



**POLITECHNIKA
RZESZOWSKA**
im. IGNACEGO ŁUKASIEWICZA



**WYDZIAŁ
MATEMATYKI
I FIZYKI STOSOWANEJ**
POLITECHNIKI RZESZOWSKIEJ

POLITECHNIKA RZESZOWSKA

im. Ignacego Łukasiewicza

WYDZIAŁ MATEMATYKI I FIZYKI STOSOWANEJ

APTEEN

Adaptive Threshold-sensitive Energy Efficient Sensor Network Protocol

Przedmiot: Projektowanie systemów i sieci komputerowych

Wykonawcy:

Vitalii Omeliukh [181517]

Dmytro Pavlyk [181518]

Grupa: 4

Opiekun projektu:

dr inż. Andrzej Paszkiewicz

Rzeszów, 25 stycznia 2026

Spis treści

Wstęp teoretyczny: Protokoły routingu w WSN	5
Charakterystyka protokołu LEACH	5
Charakterystyka protokołu TEEN	8
Hybrydowy protokół APTEEN	9
Analiza zastanego rozwiązania (Fork repozytorium)	12
Struktura oryginalnego projektu	12
Identyfikacja braków w implementacji	12
Analiza procesu symulacji	16
Wnioski z analizy dla procesu wdrożenia zmian	18
Wdrożone zmiany i implementacja APTEEN	19
Modyfikacje w strukturze plików	19
Implementacja logiki protokołu APTEEN	19
System scenariuszy i obsługa CLI	22
Lokalizacja i nowa wizualizacja	23
Uruchomienie symulacji i analiza wyników	25
Konfiguracja środowiska i zależności	25
Narzędzia symulacyjne i parametry sterujące	26
Eksperyment 1: Analiza porównawcza LEACH i APTEEN	28
Podsumowanie i wnioski końcowe	35
Zakres zrealizowanych prac	35
Wnioski z analizy porównawczej	35
Znaczenie praktyczne projektu	36
Bibliografia	37

Wstęp teoretyczny: Protokoły routingu w WSN

Bezprzewodowe sieci sensorowe (WSN – ang. *Wireless Sensor Networks*) stanowią jedną z najbardziej dynamicznie rozwijających się technologii w dziedzinie monitoringu środowiska i systemów nadzoru. Składają się one z setek, a nawet tysięcy niewielkich, autonomicznych węzłów, które integrują w sobie funkcje pomiarowe (sensory), obliczeniowe (mikrokontrolery) oraz komunikacyjne (transceivery radiowe) [2, 3].

Rys. 1. Schemat architektury bezprzewodowej sieci sensorowej (WSN).

Podstawowym ograniczeniem tych systemów jest zasilanie. Węzły są zazwyczaj wyposażone w baterie o niewielkiej pojemności, których wymiana w trudnodostępnym terenie jest często niemożliwa lub nieopłacalna. Z tego względu kluczowym wyzwaniem projektowym staje się minimalizacja zużycia energii, aby zmaksymalizować czas życia całej sieci [1]. Tradycyjne podejścia do routingu, takie jak transmisja bezpośrednia (każdy węzeł wysyła dane prosto do stacji bazowej) lub proste protokoły wieloskokowe (MTE – *Minimum-Transmission-Energy*), często okazują się nieefektywne. W przypadku transmisji bezpośredniej, węzły oddalone od stacji bazowej zużywają energię w tempie wykładniczym, natomiast w MTE węzły znajdujące się blisko stacji bazowej są przeciążone ruchem tranzytowym i umierają jako pierwsze, odcinając resztę sieci [2].

Aby rozwiązać te problemy, opracowano hierarchiczne protokoły routingu oparte na klastrowaniu. Dzielą one sieć na mniejsze podgrupy (klastry), w których wybrane węzły (liderzy) przejmują na siebie ciężar komunikacji dalekosiężnej i przetwarzania danych. Poniższe podrozdziały szczegółowo omawiają trzy kluczowe protokoły z tej rodziny: LEACH, TEEN oraz APTEEN, które stanowią fundament niniejszej pracy.

Charakterystyka protokołu LEACH

Protokół LEACH (*Low-Energy Adaptive Clustering Hierarchy*) został wprowadzony jako pierwsze kompleksowe rozwiązanie wykorzystujące adaptacyjne, dynamiczne klastrowanie w celu równomiernego rozłożenia obciążenia energetycznego w sieciach sensorowych [2]. Jest to protokół klasy *proactive* (proaktywny), co oznacza, że węzły cyklicznie i nieprzerwanie monitorują środowisko, przesyłając dane w regularnych odstępach czasu, niezależnie od występowania zdarzeń.

Główną ideą LEACH jest losowa rotacja funkcji lidera klastra (CH – *Cluster Head*). W tradycyjnych algorytmach statycznych, węzeł wybrany na lidera pełnił tę funkcję do wyczerpania baterii, co następowało bardzo szybko ze względu na konieczność ciągłego nasłuchu, agregacji danych i energochłonnej transmisji do stacji bazowej. LEACH zapobiega temu poprzez okresową zmianę ról, dzięki czemu energochłonne zadania są „rozsmarowywane” po wszystkich węzłach w sieci.

Model energetyczny radia (First Order Radio Model)

Aby zrozumieć zysk energetyczny płynący z zastosowania LEACH, konieczne jest przeanalizowanie modelu zużycia energii przez moduł radiowy. W literaturze przedmiotu stosuje się tzw. „model radiowy pierwszego rzędu”. Zakłada on, że całkowita energia zużywana przez węzeł składa się z energii potrzebnej na zasilanie elektroniki nadawczej/odbiorczej (E_{elec}) oraz energii zużywanej przez wzmacniacz sygnału (E_{amp}) w celu uzyskania odpowiedniego stosunku sygnału do szumu na dystansie d [2].

Rys. 2. Model radiowy pierwszego rzędu (First Order Radio Model).

Energię niezbędną do wysłania wiadomości o długości k bitów na odległość d wyraża wzór:

$$E_{Tx}(k, d) = E_{Tx-elec}(k) + E_{Tx-amp}(k, d) \quad (1)$$

Co po podstawieniu parametrów modelu przyjmuje postać:

$$E_{Tx}(k, d) = E_{elec} \cdot k + \epsilon_{amp} \cdot k \cdot d^2 \quad (2)$$

gdzie:

- E_{elec} – energia rozpraszana przez układy elektroniczne (np. 50 nJ/bit),
- ϵ_{amp} – współczynnik wzmocnienia dla modelu propagacji w wolnej przestrzeni (np. 100 pJ/bit/m²).

Z kolei energia zużywana przez odbiornik na odebranie tej samej wiadomości zależy tylko od pracy elektroniki:

$$E_{Rx}(k) = E_{elec} \cdot k \quad (3)$$

Z powyższych zależności wynika, że koszt transmisji rośnie kwadratowo wraz z odległością (d^2). Dlatego bezpośrednia komunikacja każdego węzła ze stacją bazową (która jest zazwyczaj oddalona) jest skrajnie nieefektywna. LEACH rozwiązuje ten problem, umożliwiając większości węzłów transmisję na bardzo krótki dystans (tylko do lidera klastra), a jedynie nieliczne węzły (liderzy) wykonują kosztowną transmisję dalekosiężną.

Algorytm formowania klastrów i próg decyzyjny

Działanie protokołu LEACH odbywa się w cyklicznych rundach. Każda runda składa się z dwóch głównych faz: fazy formowania klastrów (*set-up phase*) oraz fazy transmisji danych (*steady-state phase*). Aby zminimalizować narzut sterujący, faza transmisji jest zazwyczaj znacznie dłuższa niż faza formowania.

W fazie formowania klastrów, węzły autonomicznie decydują, czy w bieżącej rundzie zostaną liderami (CH). Decyzja ta jest podejmowana stochastycznie. Każdy węzeł n losuje liczbę z przedziału $\langle 0, 1 \rangle$. Jeśli wylosowana liczba jest mniejsza od wyliczonego progu $T(n)$, węzeł mianuje się liderem. Wartość progu $T(n)$ jest kluczowym elementem algorytmu i definiowana jest wzorem [2]:

$$T(n) = \begin{cases} \frac{P}{1 - P \cdot (r \bmod \frac{1}{P})} & \text{dla } n \in G \\ 0 & \text{w p.p.} \end{cases} \quad (4)$$

Gdzie poszczególne symbole oznaczają:

- P – oczekiwany procent węzłów, które mają stać się liderami (optymalnie około 5% dla typowych sieci),
- r – numer bieżącej rundy,
- G – zbiór węzłów, które nie pełniły funkcji lidera w ostatnich $1/P$ rundach.

Mechanizm ten gwarantuje, że każdy węzeł w sieci zostanie liderem dokładnie raz w ciągu $1/P$ rund. Po upływie tego okresu, wszystkie węzły ponownie stają się kandydatami i proces zaczyna się od nowa. Dzięki temu energia zużywana na bycie liderem jest sprawiedliwie rozkładana na wszystkie węzły w czasie.

Faza formowania i transmisji (Steady-State)

Po mianowaniu się liderem, węzeł CH rozgłasza swoją nową rolę za pomocą wiadomości reklamowej (*advertisement message*), wykorzystując protokół CSMA (*Carrier Sense Multiple Access*). Pozostałe węzły, niebędące liderami, nasłuchują tych ogłoszeń i podejmują decyzję o przyłączeniu się do konkretnego klastra. Decyzja ta opiera się na sile odebranego sygnału (RSSI) – węzeł wybiera tego lidera, którego sygnał jest najsilniejszy, co w przybliżeniu odpowiada najmniejszej odległości i najniższemu kosztowi transmisji.

