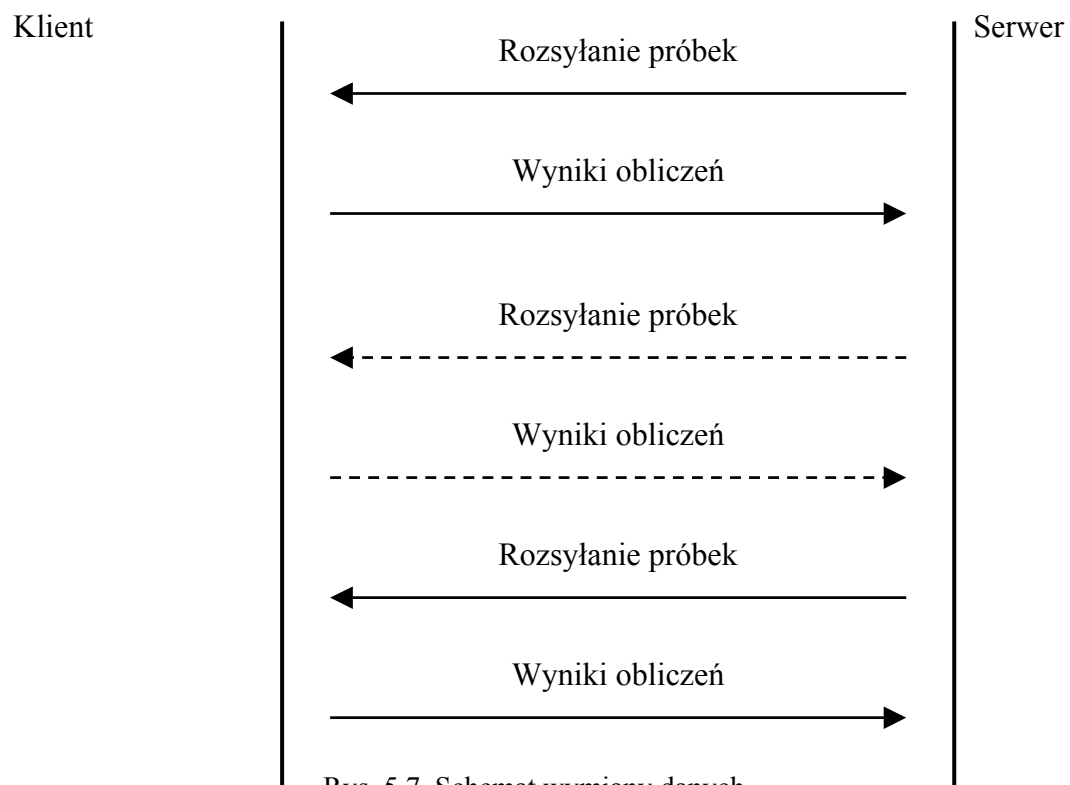
Rys. 5.5. Schemat nawiązania połączenia

5.1.6. Schemat przygotowania do pracy



Rys. 5.6. Schemat przygotowania do pracy

5.1.7. Schemat wymiany danych



5.2. Opis pakietów stosowanych w transmisji

5.2.1. Rozsyłanie informacji o serwerze w trybie Broadcast

Z chwila uruchomienia serwera, aplikacja wysyła w trybie Broadcast na standardowy port o numerze 1410 informacje, w której treści mieści się adres IP serwera. Mechanizm gniazdek dopuszcza używanie na jednym porcie dwóch komunikacji w różnych protokołach. Zatem nie ma żadnych komplikacji związanych z pracą gniazdka.

Czynność ta wykonywana jest aż do chwili stwierdzenia wystarczającej mocy obliczeniowej. Funkcja ta jest pomocna, jeżeli serwer zmienia swoją lokalizację lub, jeśli tworzy się nowe stanowisko klienckie. Oczywiście w aplikacji klienta istnieje ręczna modyfikacja tego parametru.

PID	0
CID	0
Command	5
Size	4
Data [0]	Pierwszy oktet adresu IP
Data [1]	Drugi oktet adresu IP
Data [2]	Trzeci oktet adresu IP
Data [3]	Czwarty oktet adresu IP

Rys. 5.8. Schemat pakietu dla komendy nr 5 - Rozsyłanie adresu IP

5.2.2. Proces nawiązania połączenia

Z chwilą kiedy klient dysponuje adresem i portem serwera, klient próbuje stworzyć połączenie sieciowe. Jeżeli czynność ta się nie powiedzie (awaria sieci, aplikacja serwera nie uruchomiona) wewnętrzny zegar z określoną częstotliwością będzie próbował nawiązać połączenia aż do skutku. W momencie, kiedy uda się nawiązać połączenie, przebieg wysyłania komunikatów protokołem TCP jest identyczny z przedstawionymi na rys 3.5, czyli przebiega trójetapowo (three way handshake).

Serwer w tym momencie kreuje wątek na potrzeby obsługi związanej z tym właśnie klientem. Dodatkowo tworzy element na liście klientów, na której będzie przechowywał informacje związane z klientem dopóki dany klient nie zostanie rozłączony.

5.2.3. Wysłanie parametrów transmisji

Po dokonaniu połączenia klienta z serwerem, serwer wysyła (warunkiem jest uprzednia konfiguracja parametrów, w przeciwnym razie serwer wyśle parametry domyślne) informacje związane z transmisją. Najistotniejszą z nich jest informacja czy próbki sygnałów akustycznych są stereofoniczne bądź monofoniczne, gdyż od tego zależy sposób traktowania przychodzących pakietów.

Kolejną informacją jest przydzielony identyfikator CID (zatem klient nie zna jeszcze swojego numeru). Z reguły numer CID jest wartością ostatniego oktetu numeru IP, jednak dopiero otrzymanie takiego pakietu jest dla klienta informacją, że właśnie on jest posiadaczem takiego identyfikatora.

PID	0
CID	CID
Command	27
Size	5
Data [0]	1 = mono / 2 = stereo
Data [1]	LSB częstotliwości próbkowania
Data [2]	MSB częstotliwości próbkowania
Data [3]	Ilość bitów / próbkę - (8 lub 16)
Data [4]	User Identifier

Rys. 5.9. Schemat pakietu dla komendy nr 27 – Typ transmisji

5.2.4. Wysłanie informacji od klienta

Z chwila, gdy wystartuje aplikacja klienta, klient rozpoczyna procedurę liczącą swoją wydajność obliczeniową. Procedura wykonuje 100 transformat FFT o długości 16384 punktów. Liczenie podzielone jest na kilka etapów, pomiędzy którymi występuje przerwa, w ten sposób zwiększona jest dokładność pomiaru, a sama operacja jest wykonana na osobnym wątku posiadającym najwyższy priorytet (TimeCritical). Po wykonaniu obliczeń ustawiony jest znacznik – „mam policzony test wydajnościowy” i jeżeli klient w międzyczasie uzyskał połączenie z serwerem zostaje wysłany pakiet z informacjami o kliencie.

PID	0
CID	0
Command	28
Size	21
Data [0]	LSB Test wydajności
Data [1]	MSB Test wydajności
Data [2]	Identyfikator systemu operacyjnego
Data [3..15]	Nazwa użytkownika
Data [16]	LSB wersji pliku
Data [17]	MSB wersji pliku
Data [18]	LSB wersji produktu
Data [19]	MSB wersji produktu
Data [20]	Brak współczynników = 0 Mam współczynniki =1

Rys. 5.10. Schemat pakietu dla komendy nr 28 – Typ klienta

Pola 16..19 zawierają numer wersji aplikacji klienta, co ma swoje uzasadnienie w sytuacji dalszego rozwoju projektu. W chwili, gdy dokonano by zmian w algorytmach obliczeniowych klienta, klient posiadający nieaktualną wersję aplikacji mógłby wprowadzać błędne wyniki obliczeń. W ten sposób można łatwo wykryć klienta dysponującego nieaktualną wersją oprogramowania.

Pole 20 – informuje z kolei czy klient posiada odebrane współczynniki od serwera, informacja taka jest przydatna, gdy aplikacja klienta nie została zamknięta, a jedynie klient np. rozłączył się i ponownie zalogował wówczas dosyłanie mu współczynników jest zbędne

5.2.5. Żądanie współczynników

Po nawiązaniu połączenia z serwerem równolegle przeprowadza się test wydajnościowy mocy obliczeniowej komputera. Następnie klient wysyła do serwera żądanie przysłania współczynników. Co prawda klient teoretycznie nie będzie nigdy wykorzystywał wszystkich elementów otrzymanego zbioru a tylko jego część, jednak w tej chwili klient ani serwer dokładnie jeszcze nie wiedzą, jaką część pracy zostanie przydzielona temu klientowi. Ponadto posiadanie całego zbioru współczynników jest bardzo elastycznym rozwiązaniem np. na wypadek rekonfiguracji zmianie ulegną tylko dwa parametry: indeks początkowy i indeks końcowy, natomiast współczynniki same w

sobie nie ulegną zmianie. Wewnętrzny zegar będzie tak długo inicjował żądanie współczynników aż je otrzyma.

PID	0
CID	0
Command	30
Size	0
Data	Puste

Rys. 5.11. Schemat pakietu dla komendy nr 30 – żądanie współczynników

5.2.6. Wysyłanie współczynników

Wysyłanie współczynników może być wywołane na dwa sposoby: pierwszym jest żądanie od klienta poprzez komendę omówioną w punkcie 5.6, drugim jest zmiana współczynników w trakcie pracy serwera i zalogowanych już klientów. W tym drugim przypadku informacje o nowych współczynnikach otrzymują wszyscy zalogowani klienci.

Transmisja współczynników splotu odbywa się w wielu pakietach zależnie od długości sumarycznej odpowiedzi impulsowej filtra.

Pakiet ma standardową budowę. W pierwszym pakiecie zawarta jest informacja o sumarycznej długości współczynników, które serwer będzie przysyłał do klientów.

Transmisja wykonywana jest w trybie Unicast (protokół TCP), zatem serwer nie zajmuje się obsługą poprawności transmisji. Klient po odczytaniu informacji o długości wszystkich elementów odczytuje tak długo, aż rozpozna pakiet końcowy i jeżeli liczba wszystkich odczytanych współczynników jest równa zadeklarowanej w pierwszym pakiecie to wówczas klient odeśle potwierdzenie przyjęcia współczynników (czynność omówiona w punkcie (5.8).

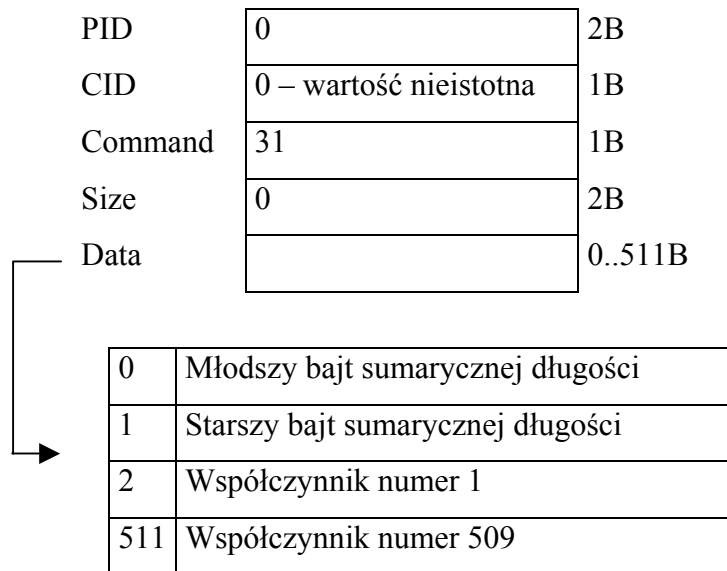
Faza rozsyłania współczynników jest bardziej skomplikowana od poprzednio omówionych. Wysyłanie danych odbywa się już tzw. Multipakietami, które są częścią skończonego ciągu danych, lecz o rozmiarze przekraczającym możliwości wysłania jednym pakietem.

Niech N – liczba porządkowa kolejności pakietów,

M – całkowita ilość pakietów w transmisji,

K – liczba porządkowa współczynnika w pakiecie w polu DATA,

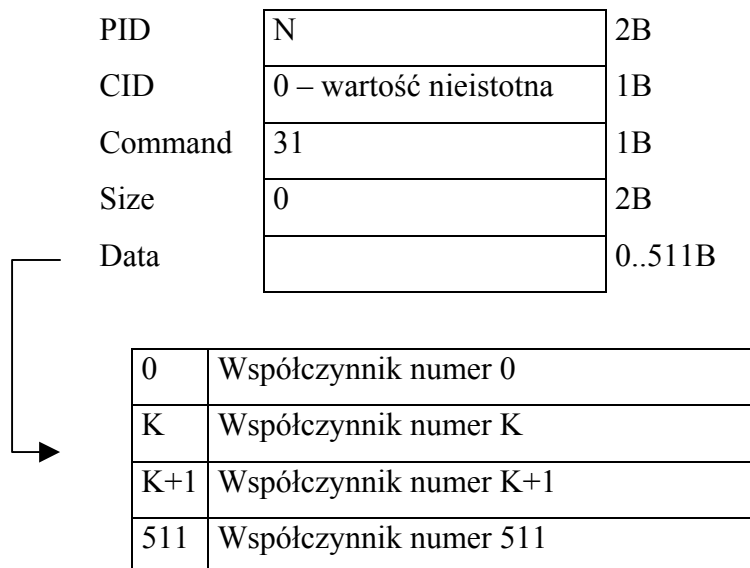
1. Postać pierwszego pakietu o $N = 0$:



$$Długość_{całkowita} = Data[0] + Data[1] * 2^{15}$$

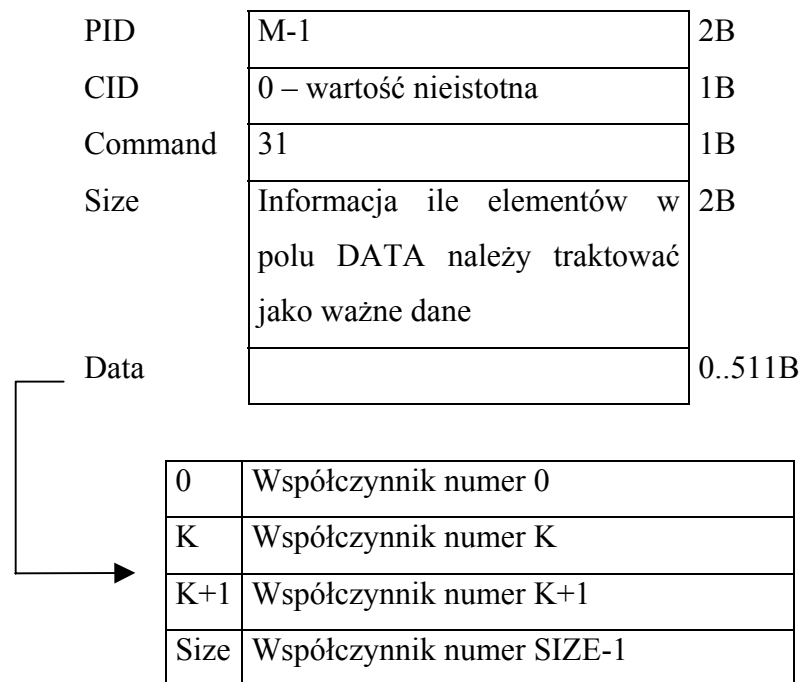
Rys. 5.12. Transmisja współczynników – budowa pierwszego pakietu

2. Postać pakietu zawierającego się w przedziale $N > 0$ i $N < M$:



Rys. 5.13. Transmisja współczynników – budowa pośredniego pakietu

3. Postać pakietu kończącego transmisję:



Rys. 5.14. Transmisja współczynników – budowa ostatniego pakietu

5.2.7. Potwierdzenie przyjęcia współczynników

Jeśli klient wykryje, że odebrany pakiet zawierający współczynniki jest ostatnim, a także że liczba załadowanych elementów do wewnętrznej tablicy jest zgodna z zadeklarowaną długością, odsyła pakiet będący potwierdzeniem i informacją, że klient posiada ważne współczynniki.

Może się jednak wydarzyć, że w trakcie transmisji uległy uszkodzeniu lub zaginięciu pewne pakiety. Wówczas klient ponownie wysyła żądanie współczynników, a serwer z kolei analogicznie jak to opisano powyżej wyśle całą partię danych. Klient znowu dokona analizy stanu odebranych danych i jeżeli stwierdzi poprawność transmisji odeśle potwierdzenie:

PID	0
CID	0
Command	32
Size	0
Data	Puste

Rys. 5.15. Schemat pakietu dla komendy nr 32 – potwierdzenie przyjęcia współczynników

5.2.8. Żądanie podziału zadań

W normalnie pracującym systemie do nadzorowania pracy całego przebiegu jest zobowiązany tylko i wyłącznie serwer. Komenda ta jest jednak wprowadzona i zaimplementowana po stronie serwera z myślą o rozwinięciu tej funkcji w późniejszych pracach nad systemem. W ten sposób tylko zasygnalizowano obecność takiej funkcjonalności po stronie serwera, po jej otrzymaniu serwer dokona ponownej analizy posiadanej mocy obliczeniowej wśród klientów (patrz punkt 5.10)

5.2.9. Podział zadań

W chwili, gdy system posiada wystarczającą ilość mocy obliczeniowej serwer roześle do każdego z klientów z osobną informację, jaką część pracy mu przydzielono. W rzeczywistości są to parametry określające, jaka część współczynnika odpowiedzi impulsowej będzie splatana z napływającymi danymi od serwera. Po otrzymaniu takiego pakietu klient oblicza minimalną długość FFT dla obliczenia widma takiego przedziału i dokonuje transformaty.

PID	0
CID	CID
Command	35
Size	4
Data [0]	LSB Indeks 0
Data [1]	MSB Indeks 0
Data [2]	LSB Indeks 1
Data [3]	MSB Indeks 1

Rys. 5.16. Schemat pakietu dla komendy nr 35 – przyznanie podziału

5.2.10. Potwierdzenie przyjęcia podziału zadań

Jeżeli klient otrzymał poprawny pakiet z informacją o podziale zadań i wykonał konieczne czynności z pozytywnym rezultatem odsyła do serwera informacje o ustalonym parametrze długości FFT, a także informacje o zakresie z jakiej części będzie korzystał (informacja redundantna jednak wprowadzona dla kontroli).

PID	0
CID	CID
Command	36
Size	6
Data [0]	LSB Długości FFT
Data [1]	MSB Długości FFT
Data [2]	LSB Indeks 0
Data [3]	MSB Indeks 0
Data [4]	LSB Indeks 1
Data [5]	MSB Indeks 1

Rys. 5.17. Schemat pakietu dla komendy nr 36 – potwierdzenie przyznania podziału

5.2.11. Rozsyłanie próbek

Próbki rozsyłane są w trybie Broadcast, ponieważ do każdego klienta trafiają takie same dane a także aby zminimalizować obciążenie sieci.

Komenda ta stanowi kluczową rolę w całym procesie transmisji. Wysyłanie danych wykonywane jest w trybie Broadcast w postaci multipakietów (mechanizm multipakietu został omówiony w punkcie 5.6 – wysyłanie współczynników). Zasada działania nie różni się niczym od wysyłania współczynników. Należy tylko nadmienić, że komenda ta ma dwa warianty w zależności od parametrów próbkowanego sygnału, a mianowicie dla sygnału monofonicznego istnieje tylko komenda o numerze 11 dla sygnału stereofonicznego istnieją dwie równoległe komendy o numerach 11 i 13. To już do serwera należy rozdzielać próbki na składowe lewą i prawą.

