**Sprawozdanie z Listy 3 (Technologie Sieciowe)**

*Jakub Omieljaniuk (250090)*

Komunikacja między urządzeniami jest realizowana dzięki **mediom transmisyjnym**. Wyróżnia się media przewodowe (**kable:** *skrętka, kabel koncentryczny, światłowód*) oraz bezprzewodowe (**fale elektromagnetyczne:** *radiowe, świetlne*). Moc sygnału, bądź też jego brak, jest interpretowany przez urządzenia dzięki protokołom na poziomie **warstwy łącza danych** modelu ISO/OSI lub **warstwy dostępu do sieci** dla modelu TCP/IP.

Rodzaj medium transmisyjnego i użytego protokołu jest determinowany przez **standard IEEE**, w którym dana sieć została stworzona. Najbardziej popularnymi standardami są IEEE 802.3 dla przewodowej sieci Ethernet oraz IEEE 802.11 wykorzystywany przez sieci Wi-Fi.

Aby nasz komputer mógł przesłać wiadomość „HELLO WORLD!” do innego urządzenia, musi przekonwertować ją do postaci binarnej. To pozwoli na wprowadzenie wiadomości do medium transmisyjnego. W największym uproszczeniu **1** będzie oznaczać **sygnał**, a **0** - **brak sygnału**.

Zamieńmy zatem nasz napis na ciąg liczb korzystając z *tablicy ASCII1*:

H E L L O \_ W O R L D !

72 69 76 76 79 32 87 79 82 76 68 33

***1Tablica ASCII*** *– siedmiobitowy system kodowania znaków. Przyporządkowuje liczbom (z zakresu 0-127) litery alfabetu łacińskiego języka angielskiego, cyfry i inne symbole (np. znaki spacji, nowej linii).*

Następnie zapis liczby w notacji dziesiętnej zamieniamy na notację binarną. Dla uproszczenia nasz komunikat skrócę jedynie do frazy ”HELLO’’:

72 69 76 76 79

1001000 1000101 1001100 1001100 1001111

Podczas przesyłania danych mogą wystąpić różnego rodzaju zakłócenia, które spowodują losowe przekłamania wartości wysyłanych bitów. W tym celu wykorzystuje się **cykliczny kod nadmiarowy** (CRC – Cyclic Redundancy Check) – algorytm obliczający **kontrolną liczbę** na podstawie wartości bitów naszej wiadomości. Dołączany jest on do naszego strumienia, a następnie ponownie obliczany przez odbiorcę. Jeśli wartość jest inna od tej, którą wysłaliśmy, oznacza to, że nasz sygnał został zakłócony.

W przypadku CRC-32 liczba kontrolna ma 32 bity, a otrzymuje się ją poprzez obliczenie reszty z dzielenia naszego ciągu bitów wiadomości przez 33-bitową liczbę:

100000100110000010001110110110111,

która jest reprezentacją\* ustalonego dla CRC-32 wielomianu:

x32 + x26 + x23 + x22 + x16 + x12 + x11 + x10 + x8 + x7 + x5 + x4 + x2 + x + 1

*\* jeśli a = 1 dla axk to wstawiamy 1 na k-tej pozycji lub 0, gdy a=0. W ten sposób otrzymamy 33-bitową liczbę binarną.*

Obliczoną liczbę kontrolną dla naszej wiadomości dołączamy na koniec strumienia:

1001000 1000101 1001100 1001100 1001111 00100011011111000011011011111000

Metoda ta jest bardziej skomplikowana od obliczenia prostej **sumy kontrolnej** (jak np. w numerze PESEL) lub **kontroli** **parzystości**, ale dzięki temu jest mniejsze prawdopodobieństwo, że wiele jednoczesnych błędów nie zmieni wartości liczby kontrolnej. Każdy z tych sposobów nie ma natomiast zastosowania kryptograficznego – jest możliwa modyfikacja przysłanych bitów w taki sposób, aby algorytm CRC jej nie wykrył, zwracając tę samą sumę kontrolną.

Aby usystematyzować przesyłanie danych nasz strumień bitów poddaje się tzw. **ramkowaniu**. Polega on na podzieleniu naszych danych na ramki, obliczeniu CRC i „opakowaniu” każdej z nich **znacznikami** **początku** i **końca**.

W sieci Ethernetowej maksymalna wartość danych dla jednej ramki wynosi 1500 bajtów (**12 000 bitów**). W naszym programie znacznikami będzie ośmio-bitowa liczba: **01111110**. Urządzenia w takiej sieci będą ignorowały strumienie, które nie rozpoczną się powyższym ciągiem bitów, a także będą dokładnie rozpoznawać moment, w którym zakończy się przesyłanie danej ramki. Nasza będzie prezentować się następująco:

**01111110** 1001000 1000101 1001100 1001100 1001111 00100011011111000011011011111000 **01111110**

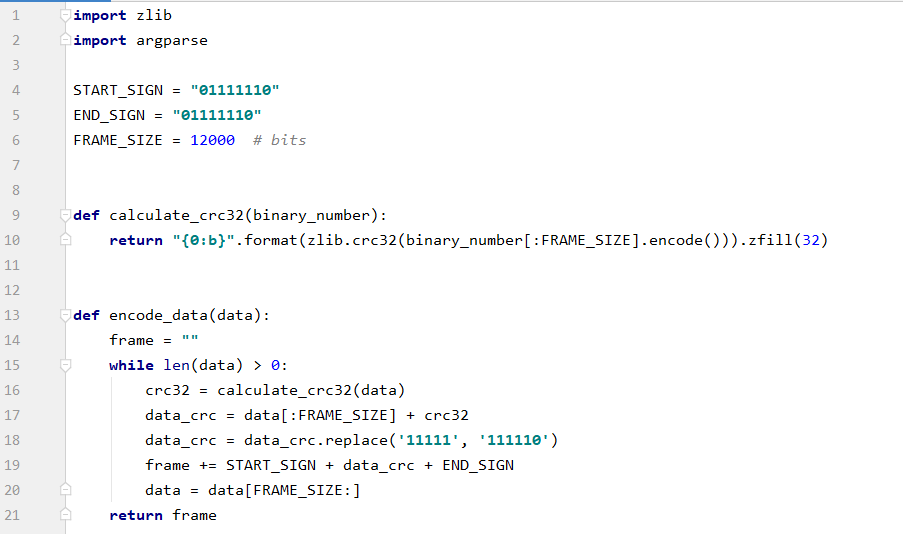
W celu uniknięcia dwuznaczności ciągu 01111110 w przypadku gdyby taki pojawił się w naszej wiadomości lub liczbie kontrolnej, skorzystamy z zasady **rozpychania bitów**. W każdym miejscu gdzie wystąpi po sobie **kolejno pięć jedynek** jako kolejną cyfrę wstawimy **dodatkowe zero**. Operacje tą należy przeprowadzić przed dodaniem znaczników, tak aby jedyną sekwencją sześciu kolejnych jedynek w ramce były znaczniki początku i końca.

Odbiorca takiej wiadomości będzie miał pewność, gdzie zaczyna i kończy się jedna ramka oraz łatwo zdekoduje wiadomość w niej zawartą usuwając nadmiarowe zera. Zastosujmy zatem rozpychanie bitów dla naszej ramki otrzymując jej ostateczny kształt:

**01111110** 1001000 1000101 1001100 1001100 1001111 00100011011111**0**000011011011111**0**000 **01111110**

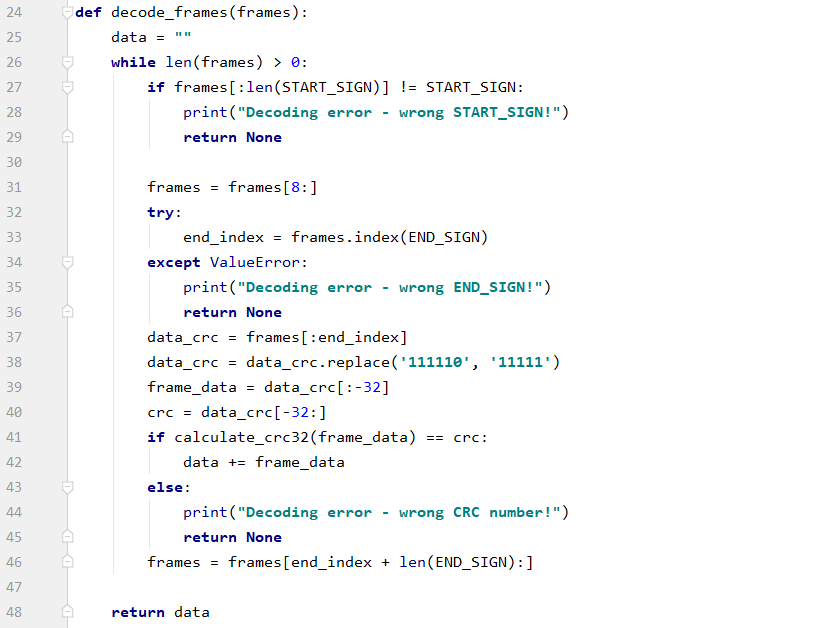
0111111010010001000101100110010011001001111001000110111110000011011011111000001111110

Cały powyższy proces przygotowywania danych do wysyłania zaimplementowałem w języku **Python**:



1. Python: Ramkowanie strumienia bitów

W przypadku odkodowywania ramek musimy zadbać o odpowiednie wyłuskanie danych z ramki   
i sprawdzenie ich zgodności z przekazaną liczbą kontrolną CRC:



2. Python: odkodowywanie ramek

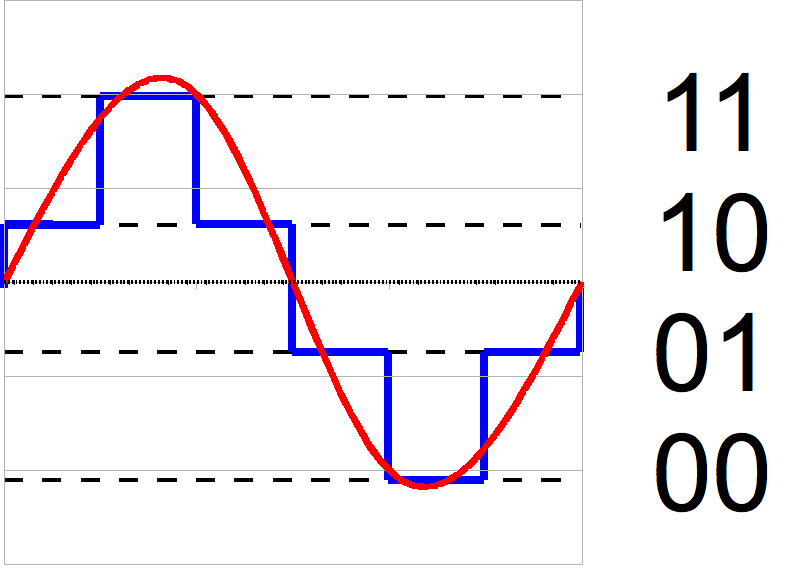
Dotychczasowy opracowany protokół komunikacji będzie się sprawdzał, gdy z medium transmisyjnego korzysta maksymalnie jedno urządzenie jednocześnie. W przypadku gdy z medium będą chciały skorzystać dwa urządzenia w tym samym czasie, ich sygnały będą się wzajemnie zagłuszały - wystąpi **kolizja**. Aby do takich kolizji dochodziło jak najrzadziej, każda sieć musi posiadać ustaloną **metodę dostępu do medium transmisyjnego**. Będzie to specjalny protokół narzucający na uczestników sieci warunki, wedle których mogą rozpocząć swoją transmisje.

**Protokół CSMA/CD** (Carrier Sense Multiple Access with Collision Detection) jest takim protokołem, wykorzystywanym w sieciach korzystających z połączenia *half-duplex2*.

***2half-duplex*** *(ang. duplex – dwustronny) - określenie połączenia, w którym możliwe jest jedynie naprzemienne wysyłanie   
i odbieranie informacji. Alternatywą są połączenia (full) duplex, gdzie informacje są przesyłane w obu kierunkach jednocześnie. Współczesne sieci korzystają z full duplexu, gdzie protokół CSMA/CD jest zbędny.*

Aby zrozumieć działanie protokołu CSMA/CD, wprowadzę małą dygresję pokrótce wyjaśniającą,   
jak komputer przetwarza napięcie prądu elektrycznego w kablu na ciąg zer i jedynek, czyli o tym jak działa **przetwornik analogowo-cyfrowy** (ADC – Analog to Digital Converter).

Przetwornik jest specjalnym układem sterującym, który odbiera prąd elektryczny w postaci **sygnału analogowego**, czyli takiego, który może przyjmować dowolne wartości z założonego zakresu,   
i upraszcza go (odwzorowuje) do **sygnału cyfrowego**, czyli takiego, który może przyjmować tylko określone wartości.



. Przybliżone odwzorowanie sygnału analogowego na cyfrowy

Na obrazku 3. mamy przykład, w którym płynnie zmieniające się wartości napięcia prądu z kabla są przetwarzane na zmieniające się skokowo cztery wartości: 0, 1, 2, 3 w zapisie binarnym. Jest to zatem rozbudowana forma interpretacji, o której pisałem na początku, gdzie 1 oznaczała sygnał a 0 jej brak.



. Symboliczne przedstawienie przetwornika AD

Przetwarzanie składa się z 3 etapów: **próbkowania**, czyli dokonania pomiaru wartości napięcia sygnału w danej chwili (przedział czasowy pomiędzy próbowaniami jest ustalony), **kwantyzacji** – przypisania jednej z ustalonych wcześniej wartości dziedziny sygnału cyfrowego do pobranej wartości napięcia (np. każdej wartości z zakresu 3-5V przypisz 102) i **kodowania**, gdzie nasz otrzymany ciąg zer i jedynek jest poddawany np. algorytmowi sprawdzania poprawności danych i przekształcany do formy zrozumiałej dla człowieka (proces odwrotny od opisywanego na początku sprawozdania).

Uzbrojeni w tę wiedzę możemy przejść do omówienia założeń algorytmu **CSMA/CD**:

1. **Każde urządzenie** podłączone do danej sieci w sposób ciągły **interpretuje sygnały** odbierane przez jego przetwornik analogowo-cyfrowy:

1. jeśli odbierane sygnały są przekształcane na najniższą ustaloną wartość (uogólniając: z kabla nie dociera żaden sygnał) interpretuje się to jako **wolne łącze** – żadne z urządzeń prawdopodobnie nie wysyła aktualnie wiadomości. W takiej sytuacji urządzenie może rozpocząć nadawanie swojej wiadomości.
2. jeśli z kabla będą docierały sygnały przekształcane na ciąg zer i jedynek oznacza to **zajętość łącza** – inne urządzenie korzysta aktualnie z medium do przesłania swojej wiadomości. W takiej sytuacji pozostali uczestnicy sieci nie mogą rozpocząć swojej transmisji.
3. jeśli z kabla będą docierały sygnały przetwarzane na największą założoną wartość oznacza to, że w medium transmisyjnym doszło do **kolizji**. Jest to tzw. **sygnał JAM** wysyłany przez urządzenie, które wykryło kolizję w sieci. W momencie odebrania takiego sygnału, urządzenia uznają odebrane tuż przed kolizją dane za przekłamane i oczekują na je ponowne wysłanie przez urządzenia, które weszły w kolizję.
4. jeśli **podczas wysyłania własnej wiadomości** urządzenie wykryje, że odczytuje w tym samym czasie **inne wartości sygnału niż sam generuje**, oznacza to, że w sieci doszło do nałożenia się jego sygnału z innym. W takim przypadku przerywa swoją transmisje i wysyła sygnał JAM, aby zasygnalizować innym urządzeniom, że doszło do kolizji i dane są nieważne. Następnie czeka losową ilość czasu i wysyła te dane ponownie, gdy łącze będzie wolne.