Po sformowaniu klastrów następuje kluczowy etap dla oszczędności energii wewnątrz klastra: lider tworzy harmonogram TDMA (*Time Division Multiple Access*). Harmonogram ten przydziela każdemu członkowi klastra unikalną szczelinę czasową, w której może on wysłać swoje dane. Ma to dwie zalety:

1. **Brak kolizji:** Wewnątrz klastra nie dochodzi do kolizji pakietów, co oszczędza energię na retransmisje.
2. **Usypianie radia:** Węzły mogą wyłączyć swoje moduły radiowe przez większość czasu trwania ramki, budząc się jedynie w swoim slotcie czasowym.

W fazie ustalonej (*steady-state*), węzły przesyłają dane pomiarowe do lidera. Lider odbiera dane od wszystkich członków, a następnie dokonuje ich fuzji (agregacji), np. poprzez wyliczenie średniej lub kompresję. Zredukowana w ten sposób ilość danych jest przesyłana do stacji bazowej. Aby uniknąć interferencji między sąsiednimi klastrami komunikującymi się w tym samym czasie, LEACH wykorzystuje technologię CDMA (*Code Division Multiple Access*), gdzie każdy klaster używa innej sekwencji kodowej rozpraszania widma.

Charakterystyka protokołu TEEN

Podczas gdy LEACH sprawdza się w aplikacjach wymagających stałego monitoringu, wiele zastosowań WSN (np. wykrywanie pożarów, intruzów) wymaga natychmiastowej reakcji na zdarzenia krytyczne. Dla takich scenariuszy opracowano protokół TEEN (*Threshold sensitive Energy Efficient sensor Network protocol*), który należy do klasy sieci reaktywnych [3]. W sieciach tych transmisja danych nie odbywa się w sposób ciągły, lecz jest inicjowana nagłą zmianą wartości mierzonego atrybutu.

Koncepcja progów transmisji (Hard & Soft Thresholds)

Fundamentem działania TEEN jest mechanizm podwójnego progowania, który drastycznie redukuje liczbę przesyłanych pakietów. Lider klastra, oprócz harmonogramu, rozsyła do węzłów dwa parametry sterujące:

- **Hard Threshold (H_T) – Próg Twardy:** Jest to bezwzględna wartość mierzonego parametru (np. temperatura $> 100^\circ\text{F}$), powyżej której węzeł w ogóle rozważa włączenie nadajnika. Jeśli mierzona wartość jest poniżej tego progu, węzeł milczy, oszczędzając energię. Pozwala to na ignorowanie danych nieistotnych dla użytkownika.
- **Soft Threshold (S_T) – Próg Miękki:** Jest to minimalna różnica (inkrement) pomiędzy bieżącym pomiarem a ostatnio wysłaną wartością, która uzasadnia nową transmisję.

Logika decyzyjna węzła w TEEN

Węzeł pracujący w protokole TEEN przechowuje w pamięci zmienną SV (*Sensed Value*), reprezentującą ostatnią zaraportowaną wartość. W każdym cyklu pomiarowym węzeł sprawdza dwa warunki logiczne, aby podjąć decyzję o wysłaniu danych [3]:

1. Czy aktualna wartość sensora jest większa od progu twardego ($Current_Value > H_T$)?
2. Czy różnica między aktualną wartością a ostatnio wysłaną jest większa lub równa progowi miękkiemu ($|Current_Value - SV| \geq S_T$)?

Transmisja następuje tylko wtedy, gdy oba te warunki są spełnione jednocześnie. Po wysłaniu danych, zmienna SV jest aktualizowana do bieżącej wartości.

Dzięki takiemu podejściu, TEEN eliminuje przesyłanie powtarzających się informacji. Jeśli temperatura wzrośnie powyżej H_T , ale następnie ustabilizuje się i nie zmienia się o więcej niż S_T , węzeł przestaje nadawać, mimo że stan alarmowy trwa. Użytkownik ma pewność, że otrzymuje tylko informacje o istotnych zmianach. Możliwość regulacji parametru S_T pozwala na elastyczne sterowanie kompromisem między dokładnością odwzorowania zjawiska a zużyciem energii.

Główną wadą TEEN jest jednak problem „cichych węzłów”. Jeśli progi nie zostaną przekroczone, węzły nigdy nie skomunikują się ze stacją bazową. Użytkownik nie jest w stanie odróżnić sytuacji, w której w sieci panuje spokój, od sytuacji, w której wszystkie węzły uległy awarii.

Hybrydowy protokół APTEEN

Protokół APTEEN (*Adaptive Periodic Threshold-sensitive Energy Efficient sensor Network*) stanowi ewolucyjne połączenie zalet protokołów LEACH i TEEN [1]. Jest to protokół hybrydowy, zaprojektowany w celu zapewnienia zarówno okresowego raportowania stanu sieci (cecha LEACH), jak i natychmiastowej reakcji na zdarzenia krytyczne (cecha TEEN).

Rozszerzone parametry i tryb Count Time

Architektura APTEEN bazuje na hierarchicznym klastrowaniu (podobnym do LEACH-C), gdzie stacja bazowa bierze udział w wyborze liderów, aby zoptymalizować ich rozmieszczenie i zużycie energii. Kluczową innowacją jest jednak zestaw parametrów rozsyłanych przez lidera klastra po jego sformowaniu. Oprócz znanych z TEEN progów, dochodzi parametr czasu zliczania:

- **Attributes (A):** Zdefiniowany przez użytkownika zestaw parametrów fizycznych (np. temperatura, wilgotność), które mają być monitorowane.
- **Thresholds (H_T, S_T):** Progi twardy i miękki, działające na zasadzie filtra zdarzeń.
- **Schedule:** Harmonogram TDMA przydzielający sloty transmisyjne dla poszczególnych węzłów.
- **Count Time (T_C):** Czas zliczania. Jest to maksymalny okres czasu (mierzony np. w liczbie rund lub jednostkach czasu), jaki może upłynąć pomiędzy dwoma kolejnymi raportami z danego węzła.

Parametr T_C wprowadza komponent proaktywny do reaktywnego mechanizmu TEEN. Algorytm decyzyjny węzła w APTEEN wygląda następująco:

1. Węzeł sprawdza warunki progowe (H_T i S_T) analogicznie jak w TEEN. Jeśli są spełnione – wysyła dane (tryb reaktywny).
2. Niezależnie od progów, węzeł monitoruje czas, jaki upłynął od ostatniej transmisji. Jeśli czas ten osiągnie wartość T_C , węzeł zostaje *wymuszony* do wysłania danych, nawet jeśli progi nie zostały przekroczone.

Dzięki temu mechanizmowi, APTEEN gwarantuje, że użytkownik otrzymuje pełny obraz sieci („snapshot”) co najmniej raz na okres T_C , jednocześnie będąc informowanym natychmiast o nagłych anomaliach. Eliminuje to problem nierozróżnialności martwych węzłów, który występował w TEEN.

Zaawansowana obsługa zapytań (Query Modeling)

APTEEN wyróżnia się na tle innych protokołów zdolnością do obsługi trzech różnych typów zapytań od użytkownika, co czyni go niezwykle elastycznym narzędziem analitycznym [1]:

1. Zapytania historyczne (Historical queries)

Są to zapytania dotyczące danych z przeszłości, np. „Jaka była średnia temperatura w sektorze północnym 2 godziny temu?”. Dzięki mechanizmowi T_C , stacja bazowa okresowo otrzymuje dane ze wszystkich węzłów i archiwizuje je w swojej pamięci. W rezultacie, odpowiedź na zapytanie historyczne może zostać wygenerowana natychmiastowo przez samą stację bazową (BS), bez konieczności komunikacji z siecią sensorową i zużycia energii węzłów.

2. Zapytania jednorazowe (One-time queries)

Dają one natychmiastowy wgląd w aktualny stan sieci, np. „Jaka jest obecna temperatura w klastrze nr 5?”.

- Jeśli zapytanie dotyczy danych krytycznych (które przekroczyły progi), BS prawdopodobnie już je posiada dzięki mechanizmowi reaktywnemu.
- Jeśli zapytanie dotyczy danych poniżej progów, BS rozsyła zapytanie do odpowiednich liderów klastrów. Liderzy przekazują je do węzłów w swoich slotach czasowych. Węzły spełniające kryteria zapytania przesyłają odpowiedź w przydzielonym slotcie TDMA. Czas odpowiedzi jest deterministyczny i zależy od długości ramki TDMA.

3. Zapytania trwałe (Persistent queries)

Służą do monitorowania określonego obszaru przez zadany przedział czasu, np. „Raportuj temperaturę w sektorze zachodnim przez najbliższe 30 minut, jeśli przekroczy 50 stopni”. Działają one na zasadzie modyfikacji zachowania węzłów na określony czas. Po otrzymaniu takiego zapytania, węzły zachowują się tak, jak przy zapytaniach jednorazowych, ale powtarzają transmisję cyklicznie przez czas trwania zapytania.

Analiza wydajności i wnioski

Badania symulacyjne porównujące protokoły LEACH, TEEN i APTEEN wykazały wyraźne różnice w ich charakterystyce energetycznej.

- **TEEN** zużywa najmniej energii, ponieważ w okresach spokoju (brak przekroczeń progów) sieć praktycznie nie transmituje danych. Jest to jednak okupione brakiem pełnej wiedzy o stanie sieci.
- **LEACH** zużywa najwięcej energii ze względu na ciągłą, bezwarunkową transmisję danych, co prowadzi do szybszego wyczerpania baterii węzłów.
- **APTEEN** plasuje się pośrodku. Zużywa mniej energii niż LEACH (dzięki progom H_T/S_T), ale więcej niż TEEN (z powodu narzutu transmisji wymuszonych przez T_C).

APTEEN jest zatem rozwiązaniem kompromisowym, oferującym „to, co najlepsze z obu światów”: oszczędność energii zbliżoną do sieci reaktywnych oraz kompletność danych typową dla sieci proaktywnych. Jego elastyczność, zapewniana przez regulowane parametry H_T, S_T, T_C , pozwala administratorowi sieci na dynamiczne dostosowywanie zachowania systemu do aktualnych potrzeb aplikacji i stanu naładowania baterii.