PID	N
CID	CID
Command	11 / 13
Size	0 lub M
Data	Dane

Rys. 5.18. Schemat pakietu dla komendy nr 11/13 – rozsyłanie próbek

5.2.12. Wysyłanie wyników obliczeń

Działanie komendy jest bardzo podobne do komend omówionych w punkcie 5.12 z tą różnicą, że transmisja odbywa się już w trybie Unicast (każdy z pakietów zawiera niepowtarzalną i unikatową treść informacji)

PID	N
CID	CID
Command	12 / 14
Size	0 lub M
Data	Dane

Rys. 5.19. Schemat pakietu dla komendy nr 12/14 – wyniki obliczeń

5.2.13. Kontrola aktywności klientów

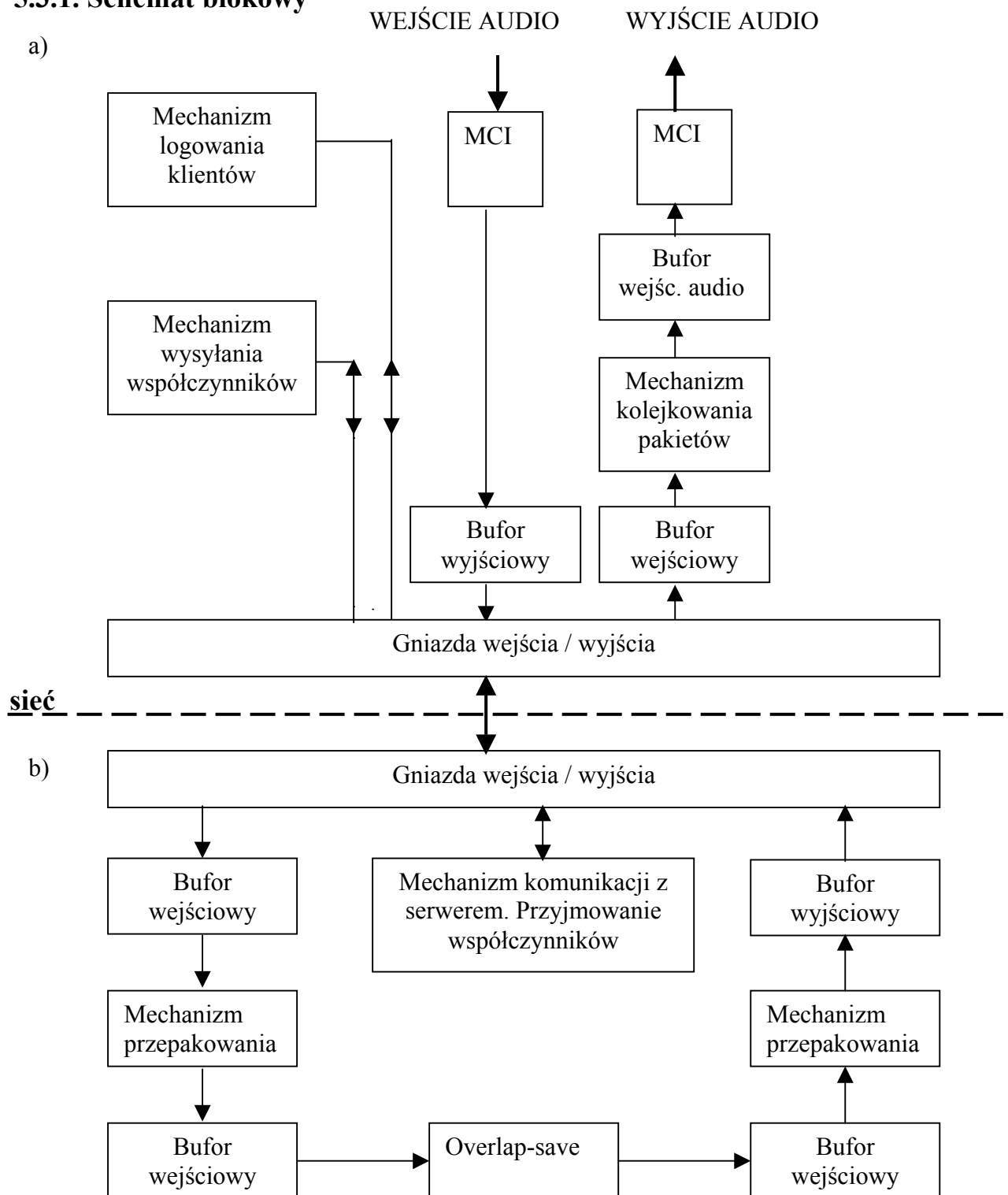
Działanie komendy przypomina działanie systemowego polecenia ping. Z tą różnicą, że odpowiedź od klienta oprócz obecności hosta w sieci, oznacza także to, że po stronie klienta działa poprawnie aplikacja.

PID	N
CID	CID
Command	99/100
Size	0
Data	Puste

Rys. 5.20. Schemat pakietu dla komendy nr 99/100 – „ping-pong”

5.3. Zasada działania

5.3.1. Schemat blokowy



Rys. 5.21. Schemat blokowy systemu przetwarzania. a) schemat serwera, b) schemat klienta

5.3.1. Mechanizm wielowątkowy

Zarówno aplikacja serwera jak i aplikacja klienta są aplikacjami wielowątkowymi. Wątek (ang. thread) jest obiektem systemu operacyjnego, reprezentującym wydzieloną część kodu w ramach procesu. Mechanizm wielowątkowy pozwala na niezależną i jednoczesną realizację wielu różnych funkcji aplikacji. W aplikacji serwera dla każdego zalogowanego klienta jest przydzielany nowy wątek na potrzeby indywidualnej z nim komunikacji. Dla operacji wysyłania danych przez gniazda sieciowe, dokonywanie przeliczeń i innych newralgicznych operacji także są kreowane nowe wątki. W aplikacji klienta wątkiem o najwyższym priorytecie (tylko wątek o wyższym priorytecie, a zatem ważniejszy jest w stanie go wywłaszczyć) jest wątek dokonujący obliczenia splotu metodą Overlap Save .

5.3.2. Bufory

W aplikacjach zastosowano bufory będące rejestrami FIFO, przechowującymi próbki, komunikaty lub odpowiedzi z wynikami. W zależności od umiejscowienia takiego bufora, bufor może spełniać różne rolę:

W przypadku serwera:

- buforować próbki bezpośrednio napływające z karty dźwiękowej, uruchomiony wątek cyklicznie sprawdza stan tego bufora i jeżeli stwierdzi obecność elementów będzie starał się go w całości wysłać poprzez gniazdo,
- buforować odpowiedzi od klientów z wynikami obliczeń tym razem nie w kolejce FIFO, lecz na liście o dostępie swobodnym. Osobny wątek jest odpowiedzialny za regularne przeglądanie listy i umieszczanie kolejnych pakietów znalezionych według algorytmów wyszukujących kolejne porządkowe pakiety. Pakiet jest umieszczany w kolejnym buforze,
- bufor, w którym umieszczone są elementy pobierane z listy omówionej w poprzednim punkcie ten bufor czyta już tylko zdarzenie mechanizmu MCI, w którego rezultacie przetworzone próbki są wysyłane do np. urządzenia wyjściowego audio.

W przypadku klienta

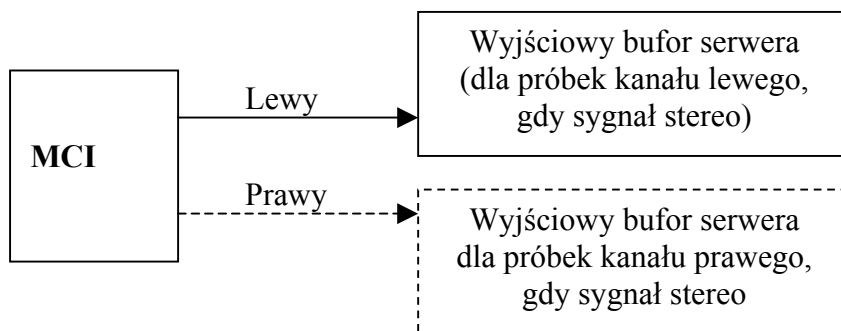
- buforowanie próbek, odebranych przez gniazdko UDP,
- buforowanie próbek przepakowanych do postaci spełniających założenia metody Overlap (przepakowanie wzwyż). I tu osobny wątek pełni rolę nadzorcy tego procesu,
- buforowanie wyników po wykonaniu obliczeń splotu,
- buforowanie próbek po wykonaniu przepakowania w dół. Ten bufor będzie kontrolował cyklicznie wątek i jeżeli stwierdzi obecność elementów wyśle je poprzez gniazdko.

Należy pamiętać o jednej wadze tego rozwiązania mianowicie sumowania się opóźnień wynikających z wędrówki danych poprzez wiele buforów. Zaletą jednak jest niepodważalna skalowalność takiego rozwiązania i niezawodność.

5.3.3. Pobieranie próbek dźwiękowych

Po wcześniejszej konfiguracji parametrów pobieranego dźwięku można przystąpić do procesu próbkowania sygnału akustycznego, korzystających z MCI (ang. *Multimedia Communication Interface*), będącej interfejsem systemu Windows.

Dźwięk jest buforowany w wewnętrznym buforze i po jego wypełnieniu jest inicjowane zdarzenie w czasie, którego zawartość bufora jest przepisywana do wyjściowego bufora serwera. Należy jednak zauważyć, że interfejs MCI jest charakterystycznym elementem jedynie systemu Windows.



Rys. 5.22. Schemat załadowania danych do bufora wyjściowego

5.3.4. Gniazda sieciowe TCP i UDP

Podstawowym obiektem używanym przez aplikacje związane z komunikacją sieciową jest gniazdo (ang. *socket*). Gniazda są zaprojektowane na podobnych zasadach, na jakich aplikacje zapisują i odczytują pliki. Aby utworzyć plik należy znać nazwę pliku i jego lokalizację, w przypadku gniazda do jego otwarcia konieczne są dwie informacje: komputer (adres IP) oraz numer portu komputera, na której pracuje aplikacja, z którą zamierza się nawiązać połączenie. W projekcie wykorzystuje się dwa numery portów w tym jeden pełniący dualną rolę: jest gniazdem zarówno dla komunikacji według protokołu TCP oraz UDP. Schemat takiego połączenia jest zamieszczony na rys 5.2

Podział funkcji w zależności od portu i jego typu:

- Port 1410 TCP – odpowiada za nawiązywanie połączenia, transmisję informacji inicjalizujących i pozostałą komunikację obsługującą żądania lub potwierdzenia. Port spełnia funkcję sterującą pracą całego systemu. Port ponadto odbiera odpowiedzi od klientów z wynikami obliczeń.
- Port 1410 UDP – wysyła pakiet „echo” informujący o tym, kto pełni rolę serwera w sieci, rozgłaszając w ten sposób numer IP serwera.
- Port 1411 UDP – Port służący wyłącznie do rozsyłania próbek w trybie Broadcast.

Proces nawiązywania i kończenia połączenia został opisany w punktach powyżej. W praktyce pierwszą czynnością, jaką powinien wykonać serwer jest otwarcie swojego gniazda TCP i przejście w stan nasłuchu (ang. Listen). I teraz dopiero, po otwarciu gniazda przez klientów są oni w stanie nawiązać połączenie z serwerem na tym porcie. Gniazda UDP nie wymagają podobnej operacji i tylko od programowej strony muszą zostać uaktywnione. Kończenie połączenia może być inicjowane dwustronnie. Serwer może zakończyć działanie procesu przetwarzania i wylogować wszystkich klientów, może też zakończyć samo działanie aplikacji. Klient też może samoistnie wyrazić chęć do zakończenia połączenia.

Sygnał „echo” jest wykorzystywany w chwili startu aplikacji klienta, jeżeli okaże się że pod ostatnio „pamiętanym” adresem nie działa aplikacja zarządcy i próba nawiązania połączenia kończy się niepowodzeniem, klient wówczas odczytuje sygnał „echo” rozgłaszane w sieci przez zarządcę. Cały system jest zaprojektowany w ten sposób, iż klient jest w stanie rozpoznać wyłączenie aplikacji zarządcy i przeniesienie jej na inny

komputer, klient potrafi na podstawie sygnału echo rozpoznać który komputer sieci pełni aktualnie rolę zarządcy. Istnieje dodatkowo opcja auto-connect, która próbuje okresowo przywrócić połączenie w przypadku rozłączenia.

5.3.5. Mechanizm kolejkowania pakietów

W punkcie 7.9 opisany zostanie algorytm wyszukujący kolejne pakiety w liście bufora wejściowego serwera. W tym punkcie zostanie opisany mechanizm „sklejania” składowych części wyników w jeden pakiet.

Zasadę działania zobrazuje przykład:

Niech $h(n)$ reprezentuje odpowiedź impulsową filtra i niech jego postać będzie następująca: $h(n)=[1\ 2\ 3\ 4\ 5\ 6\ 7\ 8]$, $H(n)$ określa jego widmo.

Założmy, że zadanie liczenia splotu będzie rozdzielone na dwa komputery, działanie dla większej ilości jest takie samo. I niech podział będzie wynosić 3:5.

Zatem komputer pierwszy (K1) będzie liczył splot tylko dla 3 elementów $h(n)$, natomiast komputer drugi (K2) będzie liczył część pozostałą, czyli 5 elementów $h(n)$, parametry metody Overlap przyjmują zatem następujące wartości $M_1=3$, $M_2=5$, $N=8$;

Zakładamy w celu prostej analizy, iż sygnał filtrowany $x(n)$ jest sygnałem skończonym o 16 elementach: $x(n)=[1\ 2\ \dots\ 16]$.

Tak jak to zostało opisane w metodzie Overlap-save w pierwszej kolejności należy uzupełnić składowe $h_1(n)$ i $h_2(n)$ koniecznymi zerami. Zatem:

$$h_1(n) = [1\ 2\ 3\ 0\ 0\ 0\ 0\ 0] \text{ i } h_2(n) = [4\ 5\ 6\ 7\ 8\ 0\ 0\ 0]$$

Dla tak przygotowanych sygnałów można obliczyć widmo $H_1(n)$ i $H_2(n)$

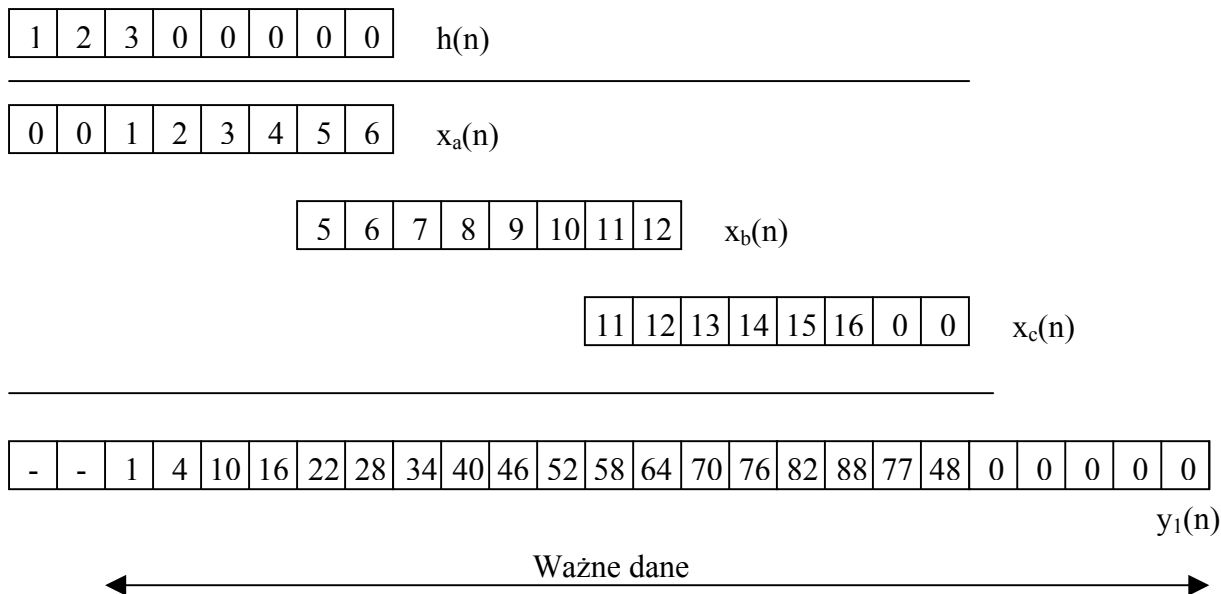
Niech sygnały $x(n) = x_1(n) = x_2(n)$. Sygnały x_1 , x_2 wypełniamy „od przodu” zerami w ilości równej lokalnemu $M-1$.

$$x_1(n) = [0\ 0\ 1\ 2\ 3\ \dots\ 16]$$

$$x_2(n) = [0\ 0\ 0\ 0\ 1\ 2\ 3\ \dots\ 16]$$

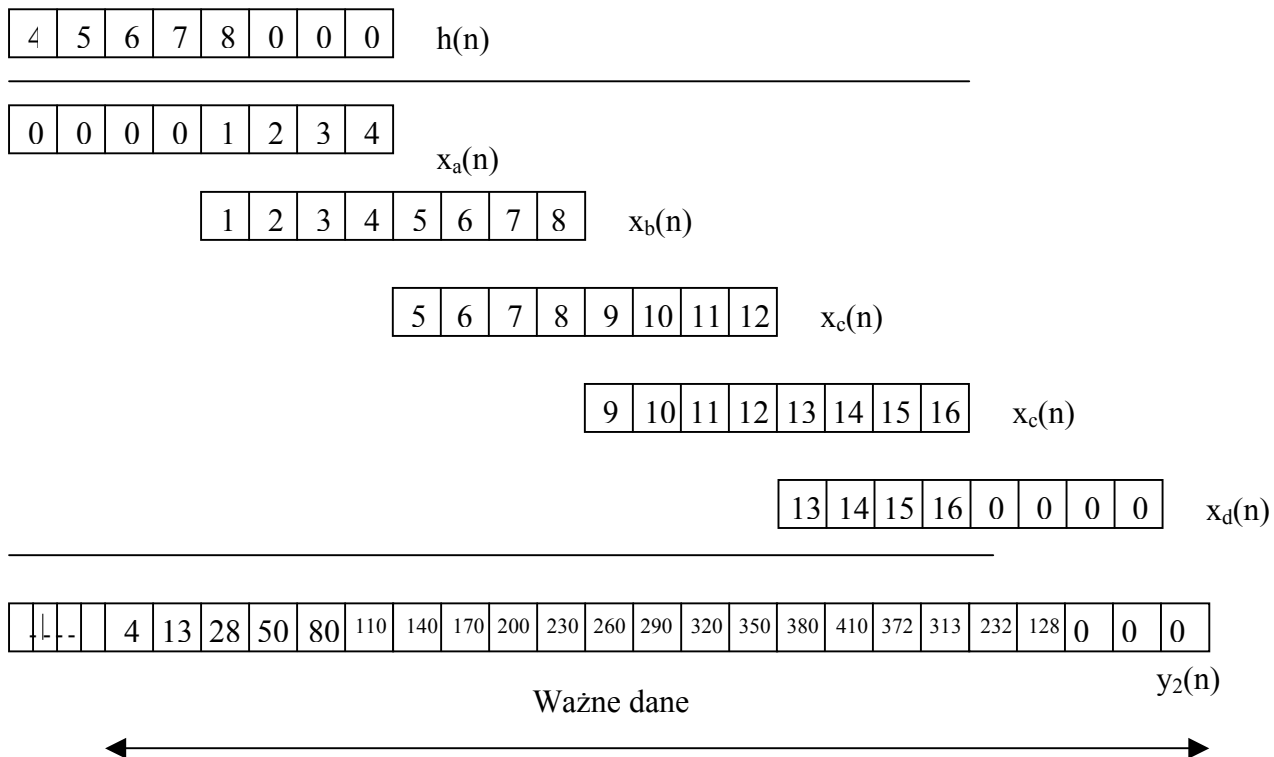
Z tak przygotowanymi danymi możemy rozpocząć cykl obliczeń:

Komputer 1:



Rys. 5.23. Proces liczenia splotu dla pierwszego komputera

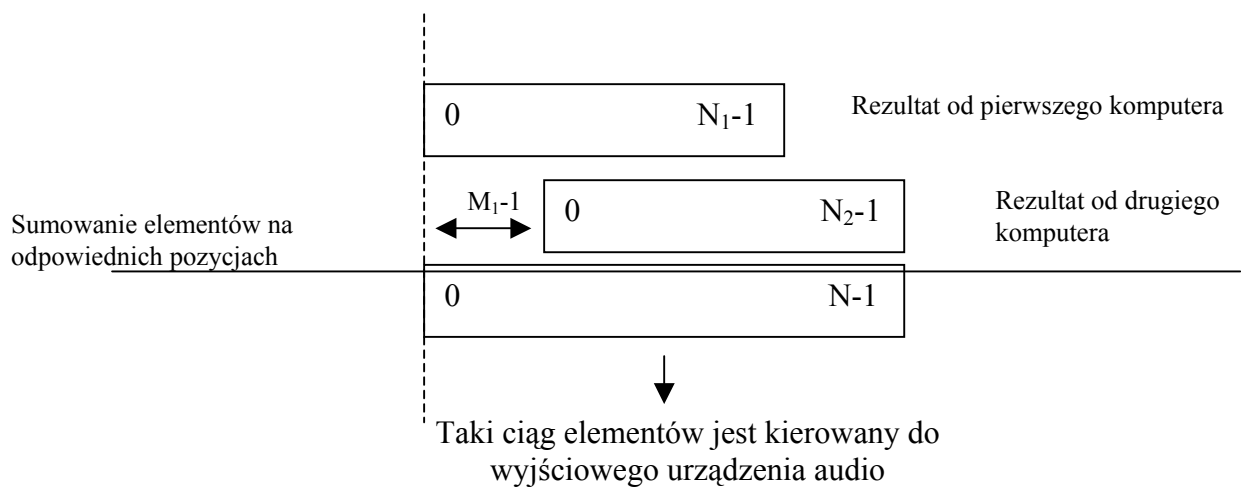
Komputer 2:



Rys. 5.24. Proces liczenia splotu dla pierwszego komputera

Wyniki obliczeń zostają wysłane za pośrednictwem gniazd do serwera. Serwer dysponując informacjami takimi jak identyfikator pakietu i klienta, od którego odebrał taki pakiet jest w stanie wyznaczyć kolejność składania pakietów, serwer także dysponuje informacją o tym jak długi parametr N – występuje u klienta, a także jaka jest wartość parametru M . Obliczenia mogłyby przebiegać przy stałej wartości parametru N , jednak w przypadku zróżnicowania mocy obliczeniowych spowodowałoby to małą elastyczność systemu.

