Natalia Szczupak

Porównanie algorytmów HEED i LEACH

Administracja systemów rozproszonych – Projekt nr 1

Opiekun pracy
mgr inż. Dariusz Rączka

Rzeszów, 6.05.2025r.

Spis treści

1. Wprowadzenie do algorytmu HEED ... 2
2. Schemat blokowy ... 3
3. Pseudokod ... 4
4. Zastosowania algorytmu HEED ... 5
5. Wprowadzenie do algorytmu LEACH ... 6
6. Fazy w podstawowym algorytmie LEACH ... 6
7. Podstawy algorytmu ... 6
8. Obliczanie zużycia energii ... 7
9. Schemat blokowy ... 8
10. Pseudokod ... 9
11. Zastosowania algorytmu LEACH ... 9
12. Implementacja algorytmu w Arduino ... 11
13. Opis algorytmu ... 18
14.1 Struktura kodu: ... 18
14.2 Algorytmy: ... 19
14.3 Działanie głównej pętli programu: ... 19
14. Porównanie obu algorytmów ... 20
15.1 Podobieństwa: ... 20
15.2 Główna różnica: ... 20
15. Wnioski ... 20
16. Bibliografia ... 21

1. Wprowadzenie do algorytmu HEED

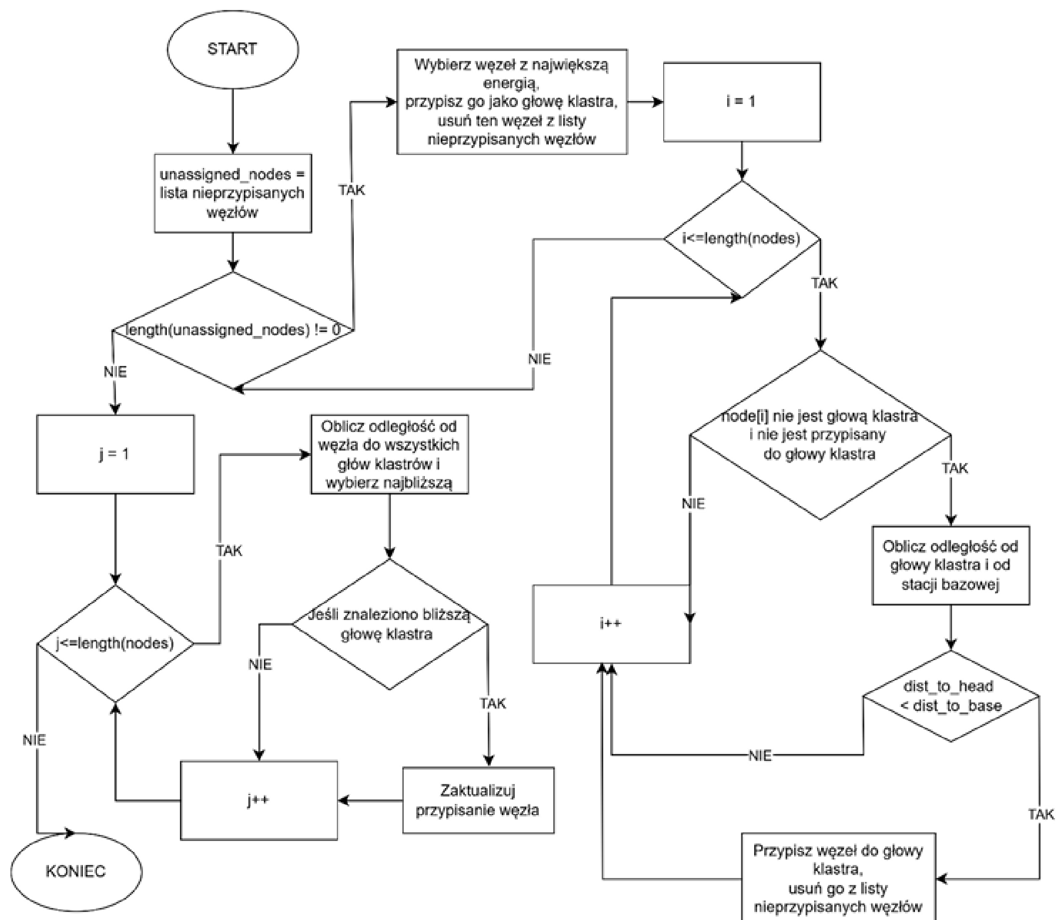
Algorytm HEED (Hybrydowy, Energooszczędny, Rozproszony) jest protokołem klastrowania o równomiernym podziale, który tworzy klastry o równych rozmiarach. Wybór liderów klastrów (CH) w HEED jest oparty na pozostałej energii węzłów czujnikowych oraz jednym z następujących parametrów: stopniu węzła (liczba sąsiadów) lub odległości sąsiednich węzłów od liderów klastrów (CH).
Tworzenie klastrów w HEED przebiega w trzech fazach: inicjalizacji, iteracji i finalizacji.
W fazie inicjalizacji każdemu węzłowi przypisuje się prawdopodobieństwo zostania tymczasowym liderem klastra. Jest ono obliczane zgodnie z następującym wzorem:

$$CH = C_p \cdot \frac{E_r}{E_m}$$

CH - prawdopodobieństwo zostania tymczasowym liderem
C_p - początkowe prawdopodobieństwo (wartość wstępnie zdefiniowana)

E_r - pozostała energia w węźle
E_m - maksymalna energia dostępna dla węzłów czujnikowych.
W fazie iteracyjnej HEED niektóre węzły stają się tymczasowymi liderami klastrów. Gdy węzeł znajduje się w zasięgu komunikacyjnym niektórych tymczasowych liderów klastrów, wybierze ten z najmniejszym kosztem. Jeśli węzeł nie jest w zasięgu komunikacyjnym żadnego tymczasowego lidera klastra, ostatecznie sam zostanie liderem klastra. W fazie finalizacji, węzły, które nie wybrały żadnego lidera klastra, stają się liderami.
HEED generuje klastry o równomiernych rozmiarach. Rozmiar klastra jest niezależny od odległości do stacji bazowej (BS). Ze względu na komunikację międzyklastrową (np. ruch przekazywany między liderami klastrów), węzły znajdujące się blisko stacji bazowej szybciej zużywają swoją energię. Dzieje się tak, ponieważ oprócz standardowej komunikacji wewnątrzklastrowej (czyli ruchu wewnątrz klastra), są one dodatkowo obciążone ruchem przekazywanym od pozostałych węzłów w sieci. Ten ruch przekazywany nie tylko wpływa na żywotność sieci, ale także prowadzi do podziałów sieci (blisko stacji bazowej).
Nasza implementacja stanowi zmodyfikowaną wersję oryginalnego algorytmu HEED. Wprowadzone zmiany mają na celu zwiększenie efektywności działania oraz optymalizację procesów klasteryzacji w sieci. Dzięki zastosowanym ulepszeniom, algorytm lepiej wykorzystuje zasoby energetyczne węzłów oraz poprawia komunikację między nimi, co czyni go bar dziej wydajnym w porównaniu do pierwotnej wersji.

2. Schemat blokowy



3. Pseudokod

K01: unassigned_nodes <- lista wszystkich, nieprzypisanych węzłów
K02: dopóki length(unassigned_nodes) <> 0 wykonuj kroki K03-K12
K03: wybierz węzeł z największą energią
K04: przypisz go jako głowę klastra
K05: usuń ten węzeł z unassigned_nodes
K06: i <- 1
K07: dopóki i <= length(nodes) wykonuj kroki K08-K12
K08: jeśli node[i] nie jest głową klastra i nie jest przypisany do głowy klastra wtedy idź do kroku K09, w przeciwnym wypadku i <- i+1 i wróć do kroku K07
K09: Oblicz odległość od głowy klastra i stacji bazowej
K10: jeśli dist_to_head<dist_to_base wtedy idż do kroku K11, w przeciwnym wypadku i <- i+1 oraz wróć do kroku K07
K11: Przypisz węzeł do głowy klastra
K12: usuń ten węzeł z unassigned_nodes
K13: j <- 1

K14: Dopóki j<=length(nodes) wykonuj kroki K15-K16
K15: Oblicz odległość od węzła do wszystkich głów klastrów i wybierz najbliższą
K16: Jeśli znaleziono bliższą głowę klastra to przejdź do kroku K17, w przeciwnym wypadku j<-j+1 oraz wróć do kroku K14
K17: Zaktualizuj przypisanie węzła
K18: Zakończ algorytm