2. W momencie, gdy odbieranie sygnałów przez dane urządzenie się skończy i łącze staje się wolne, wprowadza się dodatkowo **strefę buforową**. Jest to ustalony czas, które urządzenia muszą odczekać   
po każdym zwolnieniu się łącza, aby móc nadawać swój sygnał.

3. Każde urządzenie samo wylicza zakres z jakiego jest losowany **czas oczekiwania** po kolizji, korzystając z ustalonego algorytmu. Zakres ten uzależnia się od **liczby wykonanych prób** wysłania pakietu - im więcej prób, tym dłuższy czas oczekiwania dla danego urządzenia. Taki sposób wyłaniania kolejności korzystania z medium jest konieczny, ze względu na brak nadrzędnego urządzenia, który mógłby wyznaczać kolejność dostępu pozostałym urządzeniom.

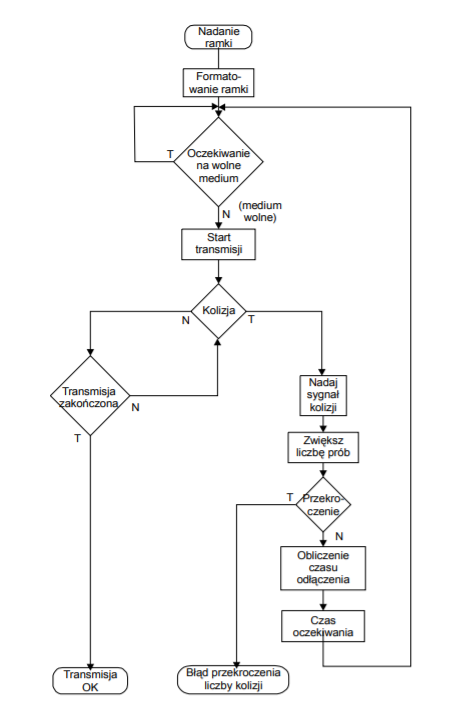
4. Przy **szesnastu kolejnych nieudanych próbach** wysłania pakietu (16 razy dojdzie do kolizji), urządzenie komunikuje informuje o niepowodzeniu program, który chciał wysłać wiadomość.   
Jeśli transmisja zakończy się sukcesem przed 16 próbą, licznik jest zerowany.

5. Urządzenie może dowiedzieć się o kolizji swojej ramki **jedynie podczas jej transmisji** (zanim skończy się jej wysyłanie). Urządzenie musi zatem mieć pewność, że przed zakończeniem wysyłania ramki, żadne z urządzeń nie rozpoczęło własnej transmisji nim nasz sygnał do niego dotarł.

Rozważmy następujący skrajny przypadek: mamy dwa urządzenia w sieci na dwóch końcach medium transmisyjnego. Węzeł sieci, znajdujący się na początku medium, rozpoczyna transmisje.   
Następnie, węzeł znajdujący się na końcu łącza, rozpoczyna swoją, tuż przed dotarciem sygnału pierwszego węzła - skoro nie dotarł do niego jeszcze żaden sygnał to uznaje, że łącze jest wolne.

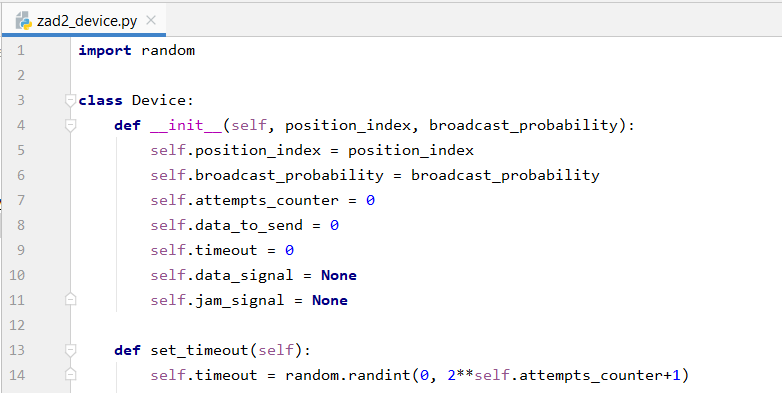
Aby pierwszy węzeł wykrył kolizję, do której w tym wypadku dojdzie, zakończenie wysyłania jego pakietu musi być co najmniej w tym samym momencie, co dotarcie do niego sygnału drugiego węzła. Z tego wynika, że **minimalna długość ramki** musi być **dwukrotnością medium transmisyjnego**.