Składanie pakietu wygląda następująco:



Rys. 5.25. Proces składania wyników

5.3.6. Mechanizm podziału pracy dla klientów

Mechanizm działa dwuetapowo. Ciągłe w czasie pracy (tuż po zalogowaniu nowego klienta) jest sprawdzane czy serwer nie dysponuje już wystarczającą mocą obliczeniową. W momencie, gdy taki moment nastąpi dokonywany jest podział zadań dla komputerów klientów.

Serwer dysponuje informacjami o testach wydajnościowych poszczególnych klientów. Informację tę stanowi czas, w jakim klient wykonał operację 16384 punktową FFT. Serwer dysponuje następującymi danymi odnośnie transmisji:

- f – częstotliwość próbkowania (czyli ilość próbek na sekundę),
- Ch – ilość kanałów (stereo/mono),
- N – długość FFT po stronie klienta,
- M – długość części współczynnika jaki klient dostał do obliczeń.

Na podstawie tych danych możemy stwierdzić ile transformat FFT należy obliczyć na sekundę:

$$W = 2 \cdot Ch \cdot \frac{f}{N - M + 1} \quad (1)$$

Wiedząc zatem ile czasu potrzebne jest klientowi na obliczenie jednej FFT (16384 punktowej) możemy wyznaczyć wzór na maksymalną wartość parametru M jaki dany klient jest w stanie przyjąć do obliczeń:

$$M = N + 1 - \frac{2 \cdot Ch \cdot f}{W} \quad (2)$$

Powyższe rozważania są prawdziwe o ile przyjmiemy, iż obliczenia odbywają się dla $N=16384$. Jeżeli przyjmiemy zmienną długość transformaty wówczas powyższe wzory zostaną następująco zmodyfikowane:

$$W = 2 \cdot Ch \cdot \frac{f}{N - M + 1} \cdot \frac{N \cdot \log_2(N)}{16384 \cdot \log_2(16384)} \quad (3)$$

$$M = \left(N + 1 - \frac{2 \cdot Ch \cdot f}{W} \right) \cdot \frac{16384 \cdot \log_2(16384)}{N \cdot \log_2(N)} \quad (4)$$

Serwer oczekując na kolejnych klientów nie robi nic innego jak wylicza za każdym razem sumę składowych według wzoru (2) lub (4). Jeżeli suma składowych osiągnie wartość przewyższającą długość odpowiedzi impulsowej filtru (należy dodać także pewien zapas mocy), wówczas dopiero serwer udostępnia możliwość przeprowadzenia operacji. W innym przypadku serwer zakłada, iż proces nie powiodłby się pozytywnie w wyniku niewystarczającej mocy obliczeniowej.

Gdy już zajdzie przypadek, w którym serwer stwierdzi gotowość do przeprowadzenia transmisji należy dokonać podziału zadań. Proces ten przebiega następująco:

W pierwszej kolejności wybierany jest najdłuższa wartość przetwarzania FFT otrzymana od klientów (L_{\max}). Następnie sumuje się ilorazy w postaci

$$L_{\text{calc}} = \frac{L_{\max}}{L_1} + \frac{L_{\max}}{L_2} + \dots + \frac{L_{\max}}{L_N}, \text{ a informacja, jaką część filtru będą przetwarzać klienci}$$

wynika z zależności: $W_k = \frac{L_{\max}}{L_1 \cdot L_{\text{calc}}} \cdot M$, (gdzie M to całkowita długość filtru). I tą wartość

rozsyła się według schematu omówionego w punkcie 5.10)

5.3.7. Pozostałe elementy systemu

Po stronie serwera:

- Mechanizm logowania klientów – jest odpowiedzialny za kontrolę i przechowywanie informacji o zalogowanych klientach (adres IP, informacje o mocy obliczeniowej itp.).
- Mechanizm wysyłania współczynników – na podstawie żądań od klientów mechanizm ten zapewnia podział współczynników na pakiety i wysłanie do żądającego klienta

Po stronie klienta:

- Mechanizm komunikacji z serwerem. Przyjmowanie współczynników – zapewnia nawiązywanie łączności z serwerem, ponowienie połączenia, komunikacje z serwerem (informacje o typie transmisji, przygotowanie do pracy), a także zapewnia pełną obsługę komunikacji w fazie rozsyłania współczynników
- Mechanizm przepakowania – ponieważ pakiety w sieci TCP posiadają ograniczoną długość, natomiast długość FFT liczonej po stronie klienta może przybrać różny rozmiar, mechanizm ten z pakietów o 512 elementach tworzy pakiety o N-elementach (przepakowanie w górę) bądź odwrotnie wówczas proces można nazwać przepakowaniem w dół.

6. Środowisko programistyczne Delphi

Do budowy aplikacji użyto kompilatora Delphi 6 Personal firmy Borland. Za cel wyznaczono sobie, aby cały projekt został wykonany w wersji ogólnie dostępnej, składającej się z najmniejszej ilości komponentów, stwarzając w ten sposób możliwość zarówno dalszego rozwijania kodu a także nauki języka na podstawie już stworzonego kodu, nie narażając na konieczność zakupu lepiej wyposażonych wersji kompilatora. Do obsługi protokołu sieciowego użyto darmowych kontrolerek firmy Indy o nazwie Indy Direct w wersji 9.10.

6.1. Delphi jako środowisko programistyczne typu RAD

Środowisko Delphi należy do narzędzi błyskawicznego tworzenia aplikacji RAD (ang. *Rapid Application Development*) w środowisku o platformie bazującej na systemie Microsoft Windows. Kompilatory wywodzące się z rodziny RAD odciążały programistę projektanta od uciążliwej pracy m.in. przy konstruowaniu interfejsu użytkownika, a pozwoliło skupić się na zadanym problemie programistycznym (w naszym przypadku komunikacji sieciowej i wykonywaniu obliczeń FFT oraz splotu).

6.2. Charakterystyka środowiska

Delphi wyposażone jest w obszerną, wizualną bibliotekę komponentów VCL (ang. *Visual Component Library*), stwarzającą możliwość wykorzystania już gotowych modułów bez konieczności ponownego oprogramowania powtarzających się czynności. Jak wspomniano w poprzednim punkcie do zalet Delphi należą łatwość obsługi, szybki i funkcjonalny kompilator, a także efektywny kod wynikowy.

Dlatego do budowy aplikacji zdecydowano się na wyżej wymienione środowisko programistyczne. Należy także nadmienić, iż od paru lat istnieje równoległe środowisko programistyczne pod nazwą Kylix bazujące na tym samym języku pozwalające na budowę aplikacji w systemie Linux. Pozwala to na łatwą „przenoszalność” kodu na inne platformy systemowe. Jednakże nie umożliwia tak wysokiej swobody systemowej jak aplikacja tworzona w Javie, ale za to dysponuje bardziej wydajnym kodem, a co przy założeniu iż głównym celem pracy jest środowisko obliczeniowe, a także na to iż przenoszalność kodu nie jest głównym wyznacznikiem wybór Delphi należy uważać za uzasadniony.

6.3. Język programowania Pascal

Język Pascal został stworzony przez Niclausa Wirtha - programistę, który również prowadził zajęcia dydaktyczne na Uniwersytecie Stanforda, a później w ETH na Politechnice w Zurychu. Wirth potrzebował języka, za pomocą którego mógłby on trafić do szerszego grona odbiorców. Stworzył coś, co teraz jest znane jako standard Pascal; nieco ograniczony język do tworzenia małych aplikacji zaprojektowany w latach 1970-1971.

Pascal jest pochodną PL1 (Język programowania 1) i ALGOLU (ALGOrytmiczny język). Pascal otrzymał nazwę na cześć Blaise Pascala – francuskiego matematyka, który zbudował pierwszą mechaniczną maszynę liczącą w XVII wieku..

Jedną z głównych cech pierwszej wersji Pascala było to, że używany kompilator wymagał bardzo małej pamięci. Kiedy w końcu lat siedemdziesiątych na scenie zaczęły pojawiać się pierwsze mikrokomputery, wiele instytucji edukacyjnych wybrało Pascal do nauki programowania na tych maszynach.

Jednak Pascal dalej nie stanowił żadnej konkurencji dla rozpowszechnionego języka C, dopiero po roku 1984, kiedy to Borland wprowadził Turbo Pascala zyskał miano profesjonalnego języka programistycznego. Produkt ten charakteryzował się bardzo wysokimi osiągnięciami, jeśli chodzi o czas kompilacji.

Następnym krokiem Borlanda było wprowadzenie wersji Pascala pod Windows. W niedługim czasie Pascal przeistoczył się w Object Pascal zyskując na możliwości operowania na obiektach.

6.4. Środowisko programistyczne: Delphi

Delphi stanowi kontynuację linii rozwojowej zapoczątkowanej pierwszą wersją Turbo Pascala. Pojawienie się programowania modułowego, nakładkowania i wykorzystanie assemblera, oraz wbudowany debugger, czy też wspomniane w poprzednim punkcie programowanie obiektowe a także pokonanie dotkliwej bariery 640K pamięci, to tylko niektóre z kolejnych funkcji wynikających ze stałego rozwoju i pracy nad tym językiem.

Postęp w dziedzinie systemów operacyjnych wymusił potrzebę wdrożenia kompilatora potrafiącego tworzyć aplikacje pod Windows, pierwsza wersja Delphi co prawda tworzyła jeszcze 16 bitowy kod wynikowy, ale pojawienie się tego typu kompilatora było istotną rewolucją w dziedzinie tworzenia oprogramowania. Kolejne wersje Delphi zaczęły wspomagać kolejne technologie związane z systemem Windows:

COM, ActiveX, serwery WWW, wielowarstwowe aplikacje SQL. Kolejnym krokiem było powstanie Kylixa odmiany Delphi (istnieje także odmiana C++ Buildera) na komputery pracujące pod kontrolą systemów UNIX/Linux.

6.5. Zalety języka Pascal

- Zoptymalizowany kompilator 32 bitowy, wielokrotnie szybszy od analogicznego zastosowanego w pokrewnym produkcie Borlanda C++ Builder
- Doskonała diagnostyka błędów, generowanie ostrzeżeń i wskazówek, co umożliwia tworzenie bardziej efektywnego kodu na ich podstawie.
- Możliwość wykorzystania kodu C++ ze względu na to, że oba języki korzystają ze wspólnego mechanizmu końcowego generowania kodu wynikowego.

6.6. Wady języka Pascal

- Szybka kompilacja i wygoda tworzenia aplikacji powoduje rozrost wielkości kodu wynikowego. Jednakże w dzisiejszych realiach nie jest to bardzo istotna wada.
- Ustupująca tylko językowi C++ szybkość działania aplikacji.
- Brak możliwości przenoszenia kodu na inne platformy systemowe, pod tym względem język C posiada większe możliwości nie mówiąc o liderze w tej klasyfikacji, jakim jest język Java

6.7. Podsumowanie

Delphi jest narzędziem do tworzenia szybkich aplikacji, zarówno tych o małym jak i średnim stopniu złożenia, doskonale nadaje się tam, gdzie wielkość kodu wynikowego nie jest krytycznym parametrem. W obecnej chwili bardzo istotnym aspektem przy tworzeniu projektów programistycznych jest tzw. czasookres powstania aplikacji, a także czas jej rozwoju i wykrycia/naprawienia błędu. Pod tym względem Delphi jest nie ocenione i dlatego zdobywa sobie coraz szersze grono zwolenników wśród profesjonalnych programistów.

7. Realizacja programowa wybranych zagadnień

7.1. Typy danych

W projekcie wszystkie dane są grupowane w tablicach o wcześniej zadeklarowanych rozmiarach. Pakiety przesyłane przez sieć niezależnie od rodzaju komendy, kierunku i typu transmisji (UDP/TCP) mają zawsze rozmiar 512 elementów. Elementami tablicy są 32 bitowe liczby całkowite ze znakiem (tzw. Smallint) o zakresie od -32768 do 32767. Wartości te reprezentują wartości próbek odczytanych z karty dźwiękowej.

a) TxPakiet:

0	1	n-1
---	---	-----	-----	-----

b) `TxPakiet = array [0..511] of smallint;`

Rys. 7.1. Reprezentacja ciągu próbek w pakiecie przesyłanym przez sieć o rozmiarze n=512, a) schemat pakietu, b) deklaracja pakietu w Delphi

Natomiast obliczenia FFT, a także splotu wykonuje się na liczbach zespolonych. Ponieważ standardowa biblioteka języka programowania Object Pascal nie posiada reprezentacji liczb zespolonych. Problem ten rozwiązano następująco:

Długość danych, na których operuje się po stronie klienta jest ściśle związana z jego mocą obliczeniową, ponieważ od niej zależy na jak dużej części współczynnika odpowiedzi impulsowej operuje komputer - klient, a także pośrednio od samej długości owego współczynnika. Aby umożliwić elastyczną modyfikację tych parametrów oraz pamiętając o tym, iż w metodzie Overlap Save najlepiej dobierać jak najmniejsze długości FFT z dostępnej puli możliwych, postanowiono jako nośniki informacji użyć tablic dynamicznych. Długość tablic jest wyznaczana tuż po podziale mocy obliczeniowej, a zarazem tuż przed rozpoczęciem obliczeń.

Stworzono zatem kolejny typ danych Tx:

a) Tx:

0	1	n-1
---	---	-----	-----	-----

c) Tx = array of smallint;

d) Var Lx: Tx

Begin

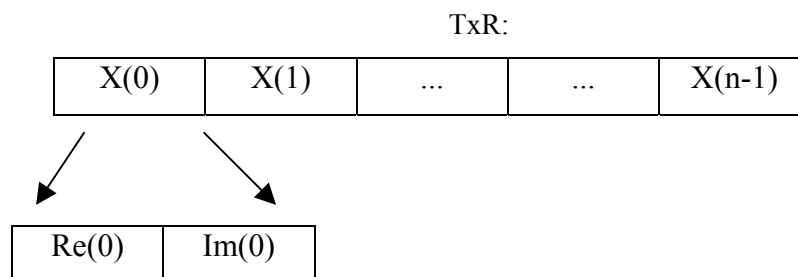
Setlength(Lx, SIZE);

...

End;

Rys. 7.2 Reprezentacja ciągu próbek w pakiecie, na którym dokonywane są obliczenia o dowolnym n , a) schemat pakietu, b) deklaracja pakietu w Delphi, c) konieczna inicjalizacja długości pakietu, bez niej dowołanie się do tablicy wywoła błąd dostępu (tablica nie zadeklarowana, pusta)

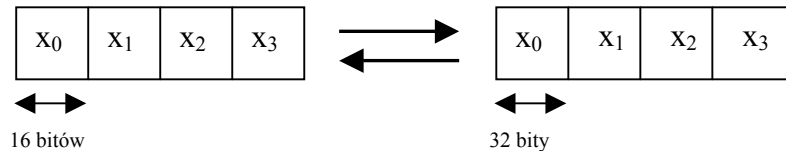
Aby móc operować na liczbach zespolonych elementami tablicy nie mogą być liczby całkowite, a rekordy składające się z dwóch pól: części rzeczywistej i urojonej. Rekord jest specyficznym typem tablicy umożliwiającym agregację różnych typów jednak o wcześniej zadeklarowanej postaci.



Rys. 7.3. Reprezentacja zbioru liczb zespolonych o ilości elementów n

Próbki otrzymywane z serwera posiadają tylko część rzeczywistą, część urojona jest równa zero i nie jest rozsyłana. Jednak w operacjach FFT należy dokonywać obliczeń na liczbach zespolonych. Dodatkowo aby uniknąć problemów związanych z przepełnieniem poszerza się zakres wartości typu danych na którym dokonywane będą obliczenia. W ten sposób 16 bitowe próbki przed wykonywaniem operacji Fourierowskich są poszerzane do

szerokości 32 bitów (ze znakiem). Operacje tą wykonuje klient po otrzymaniu próbek tuż przed liczeniem transformaty. Do tej operacji służy poniższa funkcja, parametrem wejściowym jest 16 - bitowa tablica ATab, natomiast rezultatem jest „poszerzona” 32 - bitowa tablica AWy. Funkcja „rozszerza” dane w sposób pokazany na rysunku.



Rys. 7.4. Zmiana tablicy 16bitowej w 32bitową tablicę liczb zespolonych (w prawo) oraz czynność do niej odwrotna (w lewo).

```
function ZmienRozmWgore(N: integer; ATab: Tx; var AWy: TxR):boolean;  
var i,size: integer;  
begin  
    size:= N-1;  
    if high(Awy) <> size then setlength(AWy, N);  
    for i:= size downto 0 do begin  
        AWy[i].Re:= ATab[i];  
        AWy[i].Im:= 0;  
    end;  
end;
```

Rys. 7.5. Listing funkcji ZmienRozmWGore

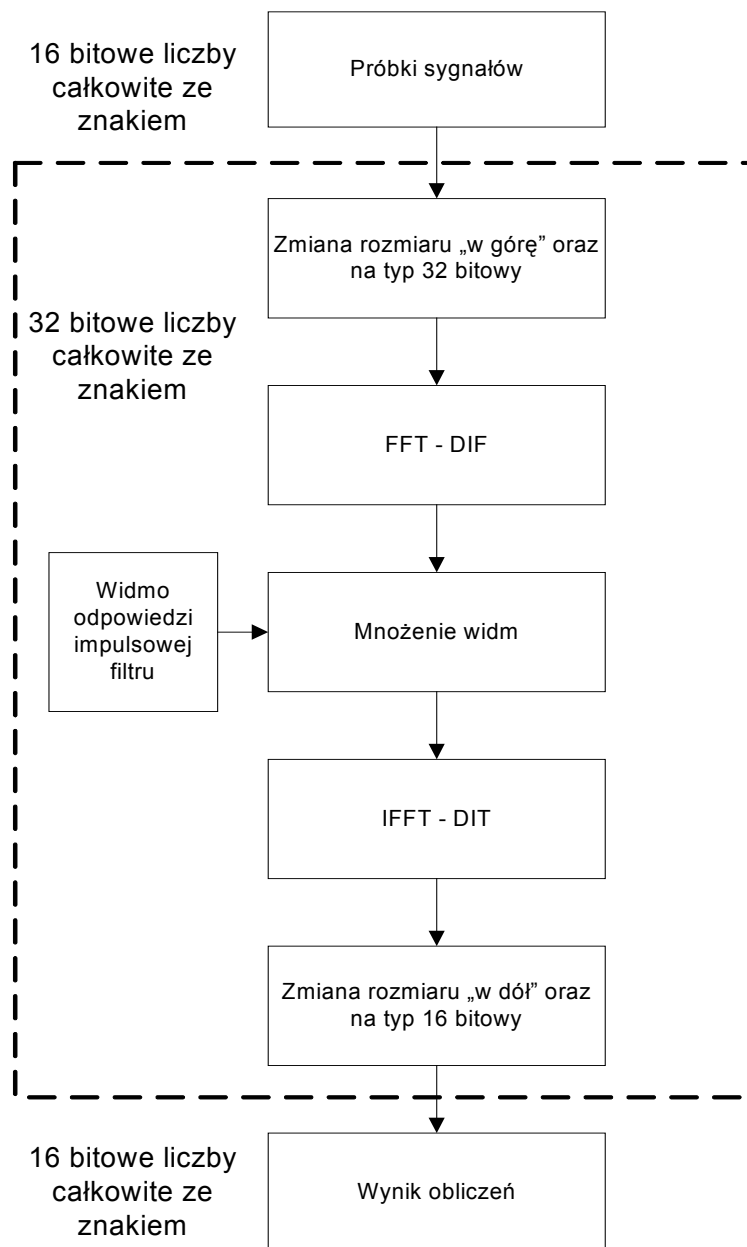
Po wykonaniu operacji obliczeniowych dokonywana jest analogiczna czynność, z tą różnicą że 32 bitowe liczby zespolone są przekształcane w tablicę 16 bitowych liczb całkowitych.

```
function ZmienRozmWDol (N: integer; ATab: TxR32; var AWy:  
Tx):boolean;  
var i,size: integer;  
begin  
    size:= N-1;  
    if high(Awy) <> size then setlength(AWy, N);  
    for i:= 1 to size do AWy[i]:= round(ATab[i].Re);  
end;
```

