Zadanie1:

TS/lab3/Zadanie1/zadanie1.py

1. Generowanie i Użycie Tablicy CRC

Program zawiera klasę CRC32, która generuje tablicę CRC (Cykliczny Kod Nadmiarowy) na podstawie wielomianu 0xEDB88320. Tablica ta jest wykorzystywana do obliczania CRC dla danych, co pomaga w detekcji błędów powstałych podczas transmisji.

2. Bit Stuffing i Unstuffing

Funkcje bit_stuffing i unbit_stuffing są używane do dodawania dodatkowego bitu ('0') po każdej sekwencji pięciu jednek w celu zapobiegania nieporozumieniom w interpretacji markerów początku i końca ramki danych, które są reprezentowane przez '01111110'.

3. Ramkowanie Danych

Funkcja frame_data dzieli dane na ramki o określonym rozmiarze (tutaj 100 bitów), dodaje do każdej ramki obliczony CRC, stosuje bit stuffing i dodaje markery początku i końca.

4. Psucie Danych

Funkcja corrupt_frame pozwala na symulację błędów transmisyjnych poprzez losowe zmienianie bitów w ramce z zadanym prawdopodobieństwem.

5. Weryfikacja i Naprawa Ramki

Funkcja verify_frame sprawdza, czy ramka jest poprawna poprzez porównanie obliczonego CRC z CRC zawartym w ramce. Pomaga to w identyfikacji uszkodzonych ramek.

6. Procesowanie Ramki

process_frames przetwarza listę ramek, weryfikując każdą z nich i zbierając te, które są poprawne, oraz licząc te uszkodzone.

7. Zapis i Odczyt Ramki

Program zapisuje i odczytuje ramki do/z plików, co umożliwia analizę transmisji danych.

Scenariusz Użycia

Program tworzy losowe dane binarne, dzieli je na ramki, losowo psuje niektóre z nich, a następnie odczytuje i weryfikuje, zapisując naprawione ramki do nowego pliku. Podsumowuje liczbę poprawnych i uszkodzonych ramek na końcu. Możemy kontrolować procent psucia ramek za pomocą corruption_probability

Zadanie2

TS/lab3/Zadanie2/zadanie2.py

Kod symuluje działanie sieci, w której nadajniki próbują wysłać swoje dane, ale jednocześnie muszą unikać kolizji na wspólnym medium transmisyjnym, korzystając z mechanizmu CSMA/CD (Carrier Sense Multiple Access with Collision Detection). Każda linia stanu sieci w twoim przykładzie pokazuje, jak sygnały z różnych nadajników interferują ze sobą i jak system reaguje na kolizje.

Proces Pracy:

Inicjalizacja: Nadajniki ('A', 'B', 'C') są zainicjalizowane na określonych pozycjach w sieci. Każdy nadajnik ma swój kolor i zakres, w jakim może wysyłać sygnał (propagować).

Transmisja: Nadajniki próbują wysyłać dane. Na początku dodają swoją pozycję do kolejki, po czym odczekują losowy czas. Następnie, gdy są gotowe do transmisji, sprawdzają medium (sieć), czy jest wolne.

Wykrywanie kolizji:

sense_medium: Sprawdza, czy cała sieć jest wolna (tj. zawiera tylko '0').

attempt_to_propagate_signal: Próbuje rozprowadzić sygnał z obecnej pozycji w obu kierunkach, jak to określono w zakresach propagacji. Jeśli na ścieżce propagacji wykryto inny sygnał niż '0', występuje kolizja.

Obsługa kolizji:

Nadajnik, który wykryje kolizję, zwiększa swój licznik kolizji.

Czas odstępu (backoff) po kolizji jest losowo obliczany na podstawie liczby kolizji, zgodnie z algorytmem wykładniczego odstępu binarnego. Czas ten jest proporcjonalny do 2 do potęgi liczby kolizji, ale nie większy niż 2^10

Czyszczenie sygnałów:

Po udanej transmisji lub kolizji nadajnik czyści swój sygnał z sieci, ustawiając zajęte pozycje z powrotem na '0'.

Interpretacja przykładowych logów:

----------XBBBBBBBBB-BBBBBBBBBB----------:

'B' próbuje wysłać sygnał, ale napotyka na kolizję z innym sygnałem, stąd 'X' na ścieżce. Nadajnik ‘A’ chce coś wysłać. Nadajnik 'A' musi cofnąć się (backoff) i spróbować ponownie później.