4. Zastosowania algorytmu HEED

Monitoring środowiskowy – HEED jest używany w sieciach sensorowych do monitorowania obszarów naturalnych, takich jak lasy, jeziora czy góry. Sensory zbierają in 4 formacje o temperaturze, wilgotności, jakości powietrza lub obecności zanieczyszczeń. Dzięki klastrowaniu i efektywnemu zarządzaniu energią, sieć może działać przez długi czas bez konieczności wymiany baterii.

Rolnictwo precyzyjne – W zastosowaniach rolniczych HEED pomaga monitorować wilgotność gleby, nasłonecznienie, temperaturę oraz inne czynniki wpływające na wzrost roślin. Sensory rozlokowane na dużych polach przekazują dane do centralnych jednostek, a HEED zapewnia energooszczędne i niezawodne przesyłanie informacji.

Monitorowanie infrastruktury miejskiej – Algorytm HEED stosowany jest w miejskich sieciach sensorowych do monitorowania stanu infrastruktury, jak mosty, drogi, budynki czy tunele. Dzięki efektywnemu zarządzaniu energią, węzły sensorowe mogą dłużej monitorować strukturę i zgłaszać wszelkie nieprawidłowości, co zwiększa bezpieczeństwo i ułatwia zarządzanie infrastrukturą.

Monitorowanie zdrowia pacjentów – W zastosowaniach medycznych HEED jest wykorzystywany w sieciach sensorowych do monitorowania parametrów zdrowotnych pacjentów w czasie rzeczywistym, np. rytmu serca, ciśnienia krwi czy poziomu cukru. Dzięki oszczędzaniu energii sensory te mogą działać dłużej, co jest kluczowe w przypadku pacjentów wymagających stałego nadzoru.

Aplikacje wojskowe – HEED znajduje zastosowanie w wojsku, gdzie sieci sensorowe są używane do monitorowania ruchu, aktywności czy innych zjawisk w obszarach o ograniczonym dostępie. Sieci te są rozmieszczane na dużych, trudnych terenach, więc efektywne zarządzanie energią pozwala na ich długotrwałe funkcjonowanie bez konieczności częstego serwisowania.

Inteligentne miasta (Smart City) – W inteligentnych miastach HEED może być używany do zarządzania sieciami sensorów, które monitorują zużycie energii, temperaturę, jakość powietrza czy obecność osób w pomieszczeniach. Dzięki temu algorytmowi urządzenia te mogą działać oszczędnie, co zmniejsza zapotrzebowanie na energię i wydłuża ich czas działania.

HEED pozwala na efektywne zarządzanie zasobami i zapewnia długoterminowe działanie sieci sensorowych, co czyni go idealnym rozwiązaniem w zastosowaniach, gdzie konserwacja jest trudna lub kosztowna, a żywotność baterii kluczowa.

5. Wprowadzenie do algorytmu LEACH

Low-Energy Adaptive Clustering Hierarchy (LEACH) jest protokołem MAC opartym na TDMA, który jest zintegrowany z grupowaniem i prostym protokołem routingu w bezprzewodowych sieciach sensorowych (WSN). Celem LEACH jest obniżenie zużycia energii wymaganego do tworzenia i utrzymywania klastrów w celu poprawy trwałości sieci bezprzewodowej.
LEACH jest protokołem hierarchicznym, w którym większość węzłów transmituje do głów klastrów, a głowy klastrów agregują i kompresują dane, przekazując je następnie do stacji bazowej (sink). Każdy węzeł stosuje algorytm stochastyczny w każdej rundzie, aby określić, czy stanie się głową klastra w tej rundzie. LEACH zakłada, że każdy węzeł ma radio wystarczająco silne, aby bezpośrednio dotrzeć do stacji bazowej lub najbliższej głowy klastra, ale używanie tego radia na pełnej mocy przez cały czas powodowałoby stratę energii. Węzły, które były głowami klastrów, nie mogą ponownie zostać głowami klastrów przez P rund, gdzie P to pożądany procent głów klastrów. Następnie każdy węzeł ma 1/P prawdopodobieństwo ponownego zostania głową klastra. Na końcu każdej rundy każdy węzeł, który nie jest głową klastra, wybiera najbliższą głowę klastra i dołącza do tego klastra. Głowa klastra tworzy następnie harmonogram dla każdego węzła w swoim klastrze, aby ten wysłał swoje dane. Wszystkie węzły, które nie są głowami klastrów, komunikują się tylko z głową klastra w sposób TDMA, zgodnie z harmonogramem utworzonym przez głowę klastra. Robią to, używając minimalnej energii potrzebnej do dotarcia do głowy klastra i muszą utrzymywać włączone radio tylko w czasie swojego przedziału czasowego. LEACH stosuje również CDMA, dzięki czemu każdy klaster używa innego zestawu kodów CDMA, aby zminimalizować zakłócenia między klastrami.