Rys. 7.6. Listing funkcji ZmienRozmWDol

7.2. Przebieg danych

Poniżej został przedstawiony schemat „wędrówki” danych, od momentu odbioru przez klienta aż do chwili wysłania wyniku. Należy jednak pamiętać, że „poszerzenie” zakresu wartości w wyraźny sposób zmniejsza wydajność algorytmów obliczeniowych



Rys. 7.7. Schemat „wędrówki” danych w procedurach obliczeniowych klienta

7.3. Algorytm radix-2 z podziałem w czasie DIT

Jako parametry wejściowe funkcja DIT przyjmuje dwa argumenty. Pierwszy z nich jest informacją o długości transformaty FFT jaką zamierza się wykonać. Drugim parametrem typu var (parametr wejściowy podlegający modyfikacji po wykonaniu funkcji) jest wejściowy sygnał $x(n)$, po wykonaniu procedury w parametrze tym znajdzie się widmo sygnału $X(n)$ (ewentualnie odwrotnie w przypadku liczenia IFFT).

```
procedure DIT (ANx: integer; var Ax:TxR);
var Nx,k: byte;
    l,m,p,q,r,qRe, qIm: integer;
    WiRe, WiIm: double;
    T: Tx32Rec;
    WTemp, WRe, WIm: double;
begin
    Nx:=trunc(log2(ANx));
    dec(ANx);
    m:= 1;
    for k:=0 to Nx-1 do begin // pętla numer 1
        l:= m shl 1; // długość bloków DFT
        WRe:= 1;
        WIm:= 0;
        WiRe:= cos(pi2/L); // mnożnik bazy Fouriera – część rzeczywista
        WiIm:= -sin(pi2/L); // część urojona

        for p:=0 to M-1 do begin // pętla numer 2
            q:=p;
            while q<ANx do begin // pętla numer 3
                r:= (q+M);
                T.Re:= (Ax[r].Re*WRe - Ax[r].Im*WIm) ; // "serce" FFT
                T.Im:= (Ax[r].Re*WIm + Ax[r].Im*WRe) ;

                Ax[r].Re:= Ax[q].Re - T.Re ; // nowa dolna próbka:
                Ax[r].Im:= Ax[q].Im - T.Im ; // górna minus "serce"
                Ax[q].Re:= Ax[q].Re + T.Re ; // nowa górna próbka:
                Ax[q].Im:= Ax[q].Im + T.Im ; // górna plus "serce"
                inc(q,L);
            end; // koniec pętli numer 3

            WTemp:= WiRe*WRe - WiIm*WIm; // kolejna wartość bazy Fouriera
            WIm:= WiRe*WIm + WiIm*WRe;
            WRe:= WTemp;
        end;

        m:= m shl 1; // ilość motylków w bloku (szerokość każdego motylka)
    end; // koniec pętli numer 1
end;
```

Rys. 7.8 Listing funkcji implementującej algorytm FFT DIT

Przebieg algorytmu składa się z trzech zagnieżdżonych pętli. Nadrzędna pętla wykonuje tyle obiegów ile etapów należy wykonać aby rozpoczynając od $N/2$ dwupunktowych DFT zakończyć na jednym N – punktowym. Parametrem modyfikowanym przez tę pętlę jest wartość parametru „ k ”. Łatwo obliczyć, że sumaryczna ilość obiegów jest równa $\log_2 N$. Druga w kolejności pętla wykonuje tyle obiegów ile na danym etapie jest wielopraczkowych DFT.

Najistotniejszą częścią algorytmu jest wykonanie czynności w trakcie obiegu w pętli trzeciej. W niej liczone są dolne (rRe , rIm) i górne (qRe , qIm) indeksy próbek motylka. Czyli dwie pary próbek które biorą udział w liczeniu dwuelementowego motylka DFT. Następnie jest liczone tzw. „serce” motylka, a jest nim wartość dolnego motylka pomnożonego przez odpowiednią wartość współczynnika W_i , adekwatną do indeksu danej próbki.

Następnie dokonuje się operacji przypisania wartości dolnej próbce (wartość próbki górnej minus „serce”) oraz wartości górnej próbce (wartość próbki górnej plus „serce”). Proces liczenia FFT metodą DIT opisany został w punkcie 2.6.2 niniejszej pracy.

7.4. Algorytm radix-2 z podziałem w częstotliwości DIF

Struktura parametrów wejściowych i wyjściowych jest identyczna do zaprezentowanej z algorytmie z podziałem w czasie (7.3).

Algorytm wykonuje FFT metodą podziału w częstotliwości, zatem na wejście podaje się sygnał o normalnej kolejności próbek, na wyjściu próbki posiadają przestawioną kolejność.

```
procedure DIF (ANx: integer; var Ax:TxR32);
var Nx,k: byte;
    l,m,p,q,r: integer;
    a,b: double;
    T: Tx32Rec;
    WiRe, WiIm, WTemp, WRe, WIm: double;
begin
    Nx:=trunc(log2(ANx));
    m:= ANx shr 1;
    dec(ANx);

    for k:=Nx-1 downto 0 do begin // pętla numer 1
        l:= m shl 1; // długość bloków DFT
        WiRe:= 1;
        WiIm:= 0;
        WRe:= cos(pi2/L); // mnożnik bazy Fouriera – część rzeczywista
```

```

WIm:= -sin(pi2/L); // część urojona

for p:=0 to M-1 do begin // pętla numer 2
  q:=p;
  while q<ANx do begin // pętla numer 3
    r:= (q+M);

    T.Re:= Ax[r].Re + Ax[q].Re;
    T.Im:= Ax[r].Im + ax[q].Im;

    a:= Ax[q].Re - Ax[r].Re;
    b:= Ax[q].Im - Ax[r].Im;

    Ax[r].Re:= (a*WiRe-b*WiIm);
    Ax[r].Im:= (a*WiIm+b*WiRe);

    Ax[q].Re:= T.Re;
    Ax[q].Im:= T.Im;

    inc(q,L); // koniec pętli numer 3
  end;
  WTemp:= WRe*WiRe - WIm*WiIm; // kolejna wartość bazy Fouriera
  WiIm:= WRe*WiIm + WIm*WiRe;
  WiRe:= WTemp;
end; // koniec pętli numer 2

m:= m shr 1; // ilość motylków w bloku (szerokość każdego motylka)
end; // koniec pętli numer 1
end;

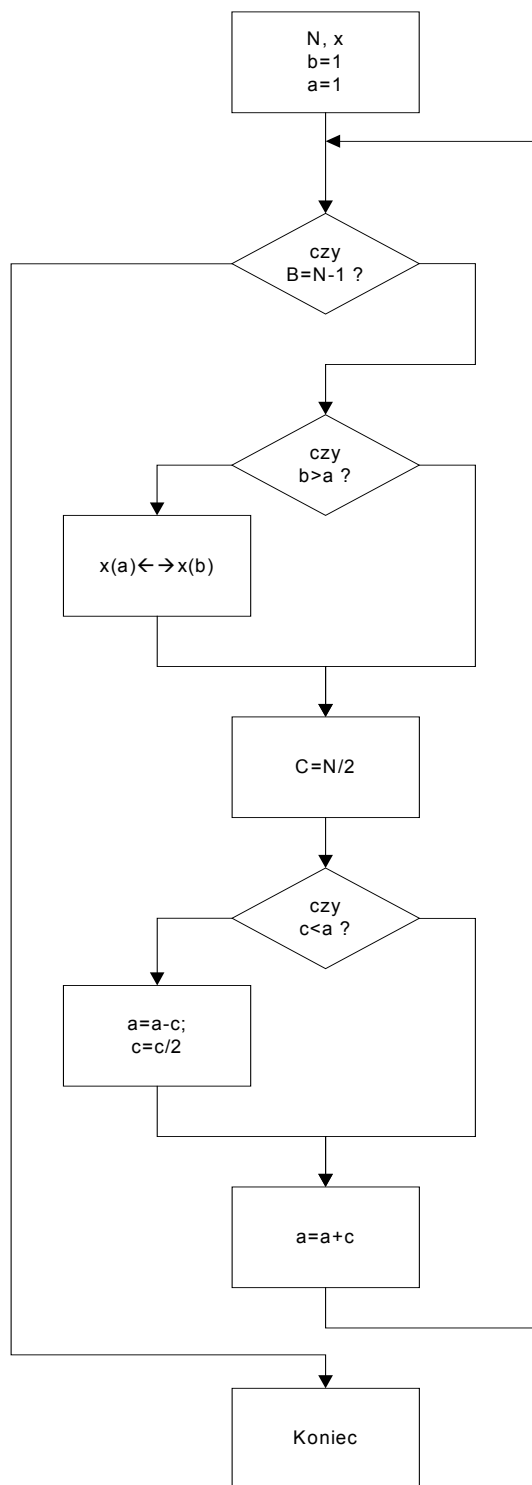
```

Rys. 7.9 Listing funkcji implementującej algorytm FFT DIF

Algorytm jest stworzony w podobnej konwencji co algorytm DIT. Podstawową różnicą jest zachowanie się w obrębie wykonania pętli trzeciej. Wartość próbek jest mnożona z wartością współczynnika W_i dopiero po wykonaniu dodawania lub odejmowania. Do górnej próbki (qRe , qIm) wpisywany jest wynik dodawania wartości próbki dolnej i górnej. Natomiast do próbki dolnej (rRe , rIm) wpisuje się wynik przemnożenia współczynnika W_i przez różnicę wartości próbek dolnej i górnej. Proces liczenia FFT metodą DIF opisany został w punkcie 2.6.3 niniejszej pracy.

7.5. Algorytm przestawiania kolejności próbek

Aby zapewnić poprawną kolejność próbek wejściowych (DIT) lub wyjściowych (DIF) np. w celu policzenia widma odpowiedzi impulsowej filtra potrzebna jest funkcja przestawiająca kolejność próbek. Zaimplementowano szybki algorytm przestawiania kolejności próbek tzw. „w miejscu”. Zasadę działania tłumaczy schemat.



Rys. 7.10. Algorytm przestawiania kolejności próbek

```
procedure BitReverse(AN: integer; var Ax: TxR);
var a,b,c, d,e: integer;
    T: TxRec;
begin
    a:= 1;
    for b:=1 to AN-1 do begin
        if b<a then begin
            T.Re:= Ax[a-1].Re;
            T.Im:= Ax[a-1].Im;

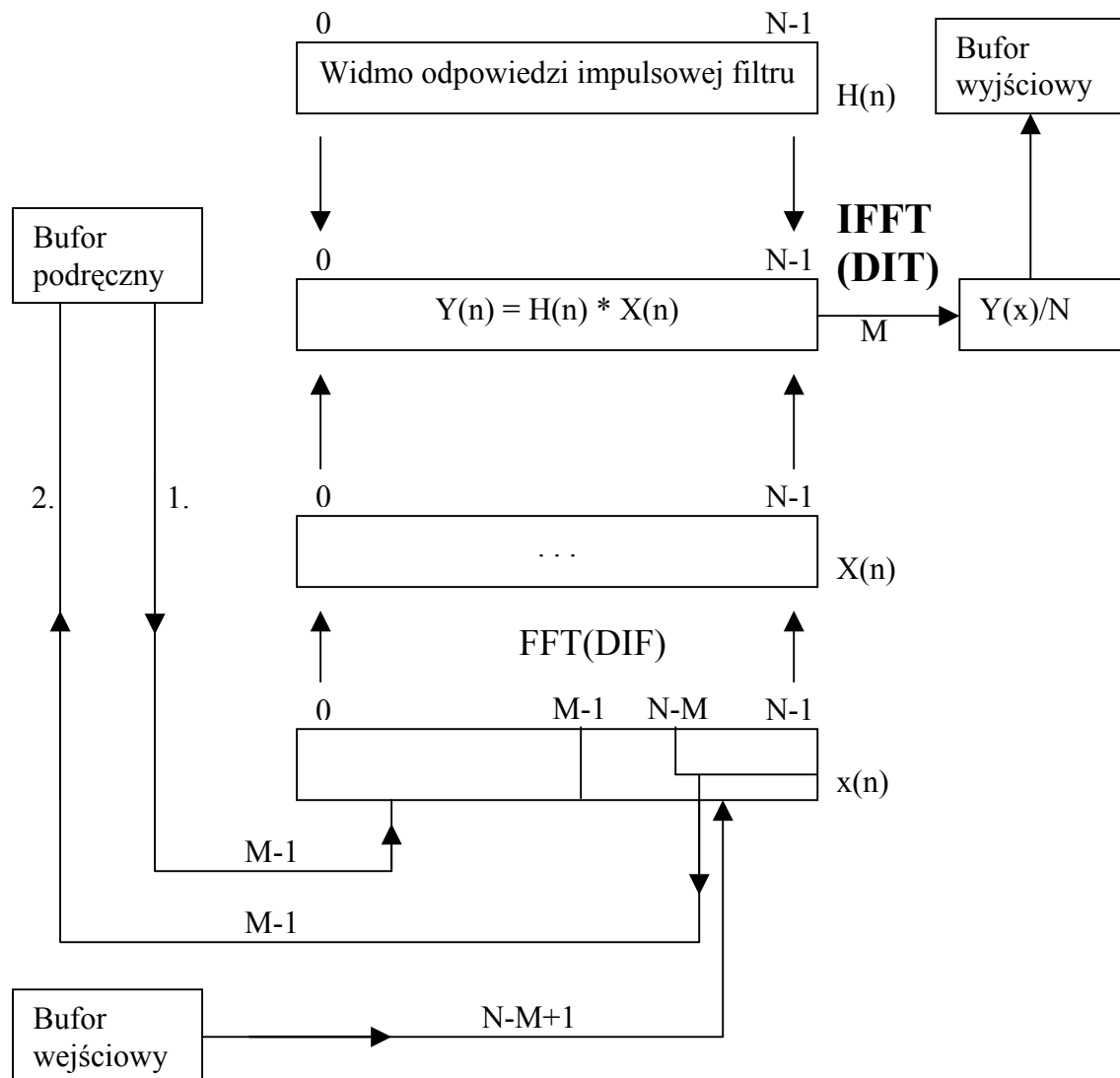
            Ax[a-1].Re:= Ax[b-1].Re;
            Ax[a-1].Im:= Ax[b-1].Im;

            Ax[b-1].Re:= T.Re;
            Ax[b-1].Im:= T.Im;
        end;

        c:= AN shr 1;
        while c<a do begin
            dec(a,c);
            c:= c shr 1;
        end;
        inc(a,c);
    end;
end;
```

Rys. 7.11. Listing funkcji implementującej algorytm przestawiania kolejności próbek BitReverse

7.6. Algorytm Overlap-save



Rys. 7.12. Schemat przedstawiający algorytm Overlap-save

W pierwszym kroku należy zapewnić (w poniższej procedurze nie jest to zaznaczone, gdyż operacja ta wykonywana jest podczas kreacji aplikacji) aby bufor podręczny `FTabCache` został wyzerowany, to znaczy że wszystkie jego elementy powinny posiadać wartość zero.

Następnie wykonywane jest cykliczne pobieranie elementów z bufora wejściowego. Procedura przepakowania (omówiona w punkcie 7.6) musi zapewniać taką organizację kolejki, aby znajdowały się w niej elementy o długości $N-M+1$ tak, aby podczas wywołania metody `Pop` z bufora wejściowego pobrać tylko te elementy, które są potrzebne

dla bieżącej operacji. Po obliczeniach ponownie zostają przepakowane dane z tablicy N -elementowej w wiele tablic 512 elementowych analogicznie, aby w procedurze wysyłania pakietów podczas wywołania metody `Pop` pobrać tylko te elementy, które w danym cyklu zostaną wysłane. Powyższe założenia wpływają bardzo na sprawne działanie algorytmu, a przede wszystkim na wzrost wydajności.

Z bufora podręcznego pobierane jest $M-1$ elementów i umieszczane są na początku tablicy elementów $x(n)$. Wynika to z zasady działania metody `Overlap – save` i są to zachodzące na siebie próbki. Z kolei po pobraniu próbek z bufora wejściowego do bufora podręcznego natychmiast przepisuje się $M-1$ elementów liczonych od końca.

Dysponujemy policzonym wcześniej widmem $H(n)$, zatem brakuje nam tylko widma uzyskanego z sygnału $x(n)$. Dokonujemy więc liczenia transformaty (metodą `DIF`, gdyż taka zachowuje normalną kolejność indeksów na wejściu) i po wykonaniu tej operacji dokonujemy wymnażania wyraz po wyrazie obu widm $X(n)$ i $H(n)$.

Uzyskany sygnał (widmo) zostaje poddane wstecznej transformacie `FFT` (tym razem metodą `DIT`, aby na wyjściu uzyskać normalną kolejność próbek). Następnie wszystkie elementy są dzielone przez liczbę N oznaczającą długość transformaty `FFT`. Wynik końcowy jest przepisywany do bufora wyjściowego metodą `Push`.

```
procedure TComputeThread.Execute;
begin
  while not Terminated do begin
    try
      Compute.OverlapSave;
    finally
      end;
    sleep(1);
  end;
end;
```

Metoda `Overlap-save` działa na zasadzie osobnego wątku, działającego cały czas w pętli nieskończonej (warunkiem wyjścia jest zakończenie aplikacji), który sprawdza zmianę stanu bufora wejściowego, jeżeli w buforze zmieni się ilość elementów następuje bezzwłoczne wywołanie procedury `Overlap`. Wykonaniu funkcji `Overlap` przydzielony jest wysoki priorytet, zatem podczas działania tej funkcji inne wątki o niższym priorytecie będą miały mniej czasu na wykonanie swoich czynności.


```
function TCompute.OverlapSave32: boolean;
var PBlok: Px;
    LBlok: Tx;
    LBlok32: TxR32;
    i,j,k,prog: integer;
begin
    if (QInL.Count=0) or (not FPoliczoneWsp) then exit;
    setlength(LBlok, FNx);
    setlength(LBlok32, FNx);

    while QInL.Count>0 do begin
        PBlok:= QInL.Pop;
        if high(PBlok^)=0 then dispose(PBLOK) else begin
            if not Main.ServNieDokObl then begin
                //załadowanie tablicy 16bitwoej elementami z cache'a <0..M-2>
                //oraz z bufora wejściowego <M-1..N-1>
                j:=0;
                k:=0;
                prog:= FNx- FMx.Size+ 1;
                for i:=0 to FNx-1 do begin
                    if i < FMx.Size-1 then LBlok[i]:= FTabCache[i]
                    else begin
                        LBlok[i]:= PBlok^[j];
                        inc(j);
                    end;

                    if i >= prog then begin //ładowanie cache'a elementami
                                            <M-
                                            //1..N-1>
                        FTabCache[k]:= LBlok[i];
                        inc(k);
                    end;
                end;

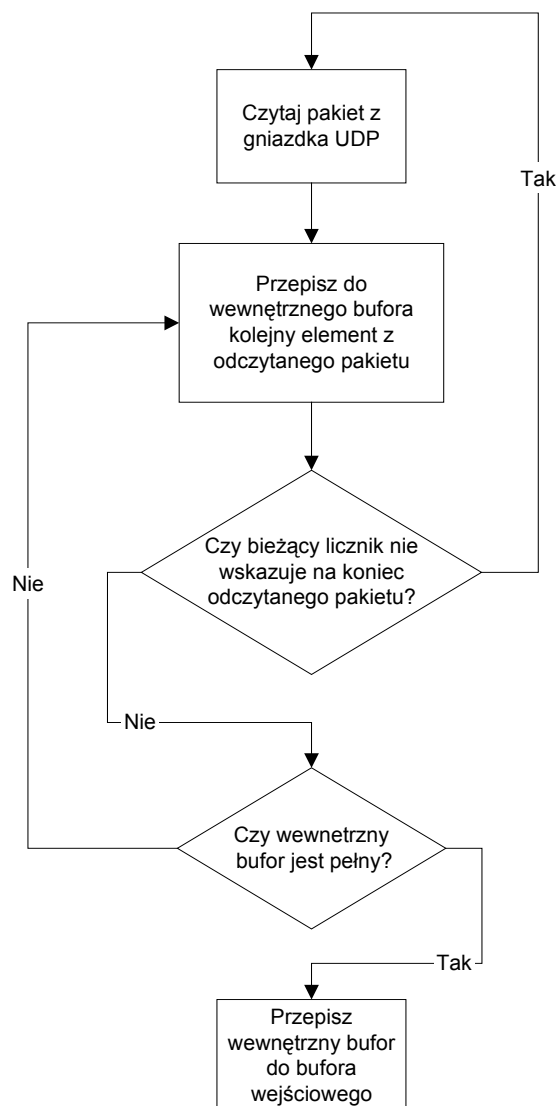
                //Poszerz zakres danych i wprowadz liczby zespolone
                ZmienRozmWgore(FNx, LBlok, LBlok32);
                DIF(FNx, LBlok32);
                BitReverse(FNx, LBlok32);
                LBlok32:= MnozFFTDziel(LBlok32, FTabWspFFTR32);
                BitReverse(FNx, LBlok32);
                IFFT(FNx, LBlok32, LBlok);

                for i:=FMx.Size-1 to FNx-1 do
                    PBlok^[i-FMx.Size+1]:= LBlok[i];
            end;
            QOut.Push(PBlok);
        end;
    end;
end;
```

Rys. 7.13 Listing funkcji Overlap-save

7.7. Algorytm przepakowania w buforze wejściowym klienta

Funkcja `Przepakuj` realizuje algorytm „przepakowania” tablic 512 elementowych w tablicy o zadanym rozmiarze o długości $L=N-M+1$. O celowości działania tego algorytmu wspomniano w punkcie 7.5. Mechanizm jest tak skonstruowany, iż jest „odporny” zarówno na większą jak i mniejszą wartość L od 512. Algorytm przechowuje elementy które z ostatniego pakietu nie zmieściły się do L -elementowej tablicy. Funkcja przepisuje tworzony w niej pakiet do bufora wejściowego dopiero gdy odbierze z gniazdka UDP wystarczającą ilość elementów.