Analiza zastanego rozwiązania (Fork repozytorium)

Punktem wyjścia do realizacji praktycznej części pracy było repozytorium `wsn_routing`, zidentyfikowane jako projekt open-source realizujący symulację algorytmów routingu w języku Python. Wstępne uruchomienie projektu pozwoliło na zweryfikowanie jego gotowości do dalszych modyfikacji oraz ocenę zaimplementowanych mechanizmów.

Struktura oryginalnego projektu

Analiza kodu źródłowego wykazała, że projekt posiada modułową architekturę, podzieloną na logiczne komponenty odpowiedzialne za routing, optymalizację oraz wizualizację. Głównym językiem implementacji jest Python 3.

Wstępny przegląd katalogów ujawnił następujący podział odpowiedzialności w systemie:

- **router/** – moduł zawierający logikę węzłów i protokołów (w tym modele LEACH).
- **optimizer/** – moduł realizujący algorytmy optymalizacyjne (m.in. PSO).
- **Skrypty uruchomieniowe** – pliki w głównym katalogu służące do startowania symulacji.

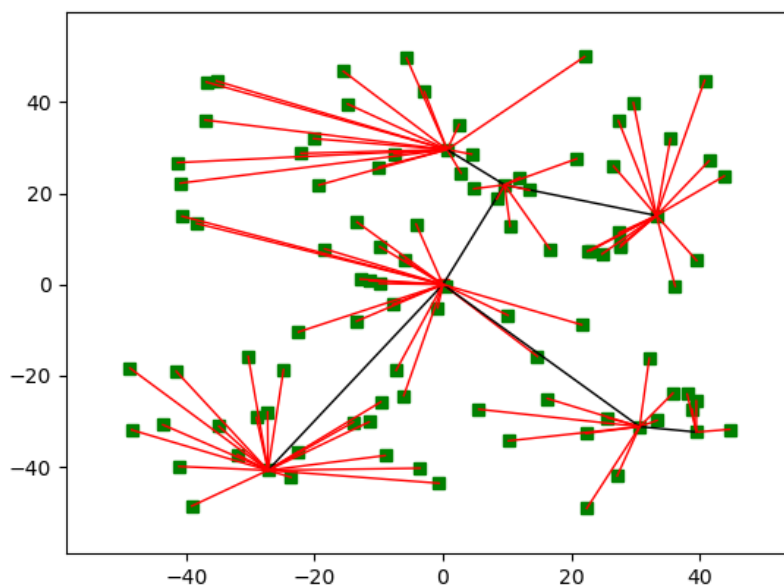
Szczegółowa organizacja plików w repozytorium, wraz z opisem odpowiedzialności poszczególnych modułów, została przedstawiona poniżej.

Projekt wyposażony jest również w system wizualizacji, który pozwala na generowanie mapy topologii sieci w przestrzeni dwuwymiarowej. Funkcja `plot()` umożliwia obserwację rozmieszczenia węzłów, ich statusu (żywy/martwy) oraz tras przesyłu danych między klastrami a stacją bazową (Rys. 3).

Weryfikacja działania skryptów testowych potwierdziła, że środowisko jest skonfigurowane poprawnie i generuje powtarzalne wyniki symulacji, co stanowiło solidną bazę do implementacji własnych rozszerzeń.

Identyfikacja braków w implementacji

Szczegółowa analiza kodu odpowiedzialnego za protokół APTEEN (`router/apteen.py`) ujawniła, że w oryginalnym repozytorium algorytm ten nie został w pełni zaimplementowany. Klasa `APTEEN` została zdefiniowana jedynie jako klasa



Rys. 3. Wizualizacja topologii sieci oraz ścieżek routingu wygenerowana przez oryginalny kod symulatora.

dziedzicząca po LEACHPrim, bez nadpisywania kluczowych metod odpowiedzialnych za logikę reaktywną.

Fragment oryginalnego kodu przedstawiono poniżej:

Listing 1. Oryginalna implementacja klasy APTEEN

```

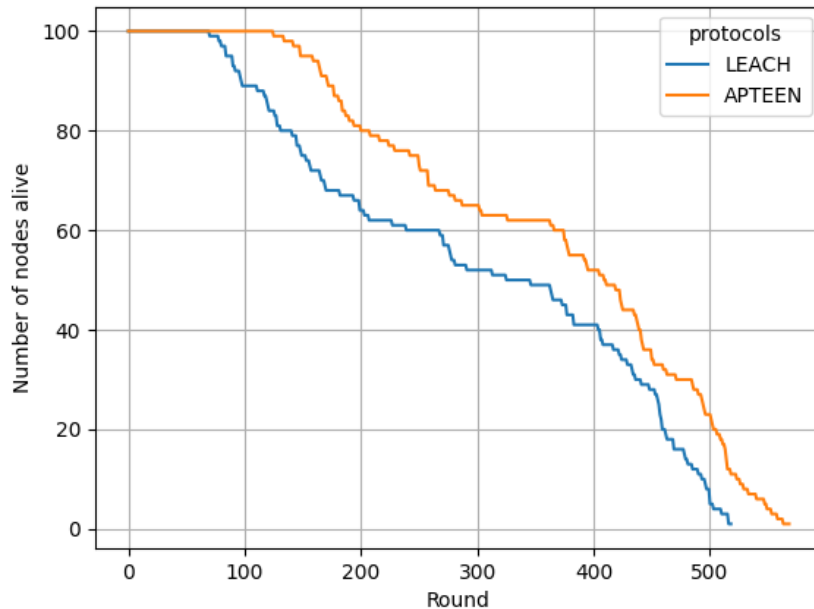
1  from router.leach import LEACHPrim
2
3  class APTEEN(LEACHPrim):
4  pass # Klasa jedynie dziedziczy po LEACHPrim, brak
      logiki progowej

```

Powyższy kod oznacza, że tzw. „APTEEN” w tej wersji działał identycznie jak protokół LEACHPrim (wersja LEACH z routingiem wieloskokowym opartym na algorytmie Prima). Brakowało w nim fundamentalnych cech protokołu hybrydowego:

1. **Brak obsługi progów:** Nie zdefiniowano parametrów *Hard Threshold* (HT) oraz *Soft Threshold* (ST).
2. **Brak mechanizmu reaktywnego:** Węzły wysyłały dane w sposób ciągły (proaktywny), a nie w odpowiedzi na zmiany wartości sensora.
3. **Brak parametru czasu zliczania:** Nie zaimplementowano licznika *Count Time* (TC), który wymuszałby okresową transmisję w przypadku braku zdarzeń.

Przeprowadzono symulację porównawczą uruchamiając skrypt `compare_leach_and_apteen.py`. Wyniki przedstawione na Rys. 4 pokazują, że „APTEEN” (pomarańczowa linia) osiąga lepsze wyniki niż standardowy LEACH (niebieska linia).



Rys. 4. Porównanie czasu życia sieci dla LEACH i oryginalnej implementacji APTEEN.

Należy jednak podkreślić, że widoczna różnica w wydajności nie wynikała z zastosowania mechanizmów oszczędzania energii specyficznych dla APTEEN (filtrowanie danych przez progi), lecz z faktu, że klasa bazowa `LEACHPrim` wykorzystuje bardziej efektywny routing wieloskokowy wewnątrz klastrów, podczas gdy standardowy LEACH wykorzystuje transmisję jednoskokową. Potwierdziło to konieczność napisania właściwej logiki APTEEN od podstaw, aby wprowadzić rzeczywistą funkcjonalność sieci hybrydowej.

Szczegółowa analiza repozytorium pozwoliła na wyodrębnienie trzech głównych warstw funkcjonalnych: warstwy logiki sieciowej, warstwy optymalizacji oraz warstwy orkiestracji symulacji. Poniżej przedstawiono charakterystykę poszczególnych katalogów i plików wchodzących w skład projektu.

Moduł routingu i logiki węzłów (router/)

Katalog `router/` stanowi rdzeń symulatora. To tutaj zaimplementowano abstrakcję sprzętową węzłów sensorowych oraz logikę protokołów komunikacyjnych.

- **node.py – Model fizyczny węzła.** Plik ten definiuje klasę `SensorNode`, która stanowi cyfrową reprezentację fizycznego urządzenia. Odpowiada za:
 - Przechowywanie stanu energetycznego (poziom naładowania baterii).
 - Implementację modelu radiowego pierwszego rzędu (obliczanie kosztu nadawania i odbioru w zależności od dystansu).
 - Zarządzanie stanem aktywności (węzeł żywy/martwy).
- **router.py – Silnik symulacji.** Klasa bazowa `Router` pełni rolę orkiestratora dla wszystkich protokołów. Nie implementuje konkretnego algorytmu routingu, lecz dostarcza wspólny interfejs do:
 - Inicjalizacji sieci (rozmieszczenie węzłów).
 - Zarządzania podziałem czasu na rundy (*rounds*).
 - Wizualizacji graficznej (rysowanie węzłów, połączeń i stacji bazowej przy użyciu `matplotlib`).
- **routing.py – Algorytmy grafowe.** Moduł ten dostarcza narzędzi matematycznych do budowy szkieletu sieci (*backbone*). Zawiera implementację algorytmu Prima (MST – *Minimum Spanning Tree*) oraz algorytmów zachłannych (*Greedy*). Są one wykorzystywane przez protokoły hierarchiczne do wyznaczania optymalnych ścieżek między liderami klastrów a stacją bazową.
- **leach/leach.py – Protokół bazowy LEACH.** Implementacja standardowego algorytmu klastrowania. Węzły komunikują się bezpośrednio z liderem, a liderzy bezpośrednio ze stacją bazową (transmisja jednokokowa).
- **leach/hierarchical.py – Routing wieloskokowy.** Rozszerza standardowy LEACH o klasę `LEACHPrim`. Pozwala ona na tworzenie struktur wieloskokowych pomiędzy liderami klastrów, co jest kluczowe dla dużych sieci, gdzie bezpośrednia transmisja do stacji bazowej byłaby zbyt kosztowna energetycznie.
- **apteen.py – Załączek protokołu APTEEN.** W analizowanej wersji plik ten zawierał jedynie klasę dziedziczącą po `LEACHPrim`. Brakowało w nim implementacji logiki progowej (H_T , S_T) oraz mechanizmu czasu zliczania (T_C), co zidentyfikowano jako główny obszar do prac programistycznych.