Proces decyzyjny urządzeń w sieci i podejmowane przez nie działania zgodne z **protokołem** **CSMA/CD** przedstawia poniższy schemat:



. Schemat protokołu CSMA/CD

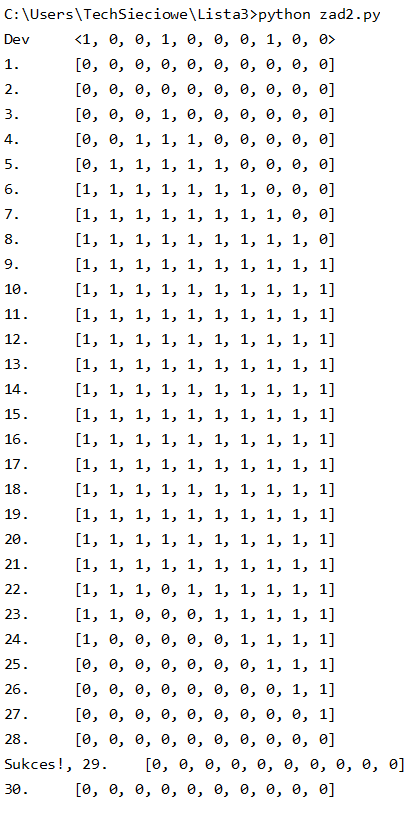
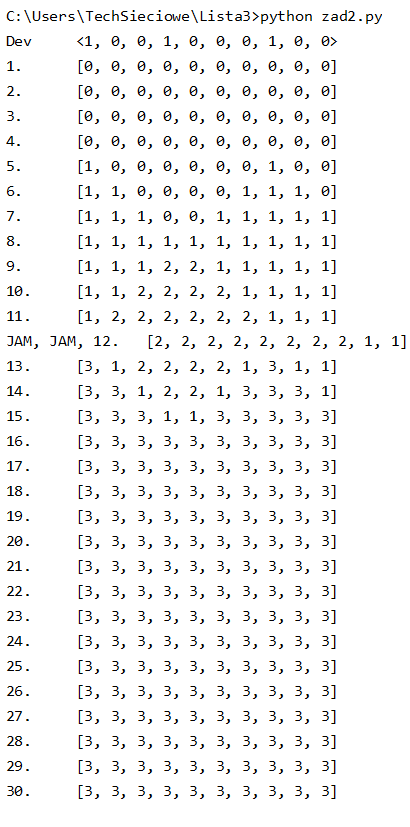
**Implementację** symulacji sieci zgodnej z protokołem CSMA/CD w **Pythonie** rozpocząłem od stworzenia klasy Device reprezentującej urządzenia w sieci:



6. Python: implementacja urządzenia w sieci

Każde urządzenie posiada swój **indeks** odpowiadający miejscu, w którym jest podłączony do medium transmisyjnego, prawdopodobieństwo chęci wysłania pakietu (w skali (0, 1)), **licznik nieudanych prób** wysłania pakietu, **liczba danych** przygotowanych do wysłania (w symulacji dla uproszczenia wysyłam zawsze minimalną dopuszczalną wielkość ramki), **licznik z czasem oczekiwania**, a także klasy reprezentujące **sygnały**, gdy urządzenie wysyła pakiet i gdy rozgłasza sygnał JAM.

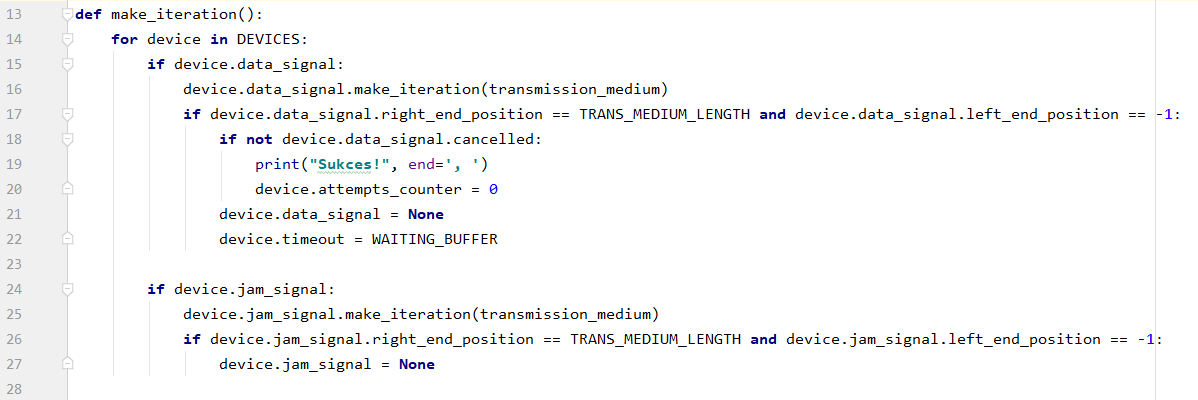
Medium transmisyjne jest reprezentowane przez tablicę o zadanej długości. Każdy element tablicy to jedna jednostka, dla której można zmierzyć napięcie prądu elektrycznego. Wartość **0** będzie oznaczała **brak** **napięcia**, **1** - zakres napięć wykorzystywanych do **przesyłania** **danych**, **2** – nałożenie się napięć dwóch sygnałów, czyli **kolizja** oraz **3** – napięcie generowane przez **sygnał** **JAM**.