Rys. 7.14. Zasada działania mechanizmu przepakowującego na wejściu

```
function TCompute.Przepakuj(APaczka: PPaczka): boolean;
var lIndex, lsize: smallint;
begin
    lSize:= PaczkaSize(APaczka);

    if lSize<FLx then begin
        if FIndeksPakowania=0 then begin
            new(FBlokLewy);
            setlength(FBlokLewy^, FLx);
            for lIndex:=0 to lSize-1 do begin
                FBlokLewy^[lIndex]:= APaczka.Data[lIndex];
            end;
            inc(FIndeksPakowania, lsize);

        end else if FIndeksPakowania + lSize <= FLx then begin
            for lIndex:=0 to lSize-1 do begin
                FBlokLewy^[FIndeksPakowania + lIndex]:=
APaczka.Data[lIndex];
            end;
            inc(FIndeksPakowania, lsize);

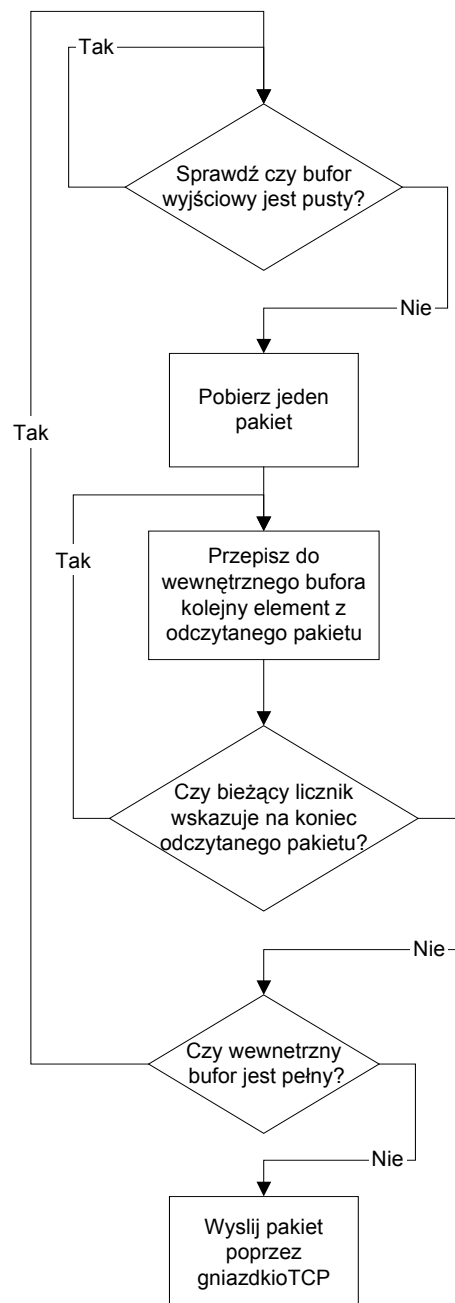
        end else if FIndeksPakowania + lSize > FLx then begin
            for lIndex:=0 to FLx - FIndeksPakowania - 1 do begin
                FBlokLewy^[FIndeksPakowania + lIndex]:=
APaczka.Data[lIndex];
            end;
            QInL.Push(FBlokLewy);
            new(FBlokLewy);
            setlength(FBlokLewy^, FLx);
            for lIndex:= FLx - FIndeksPakowania to lSize-1 do begin
                FBlokLewy^[lIndex-FLx + FIndeksPakowania]:=
APaczka.Data[lIndex];
            end;
            FIndeksPakowania:= FIndeksPakowania + lSize - FLx;
        end;
    end else if lSize=FLx then begin
        new(FBlokLewy);
        for lIndex:=0 to lSize-1 do FBlokLewy^[lIndex]:=
APaczka.Data[lIndex];
        QInL.Push(FBlokLewy);

    end else begin
        if FBlokLewy = nil then begin
            new(FBlokLewy);
            setlength(FBlokLewy^, FLx);
        end;
        for lIndex:=0 to lSize-1 do begin
            FBlokLewy^[FIndeksPakowania]:= APaczka.Data[lIndex];
            if (FIndeksPakowania+1) mod FLx = 0 then begin
                QInL.Push(FBlokLewy);
                new(FBlokLewy);
                setlength(FBlokLewy^, FLx);
                FIndeksPakowania:=0;
            end else
                inc(FIndeksPakowania);
        end;
    end;
end;
```

Rys. 7.15. Listing funkcji realizującej przepakowanie pakietów

7.8. Algorytm przepakowania w buforze wyjściowym klienta

Funkcja pełni podobne zadanie co algorytm 7.6, z tą różnicą że przepakowuje tablice $(N-M+1)$ elementowe w tablice o zawartości 512 elementów i dodatkowo dokonuje wysłania poprzez gniazdko TCP pakietu danych z wynikami obliczeń.



Rys. 7.16. Zasada działania mechanizmu przepakowującego na wejściu

```
procedure TSendThread.Execute;
var LPx: Px;
    LPPaczka: PPaczka;
    LPaczka: TPaczka;
    LIndex, LLicznik: word;
begin
    FIndeksPakowania:=0;
    FPaczka.Command:= COMM_WYNIKI;
    FPaczka.CID:= Main.UserID;
    FPaczka.Size:= 0;

    while not Terminated do begin
        try
            while QOut.Count > 0 do begin
                LPx:= QOut.Pop;
                try
                    for lIndex:=0 to high(LPx^) do begin
                        try
                            FPaczka.Data[FIndeksPakowania]:= LPx^[lIndex];
                        except
                            Main.Zaloguj('E2: IndPak:'+IntToStr(FIndeksPakowania)+'',
lIndex:'+IntToStr(lIndex));
                        end;

                        if (FIndeksPakowania>0)and((FIndeksPakowania+1) mod
(high(FPaczka.Data)+1)=0) then begin
                            FPaczka.ID:= Licznik;
                            FPaczka.CID:= Main.UserID;
                            if Main.IdTCP.Connected then
Main.IdTCP.WriteBuffer(FPaczka, sizeof(FPaczka))
                            else begin
                                Main.Zaloguj('Opróżnianie bufora, stan:
'+IntToStr(QOut.Count));
                                dispose(LPx);
                                end;

                                if Main.IdTCP.Connected then
Main.IdTCP.WriteBuffer(FPaczka, sizeof(FPaczka))
                                else begin
                                    dispose(LPx);
                                    end;

                                    FIndeksPakowania:= 0;
                                    inc(Licznik);
                                end else inc(FIndeksPakowania);
                            end;
                        finally
                            end;
                        end;
                    end;
                    sleep(1);
                except
                    end;
                end;
            end;
        end;
```

Rys. 7.17. Listing funkcji realizującej przepakowanie pakietów na wejściu

7.9. Algorytm kolejkowania w buforze wejściowym serwera

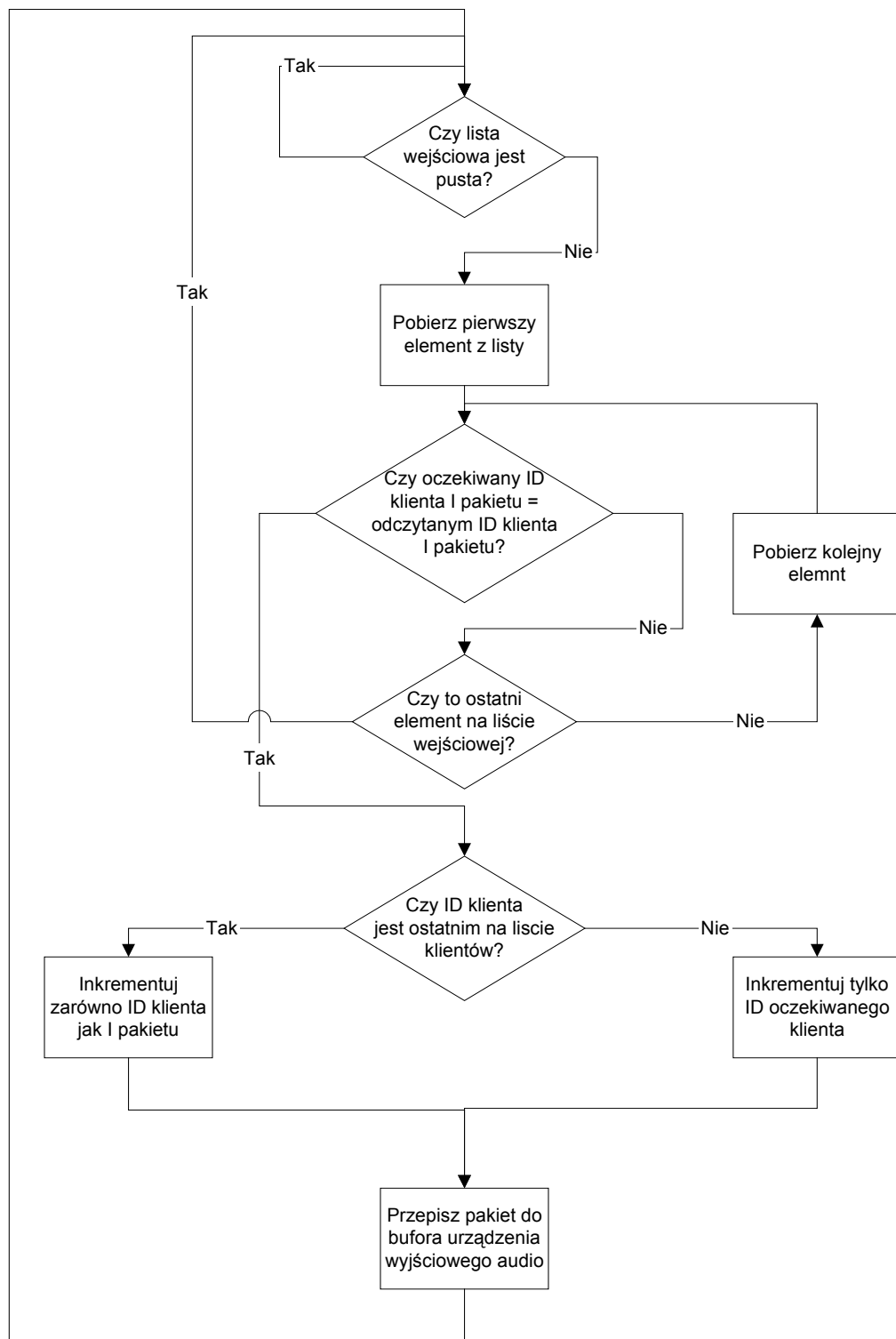
Algorytm ten służy do wybierania kolejnych elementów pobieranych z bufora wewnętrznego będącego listą wskaźników do pakietów odebranych z gniazdka TCP aplikacji serwera i umieszczone w obszarze pamięci. Stosowanie tradycyjnej kolejki w tym przypadku nie jest stosowne. Pakiety mogą przychodzić w różnych kolejnościach, tzn. w zależności od długości transformaty FFT po stronie klienta, chwilowego spadku lub wzrostu wydajności, oraz parametru ilościowego jakiej części zadania są przydzielone danemu komputerowi pakiet wcześniej wysłany do klienta może przyjść później od tego wysłanego po nim.

Zatem algorytm ten powinien mieć wewnętrzny licznik w którym brałby pod uwagę zarówno identyfikator pakietu jak i też identyfikator klienta po odczytaniu pierwszego pakietu inkrementuje znacznik identyfikatora klienta spodziewając się odczytania z listy pakietu o tym samym identyfikatorze pakietu, ale o kolejnym identyfikatorze klienta. Inkrementacja obu identyfikatorów nastąpi wówczas, gdy znacznik identyfikatora będzie wskazywał na ostatniego klienta na liście.

Jeżeli zostanie „trafiony” oczekiwany pakiet (zgodność identyfikatorów oczekiwanego i odczytanego) wówczas ten właśnie pakiet zostaje skierowany bezpośrednio do bufora urządzenia wyjściowego audio.

Jednak taki mechanizm mógłby ulec zakleszczeniu w przypadku gdyby zaginął jeden tylko pakiet, jeżeli mechanizmowi retransmisji nie uda się dokonać ponownego wysłania tego pakietu, wewnętrzna logika algorytmu powinna uznać oczekiwany pakiet za już nieważny i go pominąć (szukać kolejnego następnego po nim). Algorytm decyduje się na takie działanie w momencie kiedy bufor urządzenia audio staje się pusty. Oczywiście powodem takiego działania może być też to, że moc obliczeniowa klientów w tym czasie spadła.

Działanie tego algorytmu jest newralgicznym punktem pracy całego systemu, działa na osobnym wątku tak, aby w razie potrzeby móc przejąć jak największą moc obliczeniową komputera-serwera np. w chwili przeszukiwania listy.



Rys. 7.18. Zasada działania mechanizmu kolejującego na wejściu serwera

```
procedure TQueueThread.Execute;
var i,j,licznik, lMx: integer;
    Paczka: PPaczka;
    count: byte;
begin
    QIn.Capacity:= 1000;
    FInit:= false;
    FBufferValue:= DEFAULT_BUFFER_VALUE;
    new(LTx);
    for i:=0 to high(LTx^.Data) do LTx^.Data[i]:= 0;
    count:= 255;

    while not Terminated do begin
        try
            i:= 0;
            licznik:= 0;
            if count=255 then count:= MainForm.Clients.Count;

            while licznik < QIn.Count do begin
                Paczka:= QIn.Items[i];
                if Paczka.ID = FLastPacket.ID then begin
                    AddTx(LTx, Paczka);
                    QIn.Delete(i);
                    dispose(Paczka);
                    dec(count);
                end else inc(i);

                inc(licznik);

                if count =0 then begin
                    QQ.Push(PPaczka(LTx));
                    new(LTx);
                    FLastPacket.ID:= FLastPacket.ID + 1;
                    for j:=0 to high(LTx^.Data) do LTx^.Data[j]:= 0;
                    count:= MainForm.Clients.Count;
                end;

                end;
                i:=0;
                sleep(1);
            finally
                end;
            end;
            dispose(LTx);
        end;
    //-----
    ---function TQueueThread.AddTx(var LTx: PPaczka; Pakiet: PPaczka):
    boolean;
    var i: word;
    begin
        for i:=0 to high(LTx.Data) do
            LTx.data[i]:= LTx.data[i] +   Pakiet.Data[i];
        end;
    end;
```

Rys. 7.19. Listing algorytmu realizującego mechanizm kolejowania na wejściu serwera

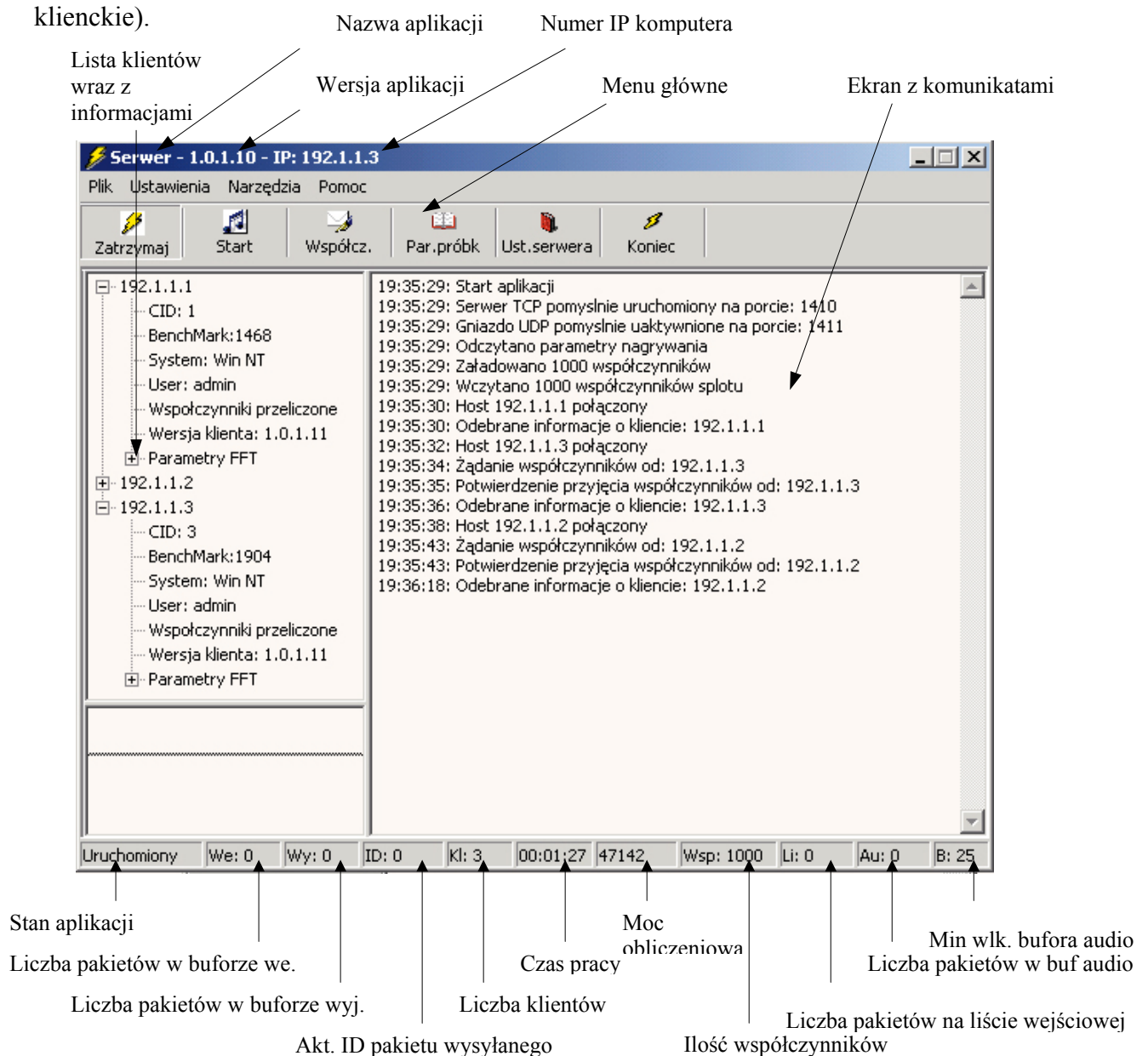
8. Użytkowanie systemu

Wymagania dotyczące poprawnej konfiguracji systemu zostały omówione w punkcie dotyczącym architektury.