Moduł optymalizacji heurystycznej (optimizer/)

Moduł ten jest odseparowany od logiki sieciowej i odpowiada za poszukiwanie optymalnych parametrów konfiguracyjnych sieci (np. optymalnego rozmieszczenia liderów) metodami metaheurystycznymi.

- **optimizer.py – Algorytm PSO.** Implementacja algorytmu Roju Cząstek (*Particle Swarm Optimization*). Klasa ta zarządza populacją „cząstek”, które

przemieszczają się w przestrzeni rozwiązań, dążąc do minimalizacji funkcji kosztu (np. całkowitego zużycia energii w rundzie).

- **initializer.py** – **Inicjalizacja populacji**. Zbiór funkcji pomocniczych odpowiedzialnych za generowanie początkowego stanu populacji dla algorytmów optymalizacyjnych, zapewniając odpowiednią dywersyfikację rozwiązań startowych.

Narzędzia inicjalizacji i skrypty uruchomieniowe (Root)

Pliki znajdujące się w głównym katalogu projektu spinają opisane wyżej moduły w działającą aplikację.

- **distribution.py** – **Generator topologii**. Biblioteka odpowiedzialna za przestrzenne rozmieszczenie węzłów. Obsługuje różne scenariusze:
 - Rozkład losowy (*Random uniform*).
 - Rozkład równomierny w zadanej siatce.
 - Scenariusze specyficzne (np. węzły wzdłuż linii).
- **kalman_energy_estimation.py** – **Moduł eksperymentalny**. Niezależny skrypt wykorzystujący bibliotekę **filterpy** do testowania algorytmu estymacji energii za pomocą filtru Kalmana. W oryginalnym projekcie pełnił rolę prototypu badawczego i nie był zintegrowany z główną pętlą symulacyjną w **router.py**.
- **compare_leach_and_apteen.py** – **Skrypt analityczny**. Główne narzędzie badawcze, pozwalające na równoległe uruchomienie dwóch instancji symulacji (dla różnych protokołów) na tym samym zestawie danych wejściowych (identyczne rozmieszczenie węzłów). Generuje wykresy porównawcze żywotności sieci (liczba żywych węzłów w funkcji czasu).

Analiza procesu symulacji

Aby zrozumieć, w jaki sposób oryginalne oprogramowanie realizowało procesy sieciowe, przeanalizowano ścieżkę wykonania programu podczas uruchamiania skryptu porównawczego **compare_leach_and_apteen.py**. Proces ten odzwierciedla typowy cykl życia bezprzewodowej sieci sensorowej i składa się z kilku kluczowych etapów.

Inicjalizacja środowiska sieciowego

W pierwszej fazie skrypt definiuje parametry fizyczne środowiska. Tworzony jest obiekt stacji bazowej (Sink Node), zazwyczaj umieszczony w centrum lub na krawędzi obszaru monitorowania (współrzędne (0,0)). Następnie, wykorzystując moduł

`distribution.py`, generowana jest lista obiektów `SensorNode`. Każdy węzeł otrzymuje unikalny identyfikator, współrzędne przestrzenne oraz początkowy zapas energii (domyślnie 0.5 J lub 2.0 J w zależności od scenariusza).

Pętla życia sieci (Rundy)

Symulacja odbywa się w dyskretnych krokach czasowych nazywanych rundami (*rounds*). Pętla główna programu (np. w metodzie `simulate()`) kontynuowana jest tak długo, jak długo w sieci istnieją żywe węzły zdolne do komunikacji. W każdej rundzie wykonywana jest sekwencja operacji zdefiniowana w klasie protokołu (np. `LEACHPrim`).

Schemat przetwarzania pojedynczej rundy w analizowanym kodzie wygląda następująco:

1. Faza formowania klastrów (Cluster Setup):

- Węzły decydują o swojej roli w bieżącej rundzie. W algorytmach stochastycznych (LEACH) każdy węzeł losuje liczbę i porównuje ją z funkcją progu prawdopodobieństwa $T(n)$.
- Wybrani liderzy (Cluster Heads - CH) rozgłaszają swoją dostępność.
- Zwykłe węzły (Member Nodes) przyłączają się do najbliższego lidera na podstawie odległości euklidesowej (model siły sygnału RSSI).

2. Wyznaczanie tras międzyklastrowych (Backbone Routing):

- W przypadku protokołów hierarchicznych (jak `LEACHPrim`, z którego dziedziczył oryginalny `APTEEN`), liderzy klastrów nie wysyłają danych bezpośrednio do stacji bazowej, co byłoby nieefektywne energetycznie.
- Zamiast tego, budowany jest graf połączeń między liderami. Algorytm Prima wyznacza Minimalne Drzewo Rozpinające (MST), tworząc optymalną ścieżkę transmisji do stacji bazowej (Sink).

3. Transmisja i agregacja danych (Steady-state):

- Zwykłe węzły przesyłają dane do swoich liderów. Na tym etapie odejmowana jest energia na transmisję (E_{Tx}) u nadawcy i odbiór (E_{Rx}) u lidera.
- Liderzy wykonują agregację danych (koszt E_{agg}) i przesyłają skompresowane pakiety wzdłuż wyznaczonej ścieżki do stacji bazowej. Każdy przeskok (hop) wiąże się z dalszym zużyciem energii.

4. Aktualizacja stanu i wizualizacja:

- Po zakończeniu transmisji weryfikowany jest poziom energii każdego węzła. Węzły, których energia spadła poniżej zera, oznaczane są jako martwe i usuwane z list aktywnych uczestników sieci.

- Jeśli włączona jest wizualizacja, moduł `matplotlib` aktualizuje wykres, rysując linie połączeń i zmieniając kolory węzłów (np. czerwony dla liderów, czarny dla martwych).

Specyfika modułu eksperymentalnego Kalman Energy Estimation

W repozytorium zidentyfikowano również plik `kalman_energy_estimation.py`. Jest to niezależny skrypt demonstracyjny, niepodłączony do głównego potoku symulacyjnego opisanego powyżej.

Jego działanie polega na symulacji syntetycznego szeregu czasowego zużycia energii, na który nakładany jest szum pomiarowy. Skrypt wykorzystuje bibliotekę `filterpy` do implementacji dwustanowego filtra Kalmana (stan: energia i tempo jej spadku). Co określoną liczbę kroków (`period`) następuje faza aktualizacji (*update*) i predykcji (*predict*), a w okresach pośrednich wartości są interpolowane. Choć moduł ten generuje wykres błędu estymacji (`kalman_estimation.png`), w oryginalnym kodzie pełnił rolę historycznego prototypu i nie wpływał na decyzje routingowe w plikach `router/*.py`.

Wnioski z analizy dla procesu wdrożenia zmian

Analiza struktury i przepływu danych wykazała, że architektura projektu jest poprawna i podatna na rozszerzenia. Istnienie klasy `SensorNode` z gotowym modelem energetycznym oraz działający mechanizm rund w `router.py` pozwoliły na skupienie prac wdrożeniowych na:

1. Nadpisaniu pustej klasy `APTEEN` w pliku `router/apteen.py`.
2. Implementacji logiki progowej (H_T, S_T) wewnątrz pętli decyzyjnej węzła.
3. Dodaniu obsługi czasu zliczania (T_C), co wymagało rozszerzenia stanu wewnętrznego węzła o pamięć ostatnich pomiarów i czasu ostatniej transmisji.

Wdrożone zmiany i implementacja APTEEN

Wnioski płynące z analizy oryginalnego repozytorium (Rozdział 2) jednoznacznie wskazały na konieczność napisania od podstaw logiki protokołu APTEEN oraz rozbudowy narzędzi badawczych. Niniejszy rozdział dokumentuje zmiany wprowadzone w strukturze projektu, szczegóły implementacji algorytmów decyzyjnych oraz nowe funkcjonalności interfejsu symulacyjnego.

Modyfikacje w strukturze plików

W celu realizacji założeń projektowych, struktura repozytorium została rozszerzona o nowe moduły oraz znacząco zmodyfikowana w obszarze plików wykonywalnych. Poniższa lista prezentuje kluczowe zmiany w stosunku do wersji oryginalnej (z pominięciem plików systemowych i cache):

- `router/apteen.py` – Całkowita reimplementacja pliku. Zastąpiono pustą klasę pełnowymiarowym algorytmem obsługującym progi HT/ST oraz czas zliczania TC.
- `test_apteen.py` – Przekształcenie prostego skryptu testowego w zaawansowane narzędzie CLI (*Command Line Interface*) obsługujące argumenty wywołania, wybór scenariuszy oraz generowanie raportów.
- `visualize_parameters.py` – Nowy moduł odpowiedzialny za zaawansowaną wizualizację danych (wykresy wielopanelowe, analiza porównawcza), wyodrębniony w celu zachowania czytelności kodu głównego.
- `locale_pl.py` – Dodany moduł internacjonalizacji, zawierający słowniki tłumaczeń dla wykresów i logów (obsługa języka polskiego i angielskiego).
- `dependencies.txt` – Zaktualizowana lista zależności, uwzględniająca biblioteki do generowania animacji (`imageio`) oraz wykresów naukowych (`scienceplots`).