8. Kolizja i wysłanie sygnału JAM

7. Pomyślne wysłanie pakietu

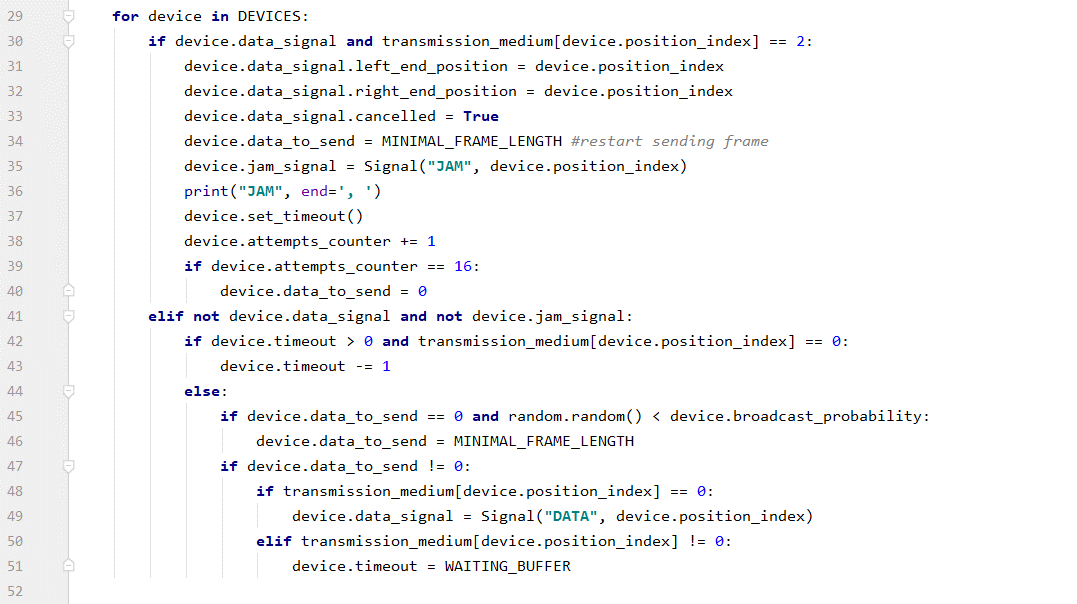
Kod realizujący algorytm ze schematu umieszczonego na zdjęciu 5.:



7. Python, symulacja sieci cz.1

Powyższy kod dokonuje iteracji dla każdego z sygnałów (wartość sygnału rozprzestrzenia się wtedy w tablicy o jedno pole w lewo i prawo), a także sprawdza czy nie zostały one pomyślnie zakończone.

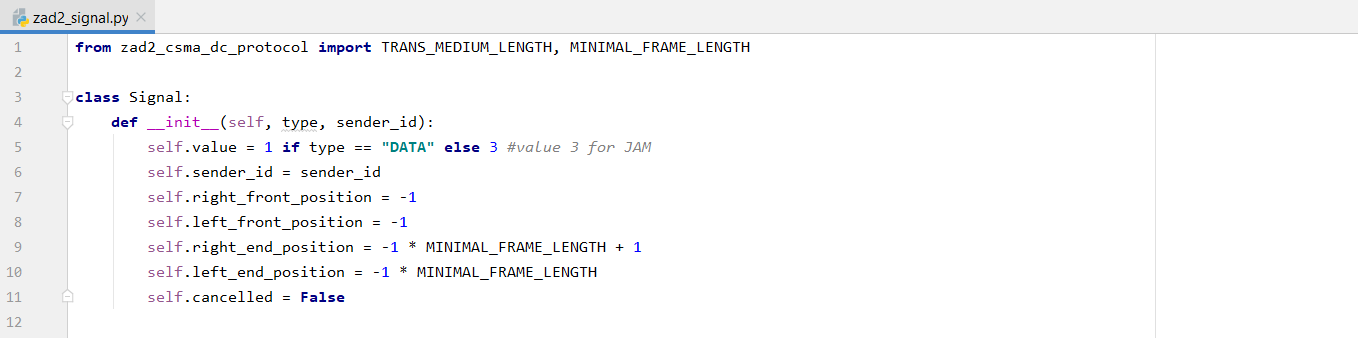
Poniższa część programu sprawdza dla każdego urządzenia czy nie doszło do kolizji (jeśli w aktualnej iteracji transmituje dane), sprawdza czy dane urządzenie ma pakiet do wysłania – jeśli tak to czy może go rozpocząć wysyłać w danej iteracji.



8. Python, symulacja sieci cz.2

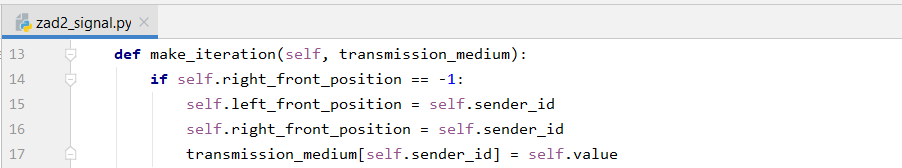
Klasa Signal przechowuje **wartość napięcia prądu** zgodnie z założeniami, która wypisałem wcześniej, **id** **urządzenia** (miejsce, w którym jest podpięte do kabla), z którego został wysłany. Aby zasymulować rozchodzenie się sygnału korzystam z **4 pomocniczych wskaźników**: aktualne miejsce początku i końca sygnału, który rozchodzi się w lewą i prawą stronę kabla. Ostatnim polem klasy jest **flaga** informująca o tym, czy wykryto **kolizję** podczas wysyłania danego sygnału.

Kod znajduje się na następnej stronie.