8.1. Obsługa aplikacji zarządcy

8.1.1. Główny ekran

Po uruchomieniu aplikacji i uruchomieniu serwera TCP na ekranie komputera ukaże się następujący widok (zakładamy, że równolegle zostały uruchomione aplikacje klienckie).



Rys. 8.1. Główny ekran aplikacji zarządcy

Lista klientów zawiera oprócz posortowanej po adresie IP liście komputerów przyłączonych do systemu informacje o każdym z klientów z osobna. Na liście szczegółowych informacji możemy znaleźć informacje o przydzielonym numerze CID dla tego klienta, wyniku testu wydajnościowego, systemie operacyjnym komputera, użytkownika zalogowanym na komputerze, informacji o posiadaniu współczynników przez klienta, wersji aplikacji (jest to ważna informacja w przypadku dalszego rozwoju niedopuszczająca do udziału w transmisji starszych wersji aplikacji), a także parametry FFT klienta, czyli długość transformaty FFT oraz przydzielony zakres od-do części odpowiedzi impulsowej.

Informacje u dołu ekranu pełnią jedynie rolę kontrolną, informująca o przebiegu transmisji i podstawowych parametrach pracy systemu.

8.1.2. Ustawienia pracy aplikacji

Menu → Ustawienia → Ustawienia serwera

Parametry serwera

Automatycznie uruchom następujące funkcje:

- ☒ Otwórz gniazdo TCP
- ☒ Uaktywnij gniazdo UDP
- ☒ Czytaj współczynniki z pliku

Ustawienia gniazd:

Port TCP	Port UDP
1410	1411

Kontrola aktywności klientów (ping-pong):

- ☒ Usługa aktywna, przeprowadzaj co: 60 sekund

- ☒ Logowanie komunikatów
- ☒ Zapisuj ustawienia
- ☒ Udostępnij tryb serwisowy

Anuluj Zapisz

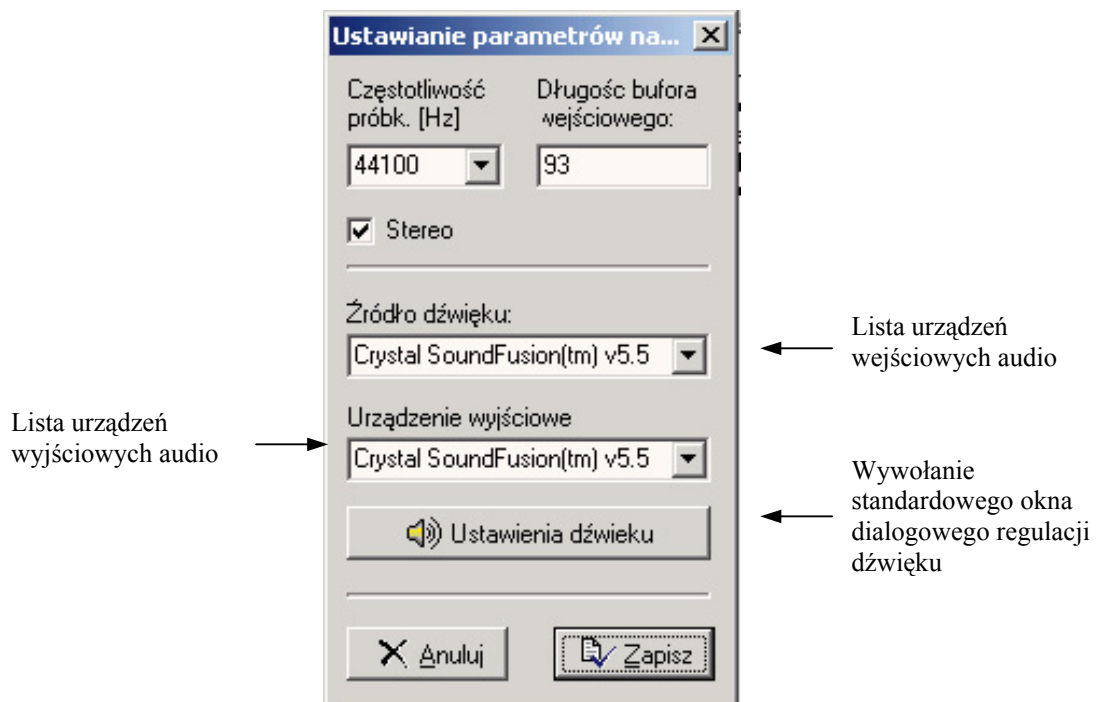
Rys. 8.2. Ustawienia pracy aplikacji

Zarządca posiada dwa główne parametry dotyczące jego pracy sieciowej: numery portów gniazd TCP i UDP. W przypadku nie zadeklarowania innych wartości parametry te przyjmują następujące numery portów: dla TCP = 1410, natomiast dla UDP = 1411. Dodatkowo można ustalić czy usługi TCP i UDP będą uruchamiane zaraz po starcie aplikacji. W okienku formatki ustawień serwera można jeszcze modyfikować usługę kontroli aktywności (ping-pong) sprawdzającą czy wszyscy klienci na liście są dalej aktywnymi klientami. Pozostałe ustawienia służą jedynie dla wygody obsługi aplikacji.

8.1.3. Ustawienia parametrów próbkowania

Menu → Ustawienia → Parametry próbkowania

Następnymi parametrami są parametry transmisji audio. Czyli: częstotliwość próbkowania, ilość kanałów fonicznych oraz szerokość bitowa próbek audio.

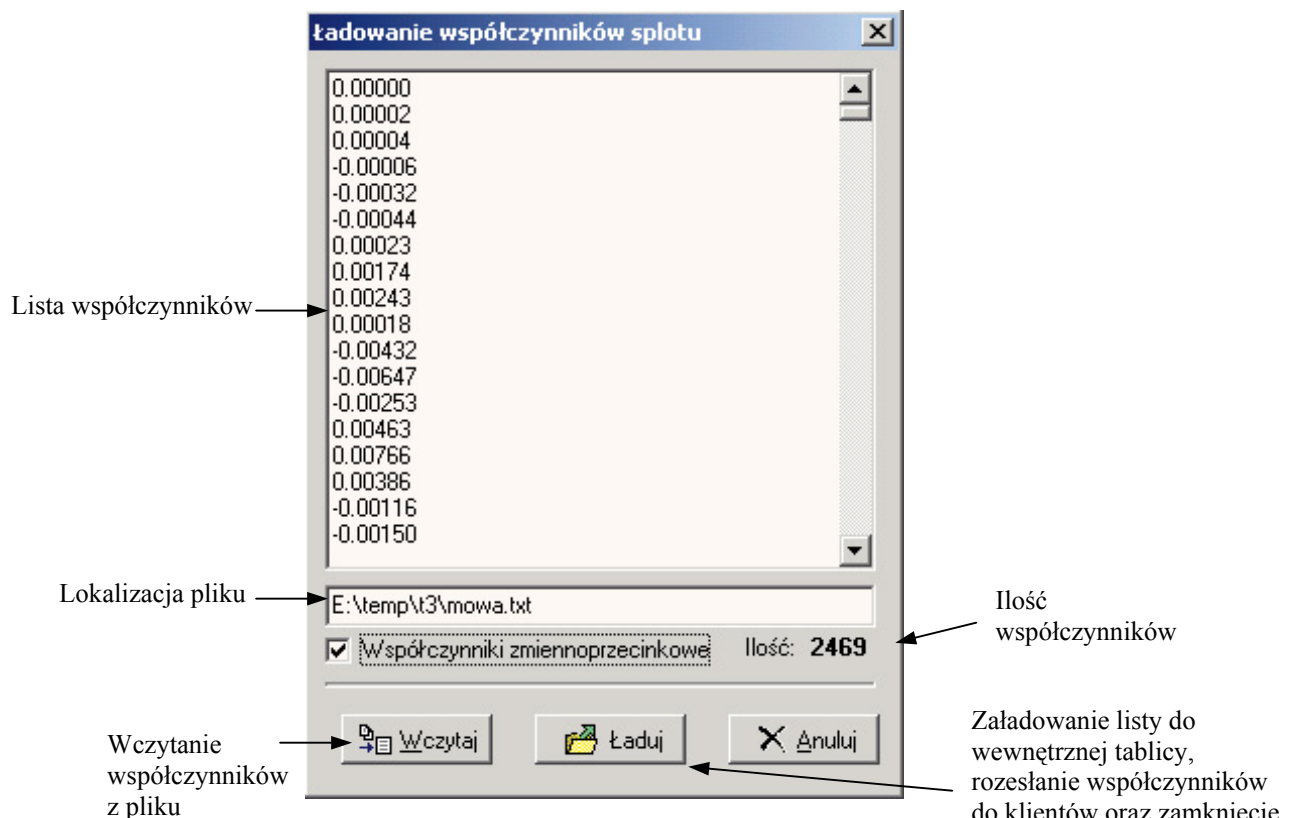


Rys. 8.3. Ustawienia parametrów próbkowania

8.1.4. Ustawienia współczynników odpowiedzi impulsowej

Menu → Ustawienia → Współczynniki

Przy pierwszym uruchomieniu aplikacji należy załadować z pliku współczynniki odpowiedzi impulsowej filtru. Załadowanie tablicy współczynników jest niezbędne do rozpoczęcia transmisji. Parametry mogą mieć zarówno postać zmiennoprzecinkową z przedziału $<-1 \dots 1>$ oraz postać liczb całkowitych ze znakiem $<-32768 \dots 32767>$. Po załadowaniu listy z pliku współczynniki są od razu rozsyłane do klientów. Istnieje możliwość pamiętania ścieżki lokalizacji pliku i ładowania ich automatycznie każdorazowo przy starcie aplikacji (przy ustawionym parametrze punkt 8.1.2). Akceptowalny plik z parametrami jest plikiem tekstowym z podziałem kolejnych elementów zarówno poprzez znaczniki $<CR>$ $<NL>$ jak i samo $<CR>$. Istnieje także możliwość ładowania odpowiedzi impulsowej z pliku typu WAVE (*.wav)

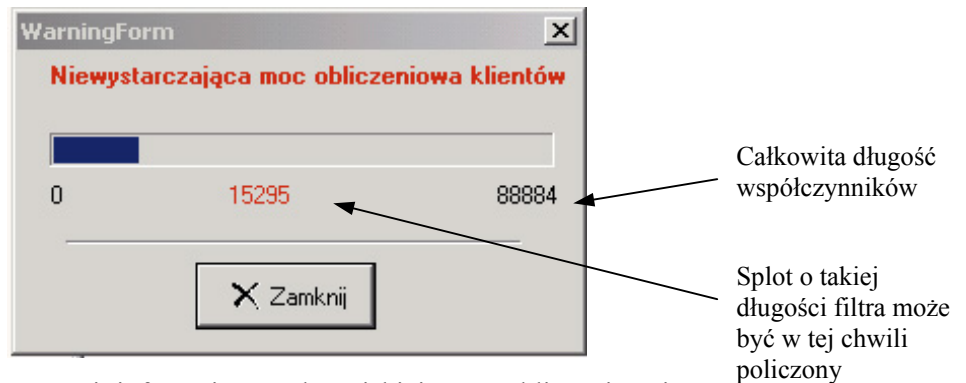


Rys. 8.4. Ustawienia współczynników

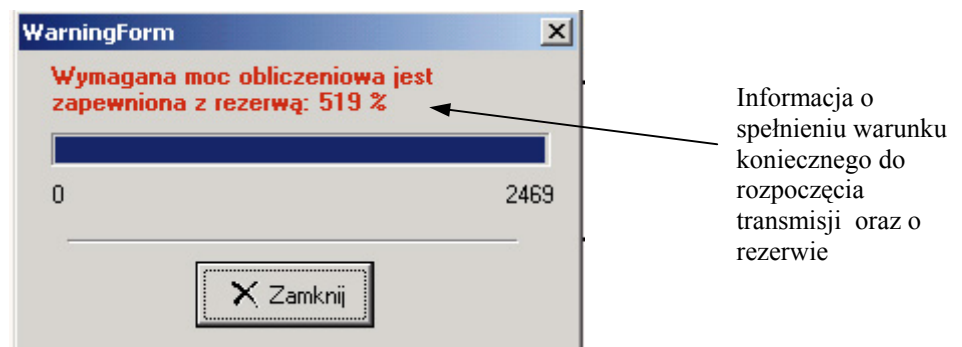
8.1.5. Rozpoczęcie transmisji

Menu → Ustawienia → Start nagrywania

Tuż przed samym procesem rozpoczęcia próbkowania sprawdzana jest wydajność całego systemu, jeżeli jest ona zbyt niska wówczas wyświetlona zostanie odpowiednia informacja i nie dojdzie do procesu rozpoczęcia transmisji aż do chwili zwiększenia mocy obliczeniowej w wyniku zalogowania dodatkowych klientów.



Rys. 8.5. Monit informujący o zbyt niskiej mocy obliczeniowej



Rys. 8.6. Monit informujący o spełnieniu koniecznego warunku do rozpoczęcia transmisji

Teraz po zaakceptowaniu i naciśnięciu klawisza „Zamknij”, w przypadku spełnienia warunku zostaje wyświetlona lista aktywnych klientów. Równolegle dokonuje się podział obowiązków i rozsyłanie tychże informacji do klientów, klienci odpowiadają potwierdzeniem i z chwilą, gdy wszyscy odeślą potwierdzenia zostaje umożliwione już bezpośrednie rozpoczęcie procesu próbkowania i transmisji próbek.



Rys. 8.7. Lista klientów i informacja o odpowiedzi z ich strony po rozesłaniu podziału zadań

8.1.6. Transmisja

Sam etap transmisji danych przebiega bezobsługowo. Użytkownik będzie obserwował cyklicznie zwiększający się identyfikator pakietu (PID) . Niewielkie chwilowe zwiększenie ilości elementów w buforach oznacza poprawne funkcjonowanie systemu, choć ich zbyt duży wzrost nie jest pożądany. Sygnał akustyczny po przetworzeniu jest odtwarzany w urządzeniu zadeklarowanym jako zewnętrzne urządzenie audio.



Rys. 8.8. Informacja o stanie bieżącej transmisji

8.1.7. Kończenie transmisji

Menu → Ustawienia → Zakończenie nagrywania

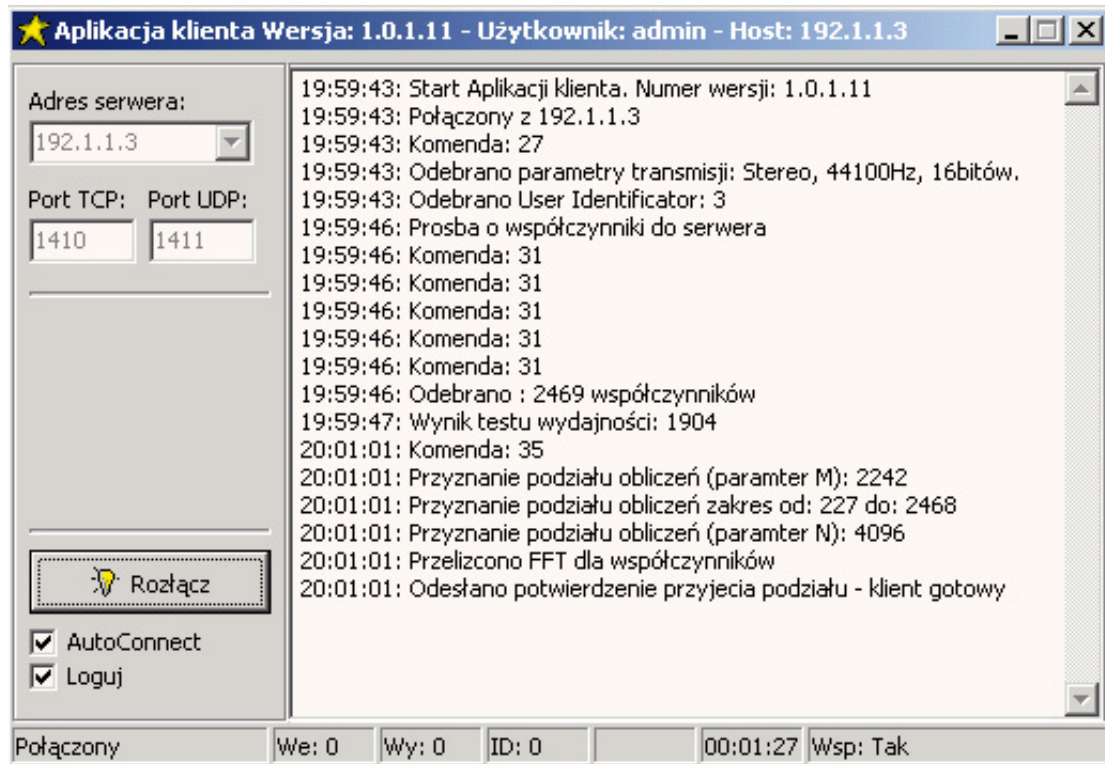
Funkcja kończy pobieranie próbek audio z urządzenia wejściowego i stara się odtworzyć wszystkie sygnały, jakie jeszcze dopływają z zewnątrz. Istnieje zatem pewne skończone opóźnienie w działaniu tej metody.

8.1.8. Uruchomienie / zakończenie pracy serwera

Menu → Ustawienia → Zakończenie nagrywania

W trakcie „życia” aplikacji jest możliwe wielokrotne uruchamianie i zamykanie pracy samego serwera TCP. Aplikacja dalej jest uruchomiona, a zatem wszystkie wcześniejsze ustawienia są aktywne.

8.2. Obsługa aplikacji klienta



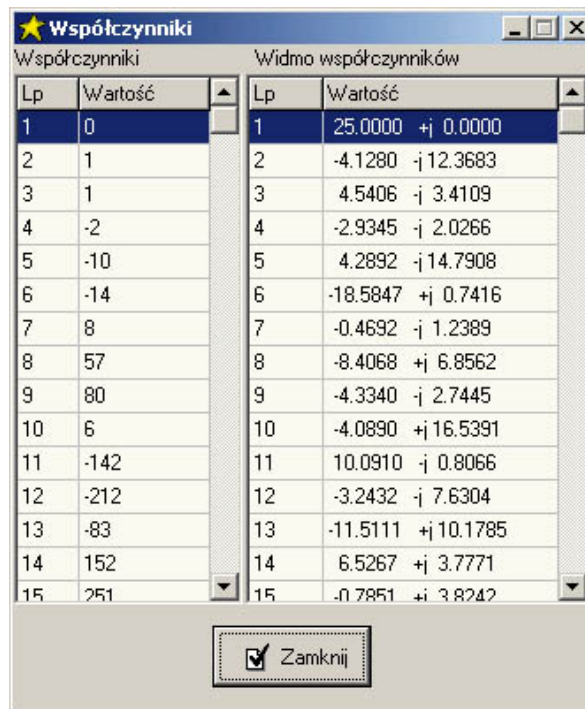
Rys. 8.9. Ekran aplikacji klienta

Aplikacja klienta na podstawie rozgłaszanego sygnału „echo” jest w stanie rozpoznać lokalizację serwera, zatem ręczne ustawianie adresu serwera jest w normalnych warunkach zbędne, możliwość jego ustalenia jednak pozostawiono. Użytkownik może modyfikować numery portów używanych przez aplikację. Klient posiada funkcję „Auto-connect”, która powoduje automatyczne wznawianie połączenia (np. po czasie 10 sekund). Istnieje możliwość zablokowania tej funkcji a także dokonywanie połączenia metodą ręczną. Na dolnym pasku aplikacji widoczne są podstawowe informacje o przebiegu transmisji i stanie połączenia. Na ekranie logującym widoczne są wszelkie komunikaty wymiany informacji z zarządcą (możliwość wyłączenia jako opcja).

Ponieważ liczba aplikacji jest dość duża postawiono jako główny cel jak największe zautomatyzowanie czynności wynikającej z jej pracy, w dużym stopniu odciażającym użytkownika od obsługi aplikacji.

8.2.1. Dodatkowe funkcjonalności

Klient posiada dodatkowe funkcjonalności przydające się raczej w fazie eksperymentalnej, dlatego też zostaną one jedynie zasygnalizowane. Istnieje możliwość zobaczenia jakie współczynniki dotarły do klienta, a także zobaczenia wartości widma obliczonego dla przydzielonej części zadania. W tym celu na ekranie aplikacji należy wcisnąć klawisz F7.