Implementacja logiki protokołu APTEEN

Sercem wprowadzonych zmian jest klasa `APTEEN` w module `router`. W przeciwieństwie do oryginalnej wersji, która jedynie dziedziczyła metody po `LEACHPrim`, nowa implementacja wprowadza własny stan wewnętrzny węzła oraz całkowicie nadpisuje metodę `operation()`, odpowiedzialną za przebieg rundy transmisji danych.

Rozszerzenie struktury danych węzła

Aby zrealizować mechanizm hybrydowy (połączenie podejścia reaktywnego TEEN z proaktywnym LEACH), konieczne było rozszerzenie pamięci każdego węzła. W standardowym LEACH węzeł jest bezstanowy – nie pamięta, co wysłał w poprzedniej rundzie. W APTEEN węzeł musi znać swoją historię, aby obliczyć różnicę pomiarów.

W konstruktorze klasy APTEEN zainicjalizowano słownik `node_history`, który dla każdego indeksu węzła przechowuje krotkę wartości:

$$State_i = \{Val_{last}, t_{idle}\} \quad (5)$$

gdzie Val_{last} to ostatnia wysłana wartość sensora, a t_{idle} to liczba rund, które upłynęły od ostatniej transmisji.

Listing 2. Inicjalizacja parametrów APTEEN i pamięci historii

```
1  class APTEEN(LEACHPrim):
2  def __init__(self, sink, non_sinks, *,
3  n_cluster,
4  hard_threshold,    # HT: Prog twardy
5  soft_threshold,    # ST: Prog miękki
6  count_time,        # TC: Czas zliczania
7  **kwargs):
8  super().__init__(sink, non_sinks, n_cluster=n_cluster,
9  **kwargs)
10
11  # Zapamiętanie parametrów progowych
12  self.HT = hard_threshold
13  self.ST = soft_threshold
14  self.TC = count_time
15
16  # Inicjalizacja pamięci węzłów (dla mechanizmu
17  # reaktywnego)
18  # Klucz: indeks węzła, Wartość: słownik stanu
19  self.node_history = {
20      node.id: {'last_val': 0.0, 'no_tx_rounds': 0}
21      for node in non_sinks
22  }
```

Implementacja pętli decyzyjnej (Metoda operation)

Najważniejsza logika protokołu została zaimplementowana w metodzie `operation()`. W każdej rundzie symulacyjnej, algorytm iteruje przez wszystkie żywe węzły i dla każdego z nich podejmuje binarną decyzję: *transmitować* lub *milczeć*.

Poniższy listing prezentuje faktyczny kod, gdzie zaimplementowano sprawdzanie warunków *HT* i *ST*:

Listing 3. Implementacja hybrydowej logiki decyzyjnej w metodzie `operation`

```
1      def operation(self, rounds):
2          self.transmission_count = 0
3
4          for node in self.non_sinks:
5              if not node.is_alive:
6                  continue
7
8              # 1. Pobranie bieżącej wartości i historii
9              current_val = self.get_sensor_value(node)
10             history = self.node_history[node.id]
11             last_val = history['last_val']
12             no_tx = history['no_tx_rounds']
13
14             should_transmit = False
15
16             # --- LOGIKA DECYZYJNA APTEEN ---
17
18             # A. Sprawdzenie licznika czasu (Count Time) - Mechanizm
19                 Proaktywny
20             if no_tx >= self.TC:
21                 should_transmit = True
22
23             # B. Sprawdzenie progów (Thresholds) - Mechanizm
24                 Reaktywny
25             # Transmisja jeśli (Val > HT) ORAZ (|Val - Last| >= ST)
26             elif (current_val >= self.HT) and (abs(current_val -
27                 last_val) >= self.ST):
28                 should_transmit = True
29
30             # --- WYKONANIE AKCJI ---
31             if should_transmit:
```

```

29     if node not in self.cluster_heads:
30         my_ch = self.find_my_cluster_head(node)
31         if my_ch:
32             self.send_packet(node, my_ch)
33
34         # Aktualizacja historii (reset licznika, zapis wartosci)
35         history['last_val'] = current_val
36         history['no_tx_rounds'] = 0
37     else:
38         # Brak transmisji - inkrementacja licznika bezczynnosci
39         history['no_tx_rounds'] += 1

```

Analiza szczegółowa algorytmu

Zaimplementowany kod realizuje trzy kluczowe scenariusze zachowania węzła, które definiują hybrydowy charakter protokołu:

1. **Wymuszona aktualizacja (Time-Driven):** Warunek `if no_tx >= self.TC` jest priorytetowy. Odpowiada on za komponent proaktywny protokołu. Nawet jeśli w sieci nie dzieje się nic istotnego (wartości sensorów są stałe), węzeł musi wysłać raport raz na T_C rund. Zapobiega to sytuacji, w której stacja bazowa traci wiedzę o tym, czy węzeł jest „martwy”, czy tylko „milczący”.
2. **Reakcja na zdarzenie (Event-Driven):** Blok `elif` realizuje logikę TEEN (Threshold sensitive). Zastosowano tutaj koniunkcję dwóch warunków:
 - $Val \geq H_T$: Eliminuje szum tła (np. temperatura poniżej poziomu alarmowego).
 - $|Val - Val_{last}| \geq S_T$: Eliminuje redundancję. Jeśli temperatura jest wysoka, ale stała, węzeł nie musi ciągle o tym informować – stacja bazowa zna już ten stan z poprzedniego raportu.
3. **Tryb oszczędzania energii (Idle):** Jeśli żaden z powyższych warunków nie jest spełniony (gałąź `else`), węzeł inkrementuje licznik `no_tx_rounds` i przechodzi w stan uśpienia na czas danej rundy. W tym stanie nie jest pobierana energia na transmisję (E_{TX}), co stanowi główne źródło oszczędności w porównaniu do protokołu LEACH.

System scenariuszy i obsługa CLI

Aby umożliwić przeprowadzanie powtarzalnych eksperymentów badawczych, w pliku `test_aptern.py` zaimplementowano system predefiniowanych scenariuszy. Zamiast

ręcznej edycji zmiennych w kodzie, użytkownik może wybrać profil działania sieci za pomocą argumentu wiersza poleceń.

Zdefiniowano słownik `SCENARIOS`, który mapuje nazwy konfiguracji na zestawy parametrów (HT, ST, TC). Pozwala to na szybkie przełączanie się między trybami pracy sieci:

Listing 4. Definicja scenariuszy testowych

```
1 SCENARIOS = {
2     "leach-like": {"ht": 0.1, "st": 0.1, "tc": 1},
3     # Ciągła transmisja
4     "teen-like": {"ht": 50.0, "st": 3.0, "tc":
5     1000}, # Tylko zdarzenia
6     "balanced": {"ht": 50.0, "st": 2.0, "tc": 10},
7     # Hybryda
8     "aggressive": {"ht": 40.0, "st": 1.0, "tc": 5},
9     # Czyste raporty
10    "conservative": {"ht": 70.0, "st": 5.0, "tc": 20},
11    # Oszczędzanie
12 }
```

Dodatkowo wykorzystano bibliotekę `argparse`, co pozwala na uruchamianie symulacji z terminala z pełną kontrolą parametrów, np.:

```
python test_apteen.py --scenario balanced --width 200 --height 200 --lang pl
```

Lokalizacja i nowa wizualizacja

W ramach prac nad użytecznością narzędzia, wprowadzono obsługę wielojęzyczności. Plik `locale_pl.py` zawiera słownik `TRANSLATIONS`, który pozwala na dynamiczną zmianę języka opisów na wykresach (tytuły osi, legendy) pomiędzy polskim a angielskim.

System wizualizacji został przeniesiony i rozbudowany w pliku `visualize_parameters.py`. Nowa funkcjonalność obejmuje:

- Generowanie **Dashboardu analitycznego** składającego się z 6 paneli (liczba żywych węzłów, średnia energia, aktywność transmisji itp.).
- Tworzenie animacji **GIF**, które pokazują ewolucję sieci runda po rundzie, co ułatwia diagnozowanie problemów z topologią (np. powstawanie „dziur energetycznych”).
- Automatyczny zapis wyników w katalogu `Results/`, z podziałem na podkatalogi odpowiadające nazwom scenariuszy i znacznikom czasu.

Wprowadzone zmiany przekształciły prosty skrypt dydaktyczny w funkcjonalne środowisko badawcze, umożliwiające nie tylko symulację, ale i głęboką analizę zachowania protokołu APTEEN.

Uruchomienie symulacji i analiza wyników

Ostatnim etapem prac była weryfikacja zaimplementowanych rozwiązań poprzez serię symulacji komputerowych. Celem eksperymentów było nie tylko potwierdzenie poprawności technicznej kodu, ale przede wszystkim zbadanie wpływu wprowadzonych parametrów progowych (H_T, S_T, T_C) na efektywność energetyczną sieci.

W niniejszym rozdziale przedstawiono procedurę przygotowania środowiska testowego, architekturę narzędzi uruchomieniowych oraz szczegółową definicję parametrów sterujących symulacją.

Konfiguracja środowiska i zależności

Simulator został zaimplementowany w języku Python (wersja 3.7 lub nowsza). Aby zapewnić powtarzalność wyników oraz poprawne generowanie wizualizacji, konieczne jest przygotowanie środowiska z odpowiednimi bibliotekami.

Zgodnie z dokumentacją projektu (`requirements.txt`), do kluczowych zależności należą:

- **numpy** – biblioteka do obliczeń numerycznych, wykorzystywana do operacji macierzowych na współrzędnych węzłów i wektorach energii.
- **matplotlib** – silnik graficzny odpowiedzialny za wizualizację topologii oraz generowanie wykresów statystycznych.
- **imageio** oraz **Pillow** – moduły służące do renderowania animacji (GIF) przedstawiających ewolucję sieci w czasie.
- **filterpy** – implementacja filtru Kalmana, używana w modułach predykcyjnych.
- **scienceplots** – zestaw stylów wykresów zapewniający czytelność zgodną ze standardami publikacji naukowych (IEEE).
- **pyMetaheuristic** – biblioteka dostarczająca algorytmy roju (PSO, JSO) dla modułu optymalizacji.