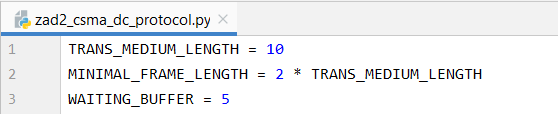
. Python: pola klasy Signal

Zapis symulacji iteracji rozchodzenia się sygnałów jest rozbudowany, ale prosty. Na początku sprawdzam czy wywołana iteracja nie jest pierwszą. Jeśli tak, to znacznikom początku sygnału przypisuje miejsce, w którym podpięte jest dane urządzenie i przypisuję wartość sygnału do tablicy w tym miejscu:



. Python: rozpoczęcie wysyłania sygnału

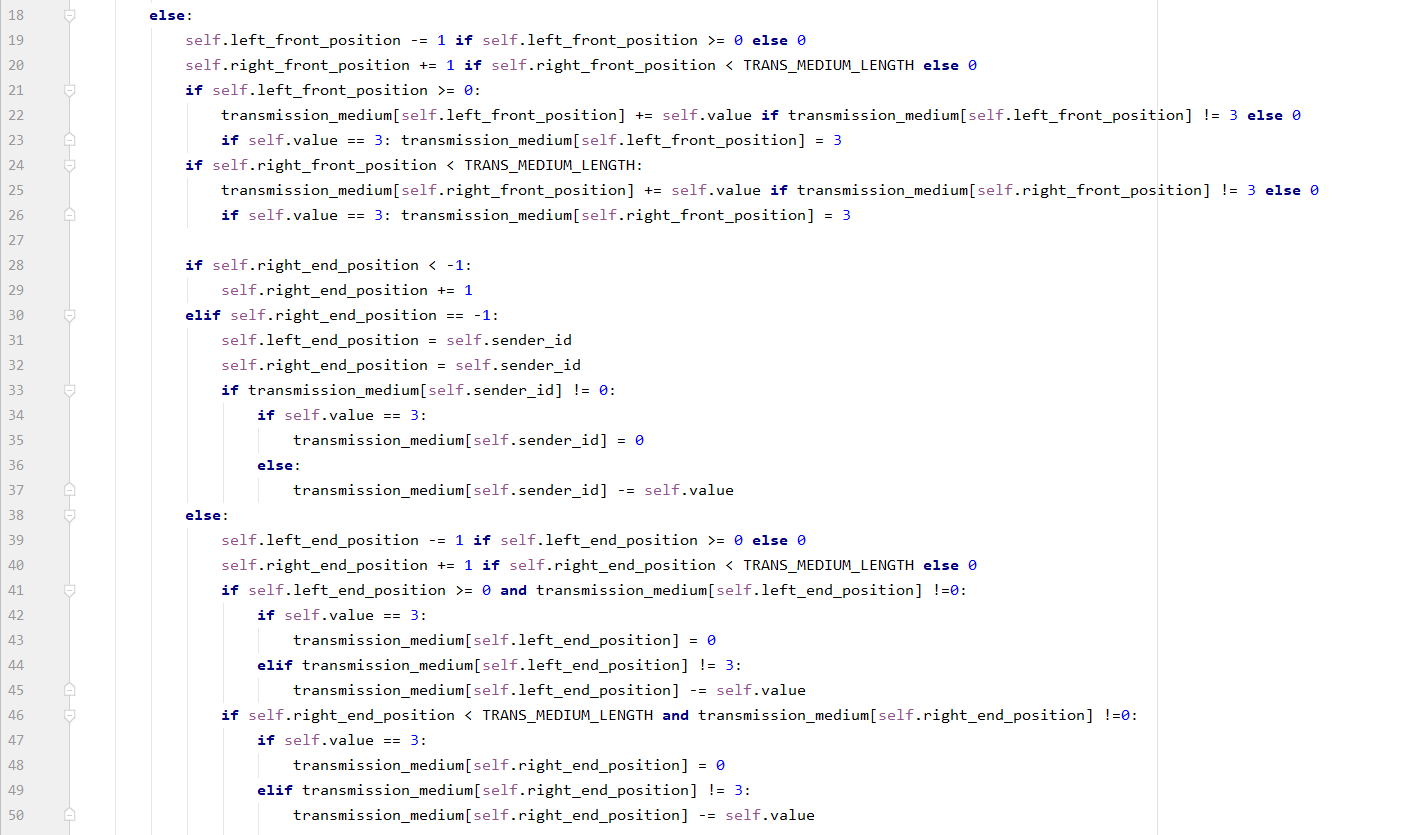
Co ważne, wskaźniki końca sygnału podczas inicjowania obiektu zostały ustawione na wartość ujemną będącą wartością jednej ramki *(rys. 9)*. Można wyobrazić sobie, że dane te są ustawione i przygotowane, aby wejść do medium transmisyjnego, znajdując się gdzieś w urządzeniu, poza medium transmisyjnym. Dzięki inkrementowaniu tych wartości w kolejnych iteracjach, kolejne dane są „wypychane” z urządzenia do kabla. Taki sposób zapewnia nam właściwą długość generowania jednego sygnału. Tak jak omawiałem to wcześniej, minimalna wartość ramki wynosi dwukrotność długości medium transmisyjnego:



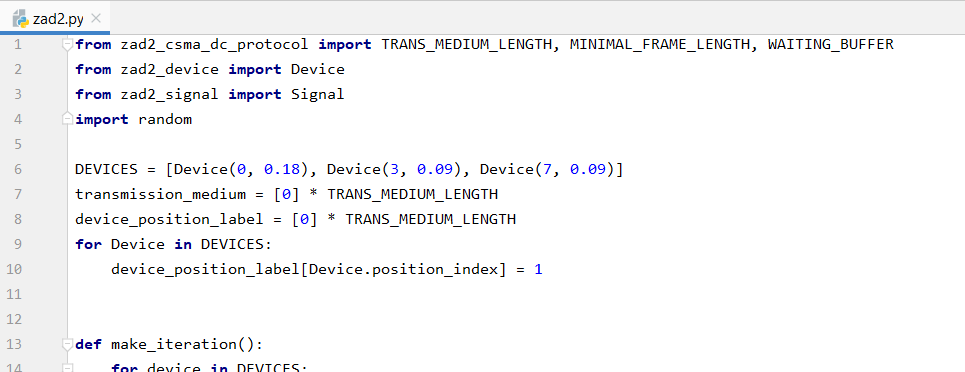
. Python: parametry funkcjonowania sieci

Wracając do algorytmu rozchodzenia się sygnału, w następnych iteracjach konsekwentnie przesuwamy znaczniki końca i początku sygnału. W kolejnych zdobywanych komórkach tablicy dodajemy wartość naszego sygnału (jeśli nie jest to sygnał JAM – wtedy wartość komórki tablicy jest ustawiana na najwyższą, czyli 3, niezależnie jaka wartość była tam wcześniej). Natomiast dzięki znacznikom końca sygnału możemy odejmować wartość naszego sygnału, w kolejnych miejscach gdzie będzie zamykał nasz sygnał.   
Dodatkowo w naszym kodzie musimy zadbać o to, aby nasze wskaźniki nie wyszły poza zakres tablicy,   
a także by nie odejmować podwójnie wartości sygnału JAM, gdy był on wysłany przez więcej urządzeń.

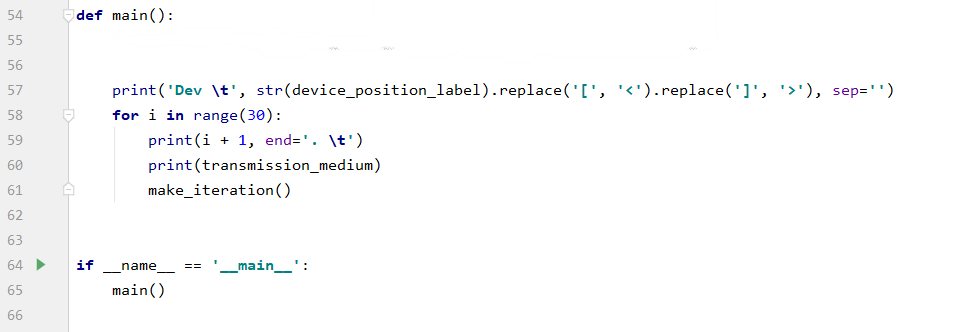
Kod umieściłem na następnej stronie.

Naszą sieć możemy formować za pomocą parametrów TRANS\_MEDIUM\_LENGTH (długość medium transmisyjnego, tj. kabla), DEVICES (lista urządzeń, które biorą udział w komunikacji wraz z podaniem ich miejsca podpięcia i prawdopodobieństwa chęci wysłania pakietów), WAITING\_BUFFER (czas oczekiwania pomiędzy wysyłanymi pakietami), a także wybrać liczbę iteracji (jednostek czasu), w której przetestujemy działanie naszej sieci. Dodatkowo, dzięki fladze cancelled dla sygnałów, możemy zliczać udane transmisje oraz kolizje. W konsoli natomiast wypisywane jest graficzne przedstawienie tego, w jaki sposób rozchodzą się sygnały w medium na przestrzeni całej symulacji.

12. Python: symulacja rozchodzenia się sygnału



. Python: parametry symulacj sieci



. Python: symulacja ustawiona na 30 jednostek czasu

**Zaletą** protokołu CSMA/CD jest to, że liczba i rozmieszczenie stacji może się zmieniać w trakcie działania sieci, bez wykonywania żadnych ponownych konfiguracji. Plusem jest także to, że sieć nie faworyzuje z góry żadnego z urządzeń. Z każdą kolizją prawdopodobieństwo kolejnych jest coraz mniejsze, ze względu na zwiększanie zakresu losowości. **Wadą** protokołu jest zwiększający się czas oczekiwania   
w przypadku kolizji, czy też konieczność dostosowywania urządzeń do ich wykrywania.



. Wywołanie programu dla medium długości 50 i 8 urządzeń

Sieci bezprzewodowe posiadają swój rodzaj protokołu CSMA, ze względu na inną specyfikę.   
W sieci bezprzewodowej nie wszystkie urządzenia „słyszą” się nawzajem, przez co węzły A i C mogą nadawać jednocześnie do węzła B, nie wiedząc o swoim istnieniu. Stosowany jest zatem inny protokół wielodostępu - **CSMA/CA** (*Carrier Sense Multiple Access with Collision Avoidance)*. Według tego protokołu, każde urządzenie przed próbą rozpoczęcia transmisji wysyła **sygnał próbny**. Jeżeli nie zaszła kolizja z sygnałem innego urządzenia – **uzyskuje zgodę** na nadawanie, po czym następuje **właściwa transmisja**.   
W sieci bezprzewodowej detekcja kolizji podczas transmisji nie jest możliwa, więc stosowany jest **sygnał potwierdzenia** poprawności odbioru.