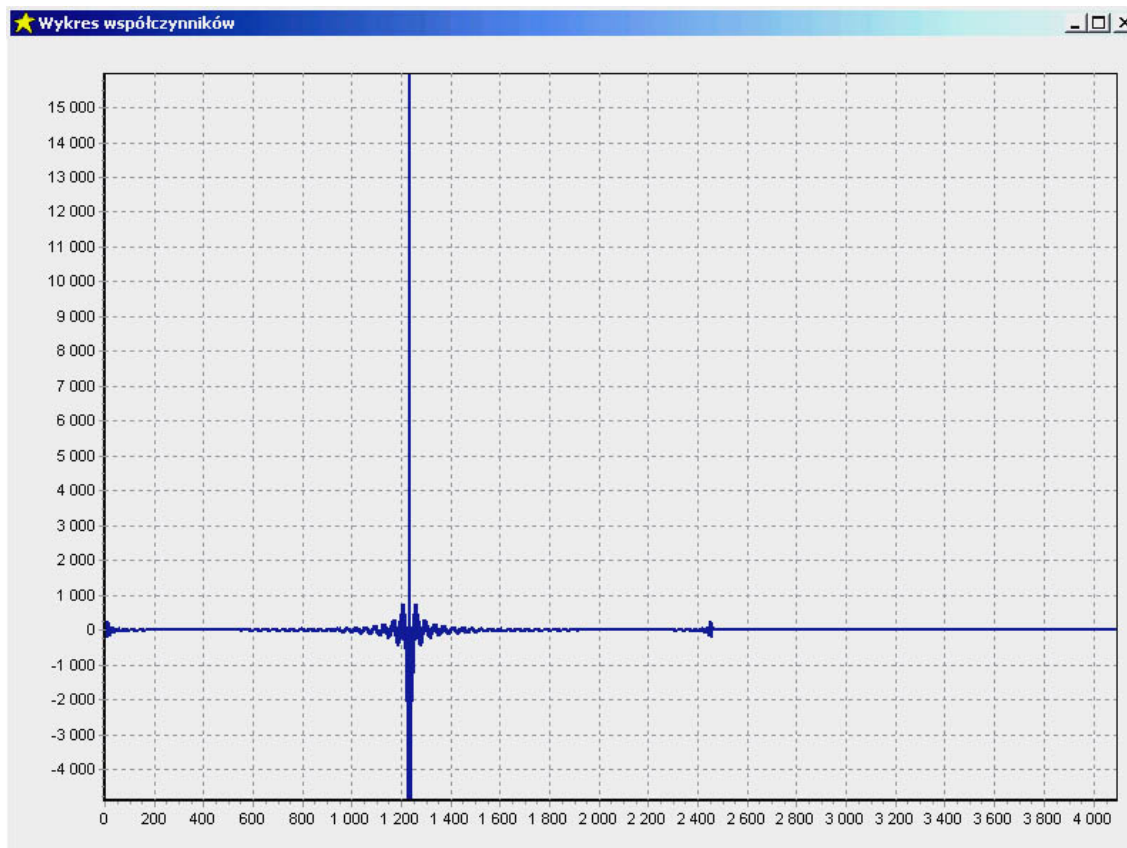
The screenshot shows a window titled 'Współczynniki' with two sub-tables. The left table, 'Współczynniki', lists 15 coefficients (Lp 1-15) with their values (Wartość). The right table, 'Widmo współczynników', lists the same 15 coefficients with their complex frequency spectrum values (Wartość).

Współczynniki		Widmo współczynników	
Lp	Wartość	Lp	Wartość
1	0	1	25.0000 +j 0.0000
2	1	2	-4.1280 -j 12.3683
3	1	3	4.5406 -j 3.4109
4	-2	4	-2.9345 -j 2.0266
5	-10	5	4.2892 -j 14.7908
6	-14	6	-18.5847 +j 0.7416
7	8	7	-0.4692 -j 1.2389
8	57	8	-8.4068 +j 6.8562
9	80	9	-4.3340 -j 2.7445
10	6	10	-4.0890 +j 16.5391
11	-142	11	10.0910 -j 0.8066
12	-212	12	-3.2432 -j 7.6304
13	-83	13	-11.5111 +j 10.1785
14	152	14	6.5267 +j 3.7771
15	251	15	-0.7851 +j 3.8242

At the bottom of the window is a button labeled 'Zamknij' with a checked checkbox.

Rys. 8.10 Ekran ze współczynnikami odebranymi od serwera, oraz obliczone widmo częstotliwości wyznaczonej części współczynników

Wartości liczbowe ułożone na liście są jednak mało obrazowe, lepszym sposobem na zobrazowanie wartości współczynników, obliczonego widma jest wykres graficzny. W tym celu na ekranie aplikacji należy wcisnąć klawisz F8. Oprócz wymienionych funkcjonalności istnieje możliwość wykreślenia charakterystyki amplitudowo-częstotliwościowej filtra, a także jego przesunięcia fazowego.



Rys. 8.11 Przykładowy wykres przebiegu odpowiedzi impulsowej załadowanej przez klienta

9. Wnioski i propozycje rozwoju projektu

Opracowany system może być wykorzystywany do przetwarzania sygnałów akustycznych w czasie rzeczywistym. Przy małych długościach odpowiedzi impulsowej pojedynczy komputer może pełnić rolę serwera jak i klienta pozwalając na proste operacje związane ze splotem sygnałów. Dla dłuższych odpowiedzi impulsowych konieczne jest już przetwarzanie sygnałów podzielone na wiele komputerów.

W projekcie opracowano wiele algorytmów dotyczących cyfrowego przetwarzania sygnałów, m.in. algorytmy FFT radix-2 zarówno DIT jak i DIF oraz algorytm związany z metodą Overlap-save. Na ich podstawie można w przyszłości zaadoptować pozostałe algorytmy FFT takiej jak radix-4 i radix-8 a także split-radix.

W projekcie został przeanalizowany od samego początku problem splatania sygnałów akustycznych. Zrealizowano także dosyć zaawansowaną komunikację sieciową na, której to podstawie można zbudować inny system realizujący przetwarzanie rozproszone niekoniecznie związane z problem przetwarzania sygnałów.

Istotną wadą obecnego rozwiązania jest czas opóźnienia z jakim dociera odpowiedź zwrotna. Wynika ona głównie z tego, że komputer liczący długą transformatę np. 64k punktową musi czekać około 1,5 sekundy (44kHz) aż “uzbiera” wystarczającą ilość próbek do policzenia widma. Dopiero po obliczeniu całości następuje wysłanie całego bloku próbek. Stosowanie większej ilości komputerów liczących krótszą transformatę może znacznie poprawić parametr związany z tym opóźnieniem.

W wyniku doświadczalnych transmisji zauważono, iż najbardziej korzystnym przypadkiem jest konfiguracja gdy moce obliczeniowe wszystkich komputerów są przybliżone, odpowiedzi od klientów przychodzą wówczas równocześnie minimalizując ilość próbek czekających na wysłanie do wyjściowego urządzenia audio.

Oczywistym jest też fakt, iż konfiguracja mniejszej ilości “mocniejszych” obliczeniowo komputerów jest bardziej korzystna od wielu mało wydajnych stanowisk obliczeniowych. Pomimo podobnej sumarycznej mocy obliczeniowej, w tym pierwszym przypadku zredukowany jest czas na obsługę klienta (obsługa buforów, składanie próbek przychodzących od klientów).

Propozycje rozwoju projektu:

- W obecnej chwili projekt działa w warunkach “idealnych”. W jedynie bardzo elementarny sposób zaadaptowano wykrywanie mechanizmów powodujących zakłócenia pracy systemu. W mechanizmach zarówno serwera i klienta są zastosowane odpowiednie procedury zachowawcze jednakże nie zrealizowano algorytmów wykrywających np. uszkodzenie któregoś z komputerów, uszkodzenie łącza.
- Zaimplementowanie wydajniejszych algorytmów FFT .
- Stworzenie modułu pozwalającego na pracę klientów w sieci Internet.
- Wydajny komputer może być podzielony na kilka procesów, które wykonują krótsze transformaty, klient częściej pobiera z bufora próbki i częściej je wysyła. Liczenie samej transformaty jest wielokrotnie krótsze.

10. Literatura

- [1] Tomasz. P. Zieliński Od teorii do cyfrowego przetwarzania sygnałów AGH 2002
- [2] Steven W. Smith The Scientist and Engineer's Guide to Digital Signal Processing - California Technical Publishing San Diego, California 1999 www.DSPguide.com
- [3] J.Izydorczyk, G.Płonka, G.Tyma Teoria sygnałów – wstęp Helion 1999
- [4] Andrew Simmonds Wprowadzenie do transmisji danych Wydawnictwo Komunikacji i Łączności 1999
- [5] Tim Parker, Mark Sportack TCP/IP Księga eksperta Helion 2000
- [6] A.Rodriguez, J.Gatrell, J.Katras, R.Peschke TCP/IP Tutorial and Technical Overview www.ibm.com/redbooks
- [7] A. Jones, J. Ohlund Programowanie sieciowe Microsoft Windows Microsoft Press 2000
- [8].G.Coulouris, J.Dollimore, T.Kindbeg Systemy rozproszone podstawy i projektowanie Wydawnictwo Naukowe-Techniczne Warszawa1998
- [9] Bruce Eckel Thinking in Java Helion 2001
- [10] S.Teixeira, X.Pacheco Delphi 4 Vademecum profesjonalisty Helion 1999
- [11] S.Teixeira, X.Pacheco Borland® Delphi™ 6 Developer's Guide SAMS 2002
- [12] R. Lischner Delphi Almanach O'REILLY 2002
- [13] R.G. Lyons Wprowadzenie do cyfrowego przetwarzania sygnałów Wydawnictwo Komunikacji i Łączności 2003
- [14] TMS320C54x DSP Library Programmer's Reference SPRU565 Texas Instruments 2001

Dodatek A. Elementy teorii dźwięku

Fale dźwiękowe są podłużnymi falami mechanicznymi. Zakres częstotliwości dla fal podłużnych jest bardzo szeroki, przy czym falami dźwiękowymi nazywamy fale o takich częstotliwościach, które wywołują wrażenie słyszenia podczas oddziaływania na ucho ludzkie i mózg. Zakres ten rozciąga się od około 20 do 20 kHz i nazywa się zakresem słyszalnym.

Fale słyszalne powstają w wyniku drgań strun, drgań słupów powietrza, lub drgań różnych płyt i membran. Wszystkie te elementy podczas drgania na przemian zagęszczają i rozrzedzają otaczające powietrze. Ważnym zagadnieniem jest inne odczuwanie natężenia dźwięku o różnych częstotliwościach. Najlepiej słyszalne są dźwięki z zakresu od 1000 do 2000 Hz. Inne częstotliwości dźwięku słabiej oddziałują na słuch ludzki nawet, gdy ich natężenie jest identyczne. Nie tylko w muzyce, lecz również w fizyce i matematyce zdefiniowane zostało pojęcie dźwięku podstawowego, nazywanego tonem prostym. Dźwięk taki można wydobyć, wykorzystując do tego celu stroik instrumentów muzycznych - kamerton. W naukach przyrodniczych tonem prostym bądź dźwiękową falą prostą nazywana jest funkcja o postaci:

$$f(t) = A \sin(\omega t + j)$$

gdzie: A - amplituda sygnału; ω - częstotliwość; j - przesunięcie (faza) sygnału.

Okazuje się, że dźwięk większości instrumentów muzycznych składa się z kilku lub, co najwyżej kilkunastu tonów prostych o różnym natężeniu i częstotliwości będącej wielokrotnością tonu podstawowego. Barwa dźwięku zależy od natężenia występujących w nim tonów prostych lub też pewnych pasm częstotliwości. Ogólnie można powiedzieć, że jeżeli w utworze muzycznym brakuje basów, to składowe tonowe o niskich częstotliwościach wchodzące w skład dźwięku mają niską amplitudę (małe natężenie). Barwa instrumentów muzycznych (a mówiąc dokładniej: natężenie składowych harmoniczných) związana jest z konkretnym instrumentem, dzięki czemu potrafimy łatwo odróżnić nutę graną na fortepianie od tej samej nuty wydobytej z gitary. W nazewnictwie przyjęto konwencję określającą tony o niskich częstotliwościach jako basy, o wysokich - jako sopran. Pomiędzy nimi występują zaś tony średnie.

Dodatek B. Karta muzyczna i próbkowanie dźwięku

Najważniejszą częścią składową, która musi występować w każdej karcie dźwiękowej, jest przetwornik A/D-D/A (analog/digital-digital/analog), zmieniający sygnał analogowy w cyfrowy i odwrotnie, odpowiedzialny za nagrywanie i odtwarzanie plików WAV. Proces nagrywania nazywany jest samplowaniem, co na język polski można przetłumaczyć jako "próbkowanie". Poziom (głośność) sygnału wejściowego, pochodzącego np. z mikrofonu lub wejścia LINE IN jest mierzony w określonych odstępach czasu, zaś wynik pomiaru zapisywany w pliku WAV. Plik ten zawiera dokładne dane, na podstawie, których można zrekonstruować dźwięk przy odtwarzaniu. Znajdują się w nim również informacje o parametrach nagrania, mających wpływ na jakość dźwięku i zapotrzebowanie na wolne miejsce na dysku. Są to:

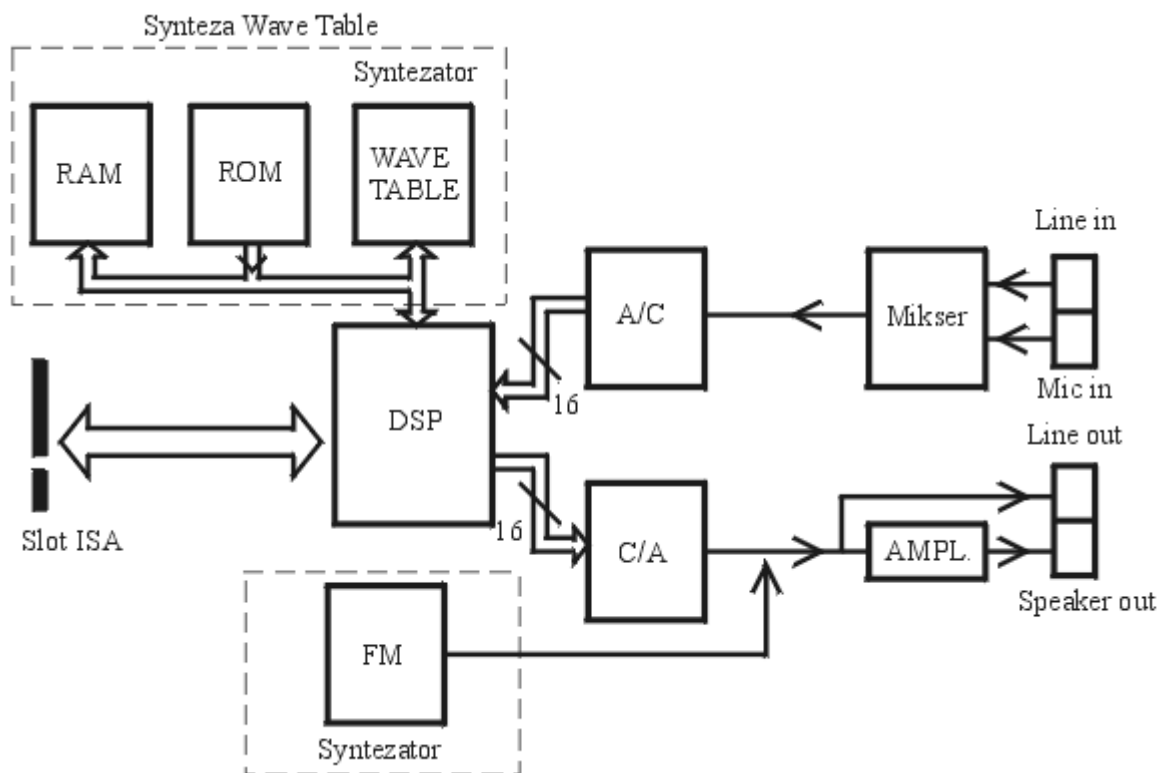
Rozdzielczość: określa, czy wartości pomiarowe zapisywane w pliku WAV mają zajmować 8 czy 16 bitów. Zaletą nagrań 8-bitowych jest ograniczona wielkość pliku WAV, wadą natomiast - dużo gorsza jakość dźwięku. W jednym bajcie można zapisać wartości od 0 do 255, natomiast w dwóch bajtach wartości od 0 do 65535.

Częstotliwość próbkowania: odstępów czasowe, w których dokonywany jest pomiar poziomu sygnału wejściowego. Częstotliwość próbkowania podawana jest w kilohercach (kHz), 1 kHz odpowiada 1000 pomiarów na sekundę. Najczęściej spotyka się karty próbkujące z częstotliwościami 8, 11, 22, 44,1 lub też 48 kHz. Częstotliwość próbkowania karty dźwiękowej ma bezpośredni wpływ na jakość dźwięku, a przede wszystkim na odtwarzanie tonów wysokich. W praktyce oznacza to, że najwyższy ton, jaki można nagrać odpowiada połowie wartości częstotliwości próbkowania; w przypadku karty posiadającej częstotliwość 44,1 kHz będzie to ok. 22 000 Hz. Ucho ludzkie potrafi odbierać dźwięki o częstotliwości do ok. 17 000 Hz. Zastosowanie karty o maksymalnej częstotliwości próbkowania 22 kHz oznacza już wyraźne obcięcie pasma częstotliwości, gdyż będzie ona w stanie nagrać tylko dźwięki o częstotliwości do 11 000 Hz.

Liczba kanałów (mono czy stereo): w przypadku próbkowania stereo pomiary dla prawego i lewego kanału są przeprowadzane i zapisywane oddzielnie. Oznacza to, że nagrania stereo wymagają dwukrotnie więcej miejsca na dysku niż plik mono.

Pomijając parametry próbkowania, dobre karty dźwiękowe różnią się znacznie od tych

gorszych jakością nagrywania i odtwarzania plików WAV. Z reguły nie jest to wina odpowiedzialnego za jakość dźwięku przetwornika A/D-D/A, lecz wzmacniacza niskich częstotliwości.



Dodatek C. Dźwięk cyfrowy

Głównym celem cyfrowej technologii audio jest uzyskanie takiej jakości dźwięku, aby brzmienie muzyki (na przykład zapisanej na płycie CD) było jak najbardziej zbliżone do swej wersji oryginalnej. Osiągnięcie to jest niemożliwe do spełnienia, ale jednakże, jeśli spełnione są pewne warunki, różnicy między oryginałem a nagraniem po prostu nie słychać. Jedynie zawodowi muzycy potrafią czasami odróżnić nagranie cyfrowe od rzeczywistych dźwięków. Wnioski na temat cyfrowego dźwięku nasuwają się same - jeśli chcemy uzyskać nagranie o bardzo dobrej jakości, należy zrobić wszystko, aby zniekształcenia powstające podczas obróbki oryginalnego dźwięku były tak małe, żeby nie było ich słychać.

Im większa rozdzielczość próbkowania dźwięku, tym lepszy odstęp sygnału od szumu kwantowania. Przy rozdzielczości 16 bitów na próbkę teoretyczny odstęp sygnału od szumu wynosi około -96 dB, natomiast przy 8 bitach na próbkę - około -54 dB.

Zniekształcenia dźwięku

Pojęcia szumów i zniekształceń są bardzo często spotykane w opisach różnego rodzaju sprzętu audio (karty muzyczne, głośniki) oraz plików (na przykład banków brzmień WaveTable). Parametry te opisują, w jaki sposób dźwięk przetworzony przez pewne urządzenie (algorytm, program) różni się od oryginalnego sygnału. Wszelkie zmiany sygnału przetwarzanego w odniesieniu do sygnału oryginalnego, które nie są przez nas zamierzone, nazywamy zniekształceniami dźwięku. Wyróżniono kilka klas zniekształceń, z których najważniejsze to szумы oraz zniekształcenia harmoniczne. Szumami określane są sygnały przypadkowe lub trudne do zdefiniowania. Zniekształceniami harmonicznymi (lub po prostu zniekształceniami) nazywane są dodatkowe sygnały harmoniczne (na przykład przebiegi sinusoidalne), pojawiające się w oryginalnym dźwięku po jego przetworzeniu (próbkowaniu czy też odtwarzaniu).