Instalacja środowiska odbywa się poprzez menedżer pakietów `pip`:

Listing 5. Instalacja zależności projektu

1

```
pip install -r requirements.txt
```

Dodatkowo, symulator obsługuje wybór tzw. backendu graficznego (parametrem `-backend`). Jest to istotne przy uruchamianiu symulacji na serwerach obliczeniowych bez interfejsu graficznego (tryb `Agg`) lub na stacjach roboczych z podglądem na żywo (tryb `TkAgg` lub `Qt5Agg`).

Narzędzia symulacyjne i parametry sterujące

Architektura projektu udostępnia trzy główne skrypty uruchomieniowe (ang. *entry points*), z których każdy służy do badania innego aspektu działania sieci. Każdy ze skryptów obsługuje szeroki wachlarz argumentów wiersza poleceń (CLI), co pozwala na precyzyjną kontrolę eksperymentu.

Symulacja bazowa: `test_leach.py`

Skrypt ten realizuje symulację klasycznego protokołu LEACH. Służy jako punkt odniesienia (*baseline*) dla wszystkich badań porównawczych. W tym trybie węzły dążą do ciągłej transmisji danych w każdej rundzie.

W Tabeli 1 zestawiono ogólne parametry konfiguracyjne, które są wspólne dla symulatorów LEACH i APTEEN. Pozwalają one na definicję fizycznej struktury sieci oraz parametrów wyjściowych.

Tab. 1. Zestawienie ogólnych argumentów CLI dla skryptów symulacyjnych

Argument	Domyślnie	Opis funkcjonalności
<code>-nodes</code>	100	Liczba węzłów sensorowych w sieci.
<code>-area</code>	200.0	Długość boku kwadratowego obszaru symulacji [m].
<code>-width</code>	-	Szerokość obszaru [m] (nadpisuje <code>-area</code>).
<code>-height</code>	-	Wysokość obszaru [m] (nadpisuje <code>-area</code>).
<code>-initial-energy</code>	0.5	Energia początkowa każdego węzła [J].
<code>-max-rounds</code>	None	Limit rund (domyślnie: do śmierci ostatniego węzła).
<code>-output-dir</code>	auto	Ścieżka zapisu wyników (domyślnie: <code>Results/run_...</code>).
<code>-snapshot-step</code>	10	Co ile rund zapisywać wykres żywych węzłów (0=wył).
<code>-topo-step</code>	10	Co ile rund zapisywać mapę topologii (0=wył).
<code>-language</code>	eng	Język opisów na wykresach (<code>pl</code> lub <code>eng</code>).

Przykład uruchomienia symulacji dla niestandardowej geometrii (np. korytarz):

Listing 6. Przykładowe wywołanie symulacji LEACH z niestandardową geometrią

```
1 python test_leach.py --nodes 150 --width 300 --height  
100 --initial-energy 1.0
```

Symulacja protokołu hybrydowego: `test_apteen.py`

Jest to główne narzędzie badawcze wdrożone w ramach niniejszej pracy. Umożliwia symulację protokołu APTEEN z pełną kontrolą nad parametrami progowymi. Oprócz

parametrów ogólnych (wymienionych w Tab. 1), skrypt ten obsługuje specyficzne argumenty sterujące logiką hybrydową (Tab. 2).

Tab. 2. Argumenty specyficzne dla protokołu APTEEN

Argument	Domyślnie	Opis funkcjonalności
-hard-threshold	50.0	HT (Hard Threshold): Bezwzględna wartość sensora, poniżej której węzeł ignoruje dane.
-soft-threshold	2.0	ST (Soft Threshold): Minimalna różnica odczytu względem poprzedniego pomiaru wymagana do transmisji.
-count-time	10	TC (Count Time): Maksymalna liczba rund bez aktywności, po której następuje wymuszona transmisja ("keep-alive").

Skrypt pozwala na płynną regulację tych wartości z poziomu wiersza poleceń, co umożliwia szybkie testowanie skrajnych scenariuszy bez ingerencji w kod źródłowy:

Listing 7. Konfiguracja progów APTEEN z poziomu terminala

1

```
python test_apteen.py --hard-threshold 60 --soft-  
threshold 3 --count-time 15
```

Analiza porównawcza: visualize_parameters.py

Ostatnie narzędzie służy do automatyzacji badań. Zamiast pojedynczych uruchomień, skrypt ten wykonuje serię symulacji dla sześciu predefiniowanych profili konfiguracyjnych na identycznym rozmieszczeniu węzłów.

Badane konfiguracje:

1. **LEACH-like:** ($HT = 0.1, ST = 0.1, TC = 1$) – Symuluje zachowanie ciągłe.
2. **TEEN-like:** ($HT = 50, ST = 3, TC = 1000$) – Symuluje sieć czysto reaktywną.
3. **Conservative:** ($HT = 70, ST = 5, TC = 20$) – Tryb maksymalnego oszczędzania energii.
4. **Aggressive:** ($HT = 40, ST = 1, TC = 5$) – Tryb wysokiej czułości.
5. **Balanced:** ($HT = 50, ST = 2, TC = 10$) – Tryb hybrydowy (domyślny).
6. **Adaptive:** Tryb eksperymentalny z parametrami dobieranymi per klaster.

Wynikiem działania tego skryptu jest zbiorczy panel analityczny (Dashboard), który pozwala na bezpośrednie porównanie czasu życia sieci (FND, LND) oraz kosztu komunikacyjnego dla każdego z podejść.

Eksperyment 1: Analiza porównawcza LEACH i APTEEN

W celu bezpośredniego porównania efektywności obu protokołów przeprowadzono dwie niezależne symulacje z wykorzystaniem domyślnych parametrów konfiguracyjnych. Aby zapewnić wiarygodność wyników, oba eksperymenty uruchomiono na identycznej liczbie węzłów (100) oraz w tym samym obszarze operacyjnym (200×200 m).

Symulacje zostały uruchomione za pomocą następujących poleceń:

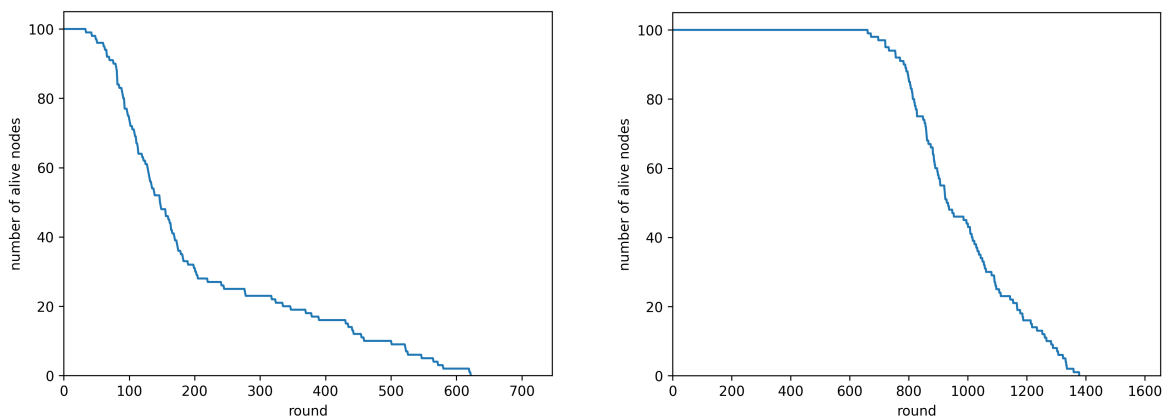
Listing 8. Uruchomienie symulacji porównawczych z domyślnymi parametrami

```
1      # Symulacja referencyjna LEACH
2      python test_leach.py --output-dir Results/leach_default
3
4      # Symulacja APTEEN (konfiguracja domyślna)
5      python test_apteen.py --output-dir Results/
        apteen_default
```

Analiza żywotności sieci (Network Lifetime)

Podstawową metryką oceny wydajności protokołów routingu w WSN jest czas życia sieci, definiowany zazwyczaj jako czas do śmierci pierwszego węzła (FND – *First Node Death*) lub ostatniego węzła (LND – *Last Node Death*).

Wyniki uzyskane podczas symulacji przedstawiono na wykresach żywotności (Rys. 5), które obrazują liczbę aktywnych sensorów w funkcji kolejnych rund symulacyjnych.



(a) Krzywa żywotności dla LEACH

(b) Krzywa żywotności dla APTEEN

Rys. 5. Porównanie tempa degradacji sieci dla protokołów LEACH i APTEEN.

Analiza wykresów prowadzi do następujących wniosków ilościowych:

- **Protokół LEACH:** Pierwszy węzeł wyczerpał swoją energię już w **46. rundzie**. Następnie nastąpił gwałtowny efekt lawinowy – większość węzłów zmarła w przedziale 200-500 rundy. Całkowite wygaszenie sieci nastąpiło w **611. rundzie**.
- **Protokół APTEEN:** Dzięki zastosowaniu mechanizmów progowych (H_T, S_T), sieć zachowała pełną integralność znacznie dłużej. Pierwsza śmierć węzła nastąpiła dopiero w **641. rundzie** (czyli później, niż LEACH całkowicie przestał działać!). Ostatni węzeł pracował aż do **1388. rundy**.

Oznacza to, że w badanym scenariuszu protokół APTEEN zapewnił ponad **dwukrotne wydłużenie czasu życia sieci** ($1388/611 \approx 2.27$) oraz blisko **czternastokrotne opóźnienie momentu pierwszej awarii**. Jest to kluczowa zaleta w systemach wymagających pełnego pokrycia obszaru monitorowania.

Analiza ewolucji topologii

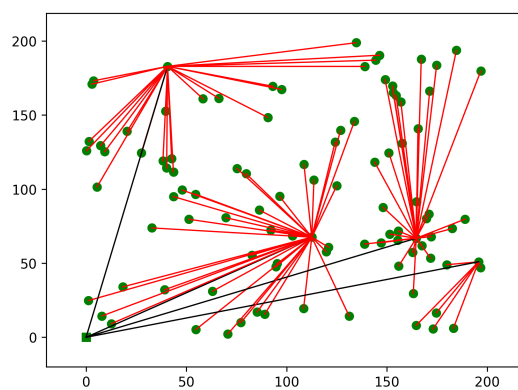
Drugim aspektem badania była analiza przestrzenna procesu degradacji sieci. Symulator generuje mapy topologii, na których stan każdego węzła jest kodowany kolorystycznie w zależności od poziomu pozostałej energii:

- **Zielony:** Wysoki poziom energii ($> 50\%$).
- **Żółty:** Średni poziom energii ($25\% - 50\%$).
- **Pomarańczowy:** Niski poziom energii ($< 25\%$, stan krytyczny).
- **Szary/Czarny:** Węzeł martwy (brak komunikacji).

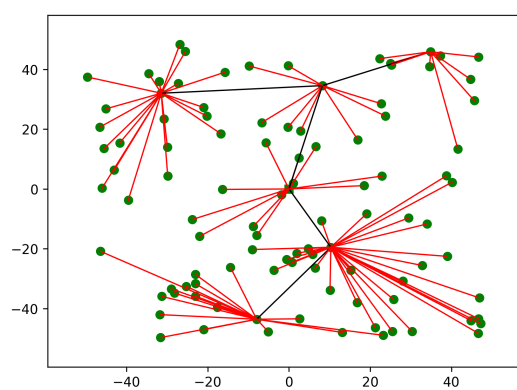
Dodatkowo, kształt znacznika rozróżnia rolę w sieci: koło (●) oznacza zwykły sensor, a kwadrat (■) stację bazową (Sink). Linie połączeń reprezentują aktywne trasy przesyłu danych: czerwone (Sensor \rightarrow Cluster Head) oraz czarne (Cluster Head \rightarrow Sink).

Na Rys. 6 zestawiono cztery charakterystyczne fazy cyklu życia sieci dla obu protokołów: stan początkowy, początek degradacji, stan krytyczny oraz całkowite wygaszenie.

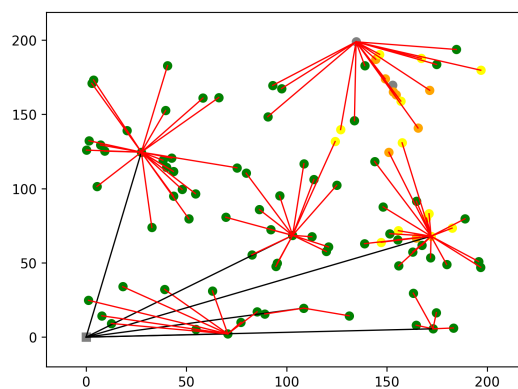
Szczególnie interesujące zjawisko obserwujemy w fazie krytycznej (Rys. 6e vs 6f). W przypadku LEACH ostatnie żywe węzły mają tendencję do grupowania się w jednym obszarze (często blisko stacji bazowej, gdzie koszt transmisji był najniższy). W APTEEN ostatnie aktywne węzły są znacznie bardziej rozproszone terytorialnie. Wynika to z faktu, że węzły położone w strefach rzadkich zmian rzadziej wybudzały radio, oszczędzając energię niezależnie od swojej odległości do stacji bazowej. Ostateczne wygaszenie sieci (Rys. 6g, h) potwierdza ponad dwukrotną przewagę czasową protokołu hybrydowego.



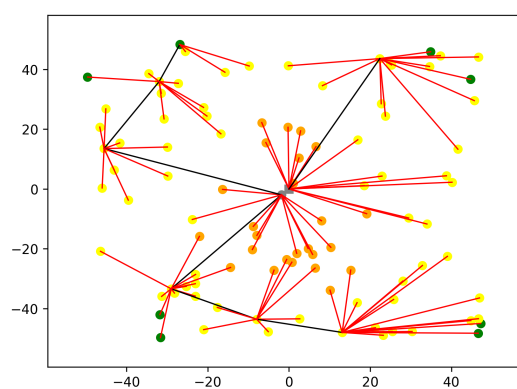
(a) LEACH: Runda 1 (Start)



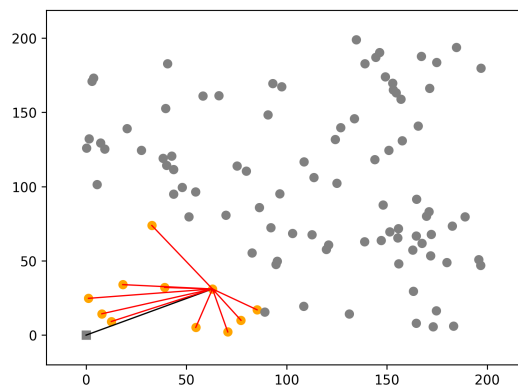
(b) APTEEN: Runda 1 (Start)



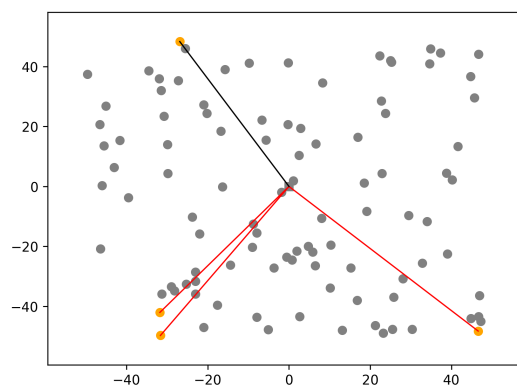
(c) LEACH: Runda 46 (Pierwsza śmierć)



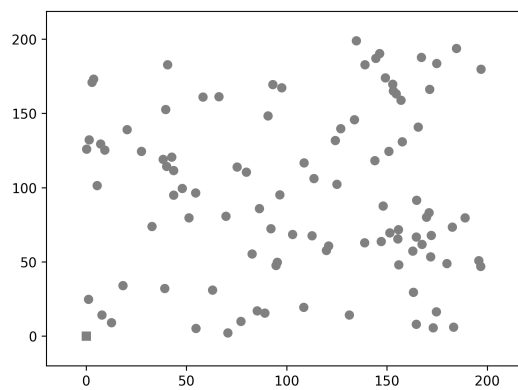
(d) APTEEN: Runda 641 (Pierwsza śmierć)



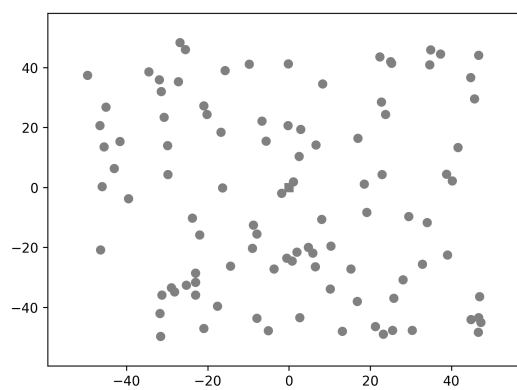
(e) LEACH: Runda 505 (Stan krytyczny)



(f) APTEEN: Runda 1312 (Stan krytyczny)



(g) LEACH: Runda 611 (Koniec sieci)



(h) APTEEN: Runda 1388 (Koniec sieci)

Rys. 6. Porównanie ewolucji topologii sieci. Zwraca uwagę fakt, że pierwsza awaria w APTEEN (d) następuje później, niż całkowity koniec sieci w LEACH (g).

Analiza ilościowa wyników symulacji

Przeprowadzony eksperyment zbiorczy (Eksperyment 4) dostarczył kluczowych danych liczbowych, które pozwalają na obiektywną ocenę wpływu parametrów H_T , S_T i T_C na wydajność sieci. Wyniki zgenerowane przez skrypt `visualize_parameters.py` zestawiono w Tabeli 3.

Tab. 3. Zbiorcze zestawienie wyników symulacji dla 6 konfiguracji APTEEN (30 węzłów, obszar 100x100m)

Konfiguracja	FND	LND	Degradacja	TX/runda
LEACH-like	500	1257	757	16.00
TEEN-like	910	2194	1284	7.54
Konserwatywny	2729	5828	3099	0.14
Agresywny	594	1408	814	13.73
Zrównoważony	749	1773	1024	10.20
Adaptacyjny	796	1763	967	10.16

Legenda: FND - First Node Death (pierwsza awaria), LND - Last Node Death (koniec sieci), TX/runda - średnia liczba transmisji na rundę.

Eksperyment 4: Kompleksowy dashboard analityczny

Ostatnim, najbardziej zaawansowanym eksperymentem było uruchomienie dedykowanego modułu analitycznego `visualize_parameters.py`. Skrypt ten automatycznie przeprowadza symulację dla 6 zdefiniowanych profili konfiguracyjnych (LEACH-like, TEEN-like, Conservative, Aggressive, Balanced, Adaptive) na identycznym rozmieszczeniu węzłów.