Zniekształcenia dźwięku podczas próbkowania

Uzyskanie zadowalających efektów podczas konwersji sygnału analogowego na postać cyfrową bądź odwrotnie może się okazać dla niektórych osób zajęciem ciekawym i zajmującym, dla innych - spędzającym sen z powiek. Dysponując gotowym materiałem muzycznym (muzyka na płycie CD, komercyjny bank brzmień MIDI), mamy pewność, że

jakość dźwięku jest wysoka. Podczas tworzenia własnych kompozycji muzycznych czy banku brzmień MIDI zachodzi jednak często konieczność przetworzenia sygnału analogowego (na przykład głosu) na postać cyfrową, czyli utworzenia tak zwanej próbki dźwięku. Proces takiej konwersji, nazywany potocznie próbkowaniem, składa się z dwóch etapów: pobierania (sprawdzania) stanu amplitudy sygnału analogowego oraz kwantyzacji - przypisania pobranej próbce dźwięku pewnej liczby, utożsamianej z amplitudą sygnału w postaci cyfrowej. W trakcie konwersji sygnału analogowego na postać cyfrową powstają pewne zniekształcenia sygnału. Przyczyny pogorszenia dźwięku podczas próbkowania:

- Kwantyzacja i szum kwantowania. Proces kwantowania dźwięku polega na przypisaniu każdej próbce sygnału pewnej liczby z ustalonego z góry zakresu - w zależności od rozdzielczości przetwornika analogowo-cyfrowego (w skrócie AC). Jeżeli dźwięk jest próbkowany z rozdzielczością 16 bitów, oznacza to, że amplituda dźwięku podzielona została na 65536 przedziałów i każdemu z tych przedziałów przypisano pewną wartość. Konwencja przypisywania liczb próbce sygnału bywa różna, najczęściej spotykaną metodą jest przyporządkowywanie liczb całkowitych, co w przedstawionym tu przykładzie oznacza, że każdemu z przedziałów zostanie przypisana wartość od -32768 do 32767. Należy zauważyć, że próbkowanie i kwantyzacja dźwięku prowadzą do utraty części informacji o oryginalnym sygnale. Mówiąc inaczej, nie jesteśmy w stanie z przetworzonego dźwięku uzyskać z powrotem oryginalu.

Z procesem kwantyzacji dźwięku ściśle związane jest pojęcie szumu kwantyzacji, określające, jak duża część informacji została zgubiona w procesie konwersji sygnału analogowego na postać cyfrową. Ponieważ charakter zniekształceń powstających w procesie kwantyzacji ma postać szumów, noszą one nazwę szumu kwantyzacji. Nie należy natomiast mylić szumu kwantyzacji z parametrem nazywanym odstępem sygnału od szumu - SNR (ang. *Signal to Noise Ratio*). Szum kwantyzacji to teoretycznie najlepszy odstęp sygnału od szumu w zdigitalizowanym dźwięku, natomiast SNR określa całkowity szum występujący podczas pracy danego urządzenia audio. Bardzo dobry sprzęt ma współczynnik SNR równy lub zbliżony do szumu kwantyzacji, co oznacza, że szumy przedostające się do sygnału użytecznego są bardzo małe - można je zaniedbać - lub też w niewielkim stopniu wpływają na zaszumienie oryginalnego dźwięku.

- Szum układów elektrycznych. Największą bolączką producentów podzespołów audio jest fakt, że każdy układ elektroniczny wprowadza do oryginalnego sygnału pewne zniekształcenia. Ponieważ są one przeważnie losowe, nazywa się je szumami układów elektronicznych. Zniekształcenia te zależą od wielu czynników związanych szczególnie z budową i sposobem działania konkretnego układu scalonego. Choć nie da się zupełnie wykluczyć szumów powodowanych przez układy elektroniczne, to jednak są sposoby, aby zniekształcenia te były odpowiednio małe. Trzeba się jednak wówczas liczyć ze znacznym wzrostem kosztów produkcji chipów.

Przetwornik AC/DC, będący elementem układu CODEC, jest odpowiedzialny za konwersję analogowo-cyfrową i cyfrowo-analogową dźwięku oraz jednoznacznie ustala szum kwantowania na pewnym poziomie (na przykład około -96 dB dla 16-bitowego przetwornika AC/DC). Wynika stąd, że oryginalny sygnał po przejściu przez sam przetwornik AC/DC będzie na pewno zawierał szumy na poziomie szumu kwantowania.

Należy zauważyć, że jeżeli nawet inne elementy układu CODEC będą się charakteryzowały bardzo dobrymi parametrami (na przykład SNR wynosić będzie 120 dB), to ogólnie najniższy możliwy do uzyskania współczynnik odstepu sygnału od szumu będzie wynosił tyle, ile szum kwantowania. Jeżeli w danym urządzeniu istnieją elementy ustalające pewien poziom szumów, wówczas inne podzespoły tego urządzenia na pewno nie poprawią tego parametru (mogą jedynie go pogorszyć). W przypadku kart muzycznych najlepszy możliwy poziom SNR jest ustalony przez szum kwantowania. Także dobór podzespołów do produkcji kart muzycznych jest w praktyce zdeterminowany przez rozdzielczość przetwornika AC/DC.

Pasmo przenoszenia sygnału. W przypadku kart dźwiękowych bardzo ważnym parametrem jest pasmo przenoszenia sygnału - zarówno przy konwersji sygnału analogowego na postać cyfrową, jak i odwrotnie. Aby uzyskać bardzo dobrą jakość dźwięku przy tworzeniu muzyki, dźwięk przechodzący przez kartę dźwiękową powinien mieć płaską charakterystykę częstotliwościową (małe różnice pomiędzy sygnałem oryginalnym a przetworzonym w każdym z zakresów częstotliwości). Można to wyjaśnić w bardzo prosty sposób. Załóżmy, że odtwarzając z komputera muzykę, słyszymy dużo basów i mało sopranów. Jednak w oryginalnym dźwięku (w pliku muzycznym) jest na

odwrot: dużo sopranów i mało basów. Oznacza to, że podbite zostały basy i osłabione soprały - w takim przypadku nie można mówić o płaskiej charakterystyce sygnału.

Badanie pasma przenoszenia sygnału w kartach muzycznych polega na przepuszczaniu przez nie kolejnych tonów prostych o zadanej częstotliwości i sprawdzaniu, czy amplitudy sygnału wejściowego i wyjściowego są sobie równe. Jeżeli różnią się one od siebie nieznacznie, uważa się, że karta dźwiękowa przenosi sygnał o danej częstotliwości, natomiast, jeżeli różnica jest duża (za taką przyjmuje się zazwyczaj 3 dB), to danej częstotliwości nie wlicza się do pasma przenoszenia sygnału. Płyne stąd wniosek, że warto sprawdzić, czy nasza karta muzyczna dobrze przenosi wszystkie częstotliwości sygnału. Przedstawiony skrótowy opis nie wyczerpuje oczywiście wszystkich aspektów tego problemu. Testy kart dźwiękowych pod kątem pasma przenoszenia sygnału wykonywane są m.in. przez wydawnictwa prasy komputerowej.

Nierównomierność pobierania próbek w procesie samplingu. W procesie próbkowania dźwięku pobierana jest równomiernie amplituda sygnału docierającego do przetwornika analogowo-cyfrowego i w ten sposób sygnał analogowy zamieniany jest na postać cyfrową. Miara określającą, jak często pobierane są próbki sygnału podczas samplingu dźwięku, jest częstotliwość próbkowania określona w hercach - na przykład częstotliwość próbkowania 44,1 kHz oznacza, że w ciągu jednej sekundy pobranych zostanie 44 100 próbek dźwięku. Co się jednak stanie, gdy podczas próbkowania układ odpowiadający za równomierne odmierzanie czasu pomiędzy pobraniem kolejnych próbek dźwięku będzie niedokładny? Będzie to oznaczać, że sygnał w procesie konwersji analogowo-cyfrowej będzie próbkowany, co pewien czas, lecz nie będzie dokładnie taki, jaki jest ustalony przez częstotliwość próbkowania, a jedynie z pewnym przybliżeniem. Okazuje się, że nawet bardzo małe odchylenia od ustalonego cyklu taktowania przetwornika AC/DC powodować mogą znaczne zniekształcenie dźwięku, szczególnie w przypadku wysokich częstotliwości sygnału. Producenci kart muzycznych rzadko podają informacje na temat parametrów opisujących dokładność taktowania przetwornika AC/DC, jednak możemy to pośrednio wywnioskować ze współczynnika THD (*Total Harmonic Distortion*), określającego całkowite zniekształcenia harmoniczne występujące w urządzeniu audio.

Nie bez wpływu na jakość dźwięku jest poddawanie próbek dźwiękowych różnego rodzaju operacjom, takim jak: filtracja, usuwanie szumów, dodawanie efektów dźwiękowych (pogłos)

Dźwięk jest zapisany w postaci cyfrowej, np. 16-bitowy, i jest ciągiem pewnych liczb całkowitych z przedziału od -32768 do 32767, określających amplitudę sygnału w chwilach pobierania jego próbki w procesie samplingu. Jeżeli teraz zechcemy zmniejszyć głośność dźwięku o połowę to w przypadku próbki nieparzystej będziemy musieli zastosować zaokrąglenie i w rezultacie np. przy odwrotnym procesie, czyli zwiększeniu amplitudy może się zdarzyć, iż wartość próbki nie przyjmie już wcześniejszej wartości i będzie obciążona błędem zaokrąglenia. Pewnym sposobem rozwiązania tego problemu jest takie przekształcenie pliku dźwiękowego, aby próbki sygnału były opisane przez liczby 24- lub 32-bitowe (zamiast 16 bitowych), oraz przeprowadzanie obliczeń w taki sposób, aby minimalizować błąd zaokrąglenia (na przykład wynik obliczeń przechowywać jako liczbę 32-bitową). Nie spodziewajmy się jednak całkowitego wyeliminowania błędów zaokrągleń, nawet w przypadku najlepszych programów muzycznych. Płyne stąd prosty wniosek, że podczas pracy z materiałem muzycznym nie należy wykonywać zbędnych operacji na dźwięku zapisanym w postaci cyfrowej.

Wpływ czynników zewnętrznych Najczęściej mamy do czynienia z przenikaniem do torów audio fal elektromagnetycznych wytwarzanych przez sieć energetyczną (przódźwięk sieci elektrycznej), co objawia się obecnością w dźwięku sygnału o częstotliwości 50 Hz (specyficzne buczenie). Efekt ten można bardzo łatwo sprawdzić - wystarczy podłączyć mikrofon do wzmacniacza długim, nie ekranowanym kablem. Na przenikanie fal elektromagnetycznych najbardziej narażone są tory audio przenoszące dźwięk analogowy, czyli ścieżki, po których płynie dźwięk w postaci odpowiednio zmieniającego się napięcia prądu elektrycznego. W takim przypadku bezpośrednio do sygnału użytecznego (dźwięku) dodawany jest sygnał pasożytniczy (zniekształcenia). Dużo odporniejsze na zniekształcenia powodowane przenikaniem fal elektromagnetycznych są cyfrowe tory audio.

Dodatek D RFC

Podstawowe informacje

RFC1360	"IAB Official Protocol Standards," Postel, J.B.; 1992
RFC1340	"Assigned Numbers," Reynolds, J.K.; Postel, J.B.; 1992
RFC1208	"Glossary of Networking Terms," Jacobsen, O.J.; Lynch, D.C.; 1991
RFC1180	"TCP/IP Tutorial," Socolofsky, T.J.; Kale, C.J.; 1991
RFC1178	"Choosing a Name for Your Computer," Libes, D.; 1990
RFC1175	"FYI on Where to Start: A Bibliography of Inter-networking Information," Bowers, K.L.; LaQuey, T.L.; Reynolds, J.K.; Reubicek, K.; Stahl, M.K.; Yuan, A.; 1990
RFC1173	"Responsibilities of Host and Network Managers: A Summary of the Oral Tradition of the Internet," vanBokkelen, J.; 1990
RFC1166	"Internet Numbers," Kirkpatrick, S.; Stahl, M.K.; Recker, M.; 1990
RFC1127	"Perspective on the Host Requirements RFCs," Braden, R.T.; 1989
RFC1123	"Requirements for Internet Hosts—Application and Support," Braden, R.T., ed; 1989
RFC1122	"Requirements for Internet Hosts—Communication Layers," Braden, R.T., ed; 1989
RFC1118	"Hitchhiker's Guide to the Internet," Krol, E.; 1989
RFC1011	"Official Internet Protocol," Reynolds, J.R.; Postel, J.B.; 1987
RFC1009	"Requirements for Internet Gateways," Braden, R.T.; Postel, J.B.; 1987
RFC980	"Protocol Document Order Information," Jacobsen, O.J.; Postel, J.B.; 1986

TCP i UDP

RFC1072	"TCP Extensions for Long-Delay Paths," Jacobson, V.; Braden, R.T.; 1988
RFC896	"Congestion Control in IP/TCP Internetworks," Nagle, J.; 1984
RFC879	"TCP Maximum Segment Size and Related Topics," Postel, J.B.; 1983
RFC813	"Window and Acknowledgment Strategy in TCP," Clark, D.D.; 1982
RFC793	"Transmission Control Protocol," Postel, J.B.; 1981
RFC768	"User Datagram Protocol," Postel, J.B.; 1980

IP i ICMP

RFC1219	"On the Assignment of Subnet Numbers," Tsuchiya, P.F.; 1991
RFC1112	"Host Extensions for IP Multicasting," Deering, S.E.; 1989
RFC1088	"Standard for the Transmission of IP Datagrams over NetBIOS Networks," McLaughlin, L.J.; 1989
RFC950	"Internet Standard Subnetting Procedure," Mogul, J.C.; Postel, J.B.; 1985
RFC932	"Subnetwork Addressing Schema," Clark, D.D.; 1985
RFC922	"Broadcasting Internet Datagrams in the Presence of Subnets," Mogul, J.C.; 1984
RFC919	"Broadcasting Internet Datagrams," Mogul, J.C.; 1984
RFC886	"Proposed Standard for Message Header Munging," Rose, M.T.; 1983
RFC815	"IP Datagram Reassembly Algorithms," Clark, D.D.; 1982
RFC814	"Names, Addresses, Ports, and Routes," Clark, D.D.; 1982
RFC792	"Internet Control Message Protocol," Postel, J.B.; 1981
RFC791	"Internet Protocol," Postel, J.B.; 1981
RFC781	"Specification of the Internet Protocol (IP) Timestamp Option," Su, Z.; 1981

Protokoły "routingu"

RFC1267	"A Border Gateway Protocol 3 (BGP-3)," Lougheed, K.; Rekhter, Y.; 1991
RFC1247	"OSPF version 2," Moy, J.; 1991
RFC1222	"Advancing the NSFNET Routing Architecture," Braun, H.W.; Rekhter, Y.; 1991
RFC1195	"Use of OSI IS-IS for Routing in TCP/IP and Dual Environments," Callon, R.W.; 1990
RFC1164	"Application of the Border Gateway Protocol in the Internet," Honig, J.C.; Katz, D.; Mathis, M.; Rekhter, Y.; Yu, J.Y.; 1990
RFC1163	"Border Gateway Protocol (BGP)," Lougheed, K.; Rekhter, Y.; 1990
RFC1136	"Administrative Domains and Routing Domains: A Model for Routing in the Internet," Hares, S.; Katz, D.; 1989
RFC1074	"NSFNET Backbone SPF-Based Interior Gateway Protocol," Rekhter, J.; 1988
RFC1058	"Routing Information Protocol," Hedrick, C.L.; 1988
RFC911	"EGP Gateway under Berkeley UNIX 4.2," Kirton, P.; 1984
RFC904	"Exterior Gateway Protocol Formal Specification," Mills, D.L.; 1984
RFC888	"STUB Exterior Gateway Protocol," Seamonson, L.; Rosen, E.C.; 1984
RFC827	"Exterior Gateway Protocol (EGP)," Rosen, E.C.; 1982
RFC823	"DARPA Internet Gateway," Hinden, R.M.; Sheltzer, A.; 1982

Warstwa OSI

RFC1240	"OSI Connectionless Transport Services on Top of UDP: Version 1," Shue, C.; Haggerty, W.; Dobbins, K.; 1991
RFC1237	"Guidelines for OSI NSAP Allocation in the Internet," Gollela, R.; Gardner, E.P.; Callon, R.W.; 1991
RFC1169	"Explaining the Role of GOSIP," Cerf, V.G.; Mills, K.L.; 1990

Dodatek E Przykład pakietu TCP

Ethernet type II	Destination address: 00-50-BA-31-04-33 Source address: 00-E0-98-42-55-3B Type IP
IP header	Version 4 Header length 5 TOS 00 Total length 144 Ident. 13282 Flags 2 Fragment offset 0 TTL 128 Protocol TCP - Transmission Control Checksum 447E Source IP Address 192.1.1.3 Dest. IP Address 192.1.1.1
TCP header	Source port UNKNOWN (0xBE8) Dest. port netbios-ssn (NETBIOS Session Service) Seq. number EE319401 Ack. number DFCB7C30 Header len. 5 Reserved 0 Urgent 0 ACK 1 PSH 1 RST 0 SYN 0 FIN 0 Window size 17516 TCP Checksum 53781 Urgent pointer 0
Undecoded	<pre> 00 00 00 64 FF 53 4D 42 32 00 00 00 00 18 07 C8 00 00 00 00 00 00 00 00 00 00 00 00 00 04 18 BC 05 01 08 C0 59 0F 20 00 00 00 02 00 28 00 00 00 00 00 00 00 00 00 00 00 20 00 44 00 00 00 00 00 01 00 05 00 23 00 00 00 00 EC 03 00 00 00 00 5C 00 4E 00 61 00 75 00 6B 00 61 00 5C 00 54 00 43 00 50 00 49 00 50 00 00 00 </pre> <pre> ...d SMB2....↑.L↑↑↑ 04Ly* ...0.(....D.....0 ..#.g.....\ M.a.u.k.a.\.T.C. P.I.P... </pre>

Ethernet type II	00 50 BA 31 04 33 00 E0 98 42 55 3B 08 00	.P 103.6\$BU;0.
IP header	45 00 00 90 33 E2 40 00 80 06 44 7F C0 01 01 03 C0 01 01 01	E..E300.040Δ400 4000
TCP header	0B E8 00 8B EE 31 94 01 DF CB 7C 30 50 18 44 6C D2 15 00 00	08.0t180T OP101 08..
Undecoded	00 00 00 64 FF 53 4D 42 32 00 00 00 00 18 07 C8 00 00 00 00 00 00 00 00 00 00 00 00 04 18 BC 05 01 08 C0 59 0F 20 00 00 00 02 00 28 00 00 00 00 00 00 00 00 00 00 00 20 00 44 00 00 00 00 00 01 00 05 00 23 00 00 00 00 EC 03 00 00 00 00 5C 00 4E 00 61 00 75 00 6B 00 61 00 5C 00 54 00 43 00 50 00 49 00 50 00 00 00	...d SMB2....↑-L↑↑ 00Ly* ...0.(....D.....0 .0.#....00....\. N.a.u.k.a.\.T.C. P.I.P...

Dodatek F Analiza wydajności użytych algorytmów radix-2

Poniżej zostały przedstawione wyniki testu wydajnościowego, który wykonuje transformatę FFT radix-2 (jeden przebieg). Test wykonywany był stukrotnie a otrzymany wynik w milisekundach odpowiada jednemu przebiegowi transformaty.

Rozmiar FFT [-]	Czas wykonania FFT w zależności od procesora [msec]					
	P120	P200 MMX	PII 266	PIII 500	Celeron 800	P4 2,53
1024	5.71	3.06	1.95	1.00	0.80	0.30
2048	17.73	6.92	4.34	2.20	1.80	0.70
4096	36.16	18.91	10.10	5.11	3.80	1.60
8192	80.42	45.04	22.10	11.02	8.41	3.50
16384	166.54	110.28	47.80	23.98	22.00	7.71
32768	372.00	231.96	102.80	59.98	113.46	17.03
65536	876.16	556.07	249.65	218.82	255.87	43.66

Rozmiar cache'a pierwszego poziomu	8kB	16kB	512kB	256kB (procesor Mobile)	128kB	512kB
---	-----	------	-------	-------------------------------	-------	-------

Analizując czasy wykonania FFT możemy zauważyć pewną zależność. Zgodnie z teorią czas wykonania dwa razy większej transformaty powinien wynosić: $2 \cdot \log_2(N_2) / \log_2(N_1)$, która to wartość jest nieznacznie większa od liczby 2.

Jednak zauważalna jest pewna niezgodność dla wartości $4 \cdot N > \text{rozmiaru cache'a pierwszego poziomu}$ (wartości wytłuszczone), jest to spowodowane tym, że operacje na próbkach są prowadzone na 32 bitowych liczbach całkowitych i dopóki mieszczą się w całości w pamięci podręcznej wykonują się znacznie szybciej niż gdy istnieje potrzeba „sięgania” po próbki do pamięci operacyjnej komputera która jest znacznie wolniejsza w działaniu.