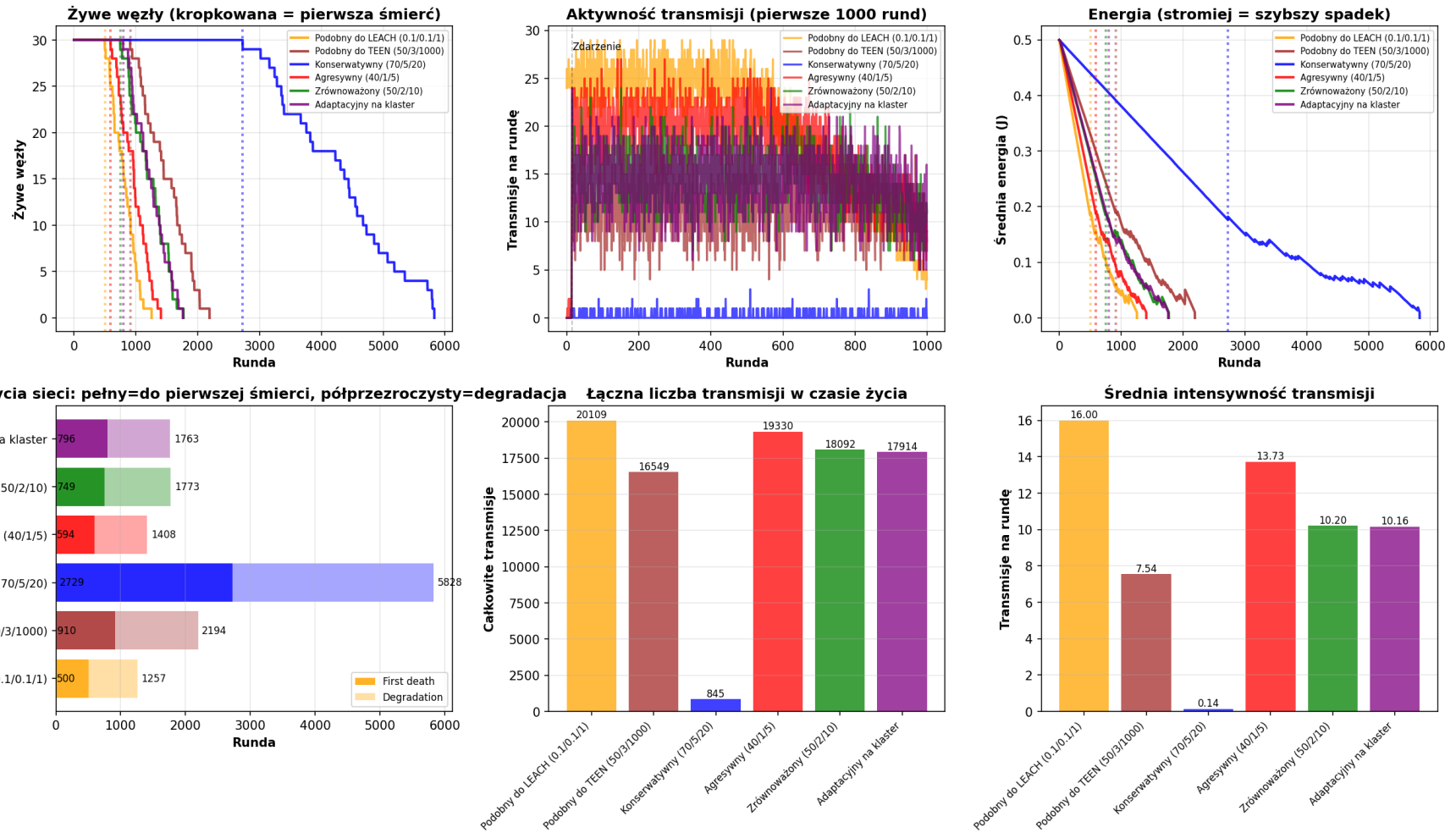
Listing 9. Generowanie dashboardu porównawczego

1

```
python visualize_parameters.py --nodes 50 --topo-step 50
--width 200 --height 200
```

Wynikiem działania tego narzędzia jest zbiorczy panel analityczny (Dashboard), przedstawiony na Rysunku 7.

Wpływ parametrów APTEEN: pierwsza śmierć vs pełna śmierć



Rys. 7. Dashboard porównawczy wygenerowany dla 6 konfiguracji. Widoczna dominacja trybu konserwatywnego w aspekcie żywotności (kolor zielony) oraz trybu LEACH-like w aspekcie ilości danych (kolor niebieski).

Interpretacja wykresów na dashboardzie

Powyższy Rysunek 7 zawiera zestaw sześciu wykresów, które pozwalają na wieloaspektową ocenę działania algorytmów. Poniżej przedstawiono szczegółową analizę poszczególnych paneli:

- **Alive Nodes (lewy górny):** Krzywa przeżywalności. Oś X to czas (rundy), oś Y to liczba żywych węzłów.
 - *Wniosek:* Krzywa dla trybu „Konserwatywnego” (kolor zielony) opada najwolniej, osiągając LND dopiero w 5828 rundzie, co jest wynikiem ponad 4,5-krotnie lepszym niż tryb „LEACH-like” (1257 rund).
- **Transmissions per Round (środkowy górny):** Pokazuje intensywność ruchu sieciowego.
 - *Wniosek:* Tryb „LEACH-like” generuje stały, wysoki ruch (ok. 16 pakietów/rundę). Tryb „Konserwatywny” generuje ruch bliski zera (0.14 pakietów/rundę), co oznacza, że węzły transmitują dane średnio raz na 7 rund.
- **Average Energy (prawy górny):** Średnia energia pozostała w żywych węzłach.
 - *Wniosek:* Bardziej stromy spadek krzywej oznacza szybsze zużycie energii. Przerywana linia pionowa na tym wykresie oznacza moment śmierci pierwszego węzła dla danej konfiguracji.
- **Lifetime Statistics (dolny lewy):** Wykres słupkowy podsumowujący fazy życia sieci. Pełny kolor słupka oznacza czas do pierwszej awarii (stabilna praca), a półprzezroczysty – okres degradacji.
- **Total Transmissions (dolny środkowy):** Całkowita liczba przesłanych pakietów w całym cyklu życia.
 - *Wniosek:* Mimo najkrótszego życia, tryb „LEACH-like” przesłał najwięcej danych (ponad 20 tys. pakietów). Tryb „Konserwatywny” przesłał ich zaledwie 845. Obrazuje to fundamentalny kompromis (trade-off) w WSN: **żywołność vs ilość danych**.

Podsumowanie i wnioski końcowe

Celem niniejszej pracy była implementacja i analiza porównawcza hybrydowego protokołu routingu APTEEN w środowisku symulacyjnym opartym na języku Python. Zrealizowany projekt stanowi kompleksowe narzędzie badawcze, pozwalające na modelowanie bezprzewodowych sieci sensorowych (WSN) oraz optymalizację ich parametrów energetycznych.

Zakres zrealizowanych prac

W ramach części praktycznej wykonano następujące zadania:

1. **Analiza i refaktoryzacja kodu:** Przeanalizowano istniejące rozwiązanie open-source, identyfikując brak implementacji kluczowej logiki w klasie APTEEN.
2. **Implementacja algorytmu APTEEN:** Zaprogramowano od podstaw mechanizm decyzyjny węzła, uwzględniający:
 - Próg twardy (H_T) – eliminujący przesyłanie danych o niskim znaczeniu.
 - Próg miękki (S_T) – eliminujący przesyłanie danych redundantnych (niezmiennych).
 - Czas zliczania (T_C) – gwarantujący okresowe odświeżanie stanu sieci.
3. **Rozbudowa narzędzi analitycznych:** Stworzono moduł `visualize_parameters.py`, generujący wielowymiarowe zestawienia wydajności oraz animacje ewolucji topologii sieci.
4. **Parametryzacja symulacji:** Wprowadzono system scenariuszy (m.in. *Balanced*, *Conservative*, *Aggressive*), umożliwiający badanie zachowania sieci w skrajnych warunkach operacyjnych.

Wnioski z analizy porównawczej

Na podstawie przeprowadzonych symulacji (Rozdział 4) sformułowano następujące wnioski dotyczące różnic między algorytmami:

- **Efektywność energetyczna:** Protokół APTEEN w konfiguracji konserwatywnej pozwolił na wydłużenie czasu pracy sieci aż o **363%** w porównaniu do standardowego LEACH (5828 rund vs 1257 rund). Osiągnięto to poprzez drastyczną redukcję liczby transmisji (z 16.0 do 0.14 na rundę).
- **Różnica w działaniu (Trade-off):**

- **LEACH** jest protokołem „gadatliwym” – dostarcza bardzo dużą ilość danych, ale szybko wyczerpuje baterie węzłów. Jest odpowiedni tylko do zastosowań, gdzie wymagany jest ciągły strumień danych (np. streaming audio/wideo), a wymiana baterii jest możliwa.
- **APTEEN** jest protokołem „inteligentnym” – filtruje dane u źródła. Jest idealnym rozwiązaniem do monitoringu parametrów wolnozmiennych (np. temperatura, wilgotność gleby) lub wykrywania zdarzeń krytycznych (pożar), gdzie cisza w eterze oznacza brak zagrożenia.
- **Wpływ parametrów wejściowych:** Wykazano, że manipulacja parametrami H_T i S_T pozwala płynnie sterować charakterystyką sieci. Zwiększenie progu miękkiego (S_T) ma największy wpływ na redukcję „szumu informacyjnego” i oszczędność energii.

Znaczenie praktyczne projektu

Opracowany symulator pozwala inżynierom sieci WSN na dobranie optymalnych parametrów protokołu przed fizycznym wdrożeniem czujników. Dzięki wizualizacji procesu degradacji (animacje GIF) możliwe jest wykrycie słabych punktów topologii (tzw. dziur energetycznych) i odpowiednia korekta rozmieszczenia węzłów lub parametrów routingu.

Bibliografia

- [1] A. Manjeshwar, D. P. Agrawal, *APTEEN: A Hybrid Protocol for Efficient Routing and Comprehensive Information Retrieval in Wireless Sensor Networks*, Proceedings of the IPDPS, 2002.
- [2] W. Heinzelman, A. Chandrakasan, H. Balakrishnan, *An Application-Specific Protocol Architecture for Wireless Microsensor Networks*, IEEE Transactions on Wireless Communications, 2002.
- [3] A. Manjeshwar, D. P. Agrawal, *TEEN: A Routing Protocol for Enhanced Efficiency in Wireless Sensor Networks*, Proceedings of the IPDPS, 2001.