6. Fazy w podstawowym algorytmie LEACH

Algorytm LEACH opiera się na cyklicznej zmianie ról węzłów w ramach danego klastra. Każdy klaster składa się z węzłów, które zbierają i mogą wstępnie przetwarzać dane, oraz głów klastrów, które pełnią funkcję agregowania danych. Proces cyklicznej zmiany ról węzłów odbywa się poprzez dwie fazy: fazę konfiguracji oraz fazę stabilnego działania. W fazie konfiguracji wybierana jest głowa klastra, a następnie rozpoczyna się faza stabilna, której długość zależy od liczby cykli pomiarowych. Po zakończeniu pomiarów i przesłaniu przetworzonych danych do stacji bazowej, proces przechodzi do kolejnej fazy konfiguracji (lub re-konfiguracji), w której może być wybrana nowa głowa klastra, a także mogą być tworzone nowe klastry, jeżeli będzie to konieczne.

7. Podstawy algorytmu

Niech N oznacza zbiór węzłów n. W fazie konfiguracji rozpoczyna się proces wyboru głów klastrów. W tym celu dla każdego węzła losowana jest wartość z zakresu 0-1. Następnie jest porównywana z wartością progową $T(n)$:

$$T(n) = \begin{cases} p \cdot \left(1 - p \cdot \left(r \, mod \frac{1}{p}\right)\right) jeśli \; n \in G \\ 0 \qquad\qquad w \; przeciwnym \; razie \end{cases}$$ [1]

Gdzie:
p – pożądany procent liczby głów klastrów w sieci (np. 0,05),
r – numer aktualnej rundy (rozpoczynając od rundy 0),
G – zbiór węzłów, które nie pełniły funkcji głowy w ostatnich 1/p rundach, dzięki czemu każdy węzeł stanie się głową klastra w ciągu 1/p rund.

8. Obliczanie zużycia energii

Zużycie energii w sieci jest obliczane na podstawie różnych procesów, takich jak transmisja danych, odbiór oraz nawiązywanie połączeń, które mogą się zmieniać w zależności od odległości między węzłami. Węzeł, który przesyła dane, ponosi koszt energetyczny związany z tą transmisją, oznaczony jako ET. Koszt ten jest obliczany według poniższego wzoru:

$$E_T = \begin{cases} E_T \cdot X \cdot L + E_{mp} \cdot L \cdot d^4 \; jeśli \; d \leq d_0 \\ E_T \cdot X \cdot L + E_{fs} \cdot L \cdot d^2 \; jeśli \; d > d_0 \end{cases}$$ [2]

gdzie:
E_T – zużycie energii na bit przesyłanych danych,
L – ilość przesyłanych danych,
E_{mp} /E_{fs} – zależy od modelu wzmacniacza nadajnika, E_{fs} jest stosowane dla modelu przestrzeni wolnej, natomiast E_{mp} dla modelu wielościeżkowego,
d – odległość euklidesowa między nadajnikiem a odbiornikiem
d_0 – próg odległości obliczany na podstawie wzoru (5). Założone jednostki fizyczne to dżule i metry dla odległości.

Z kolei węzeł odbierający dane jest obciążony kosztem energii za odbiór danych oznaczonym jako ER, który obliczany jest zgodnie ze wzorem:

$$E_R = (E_{RX} + E_{DA}) \cdot L$$ [3]

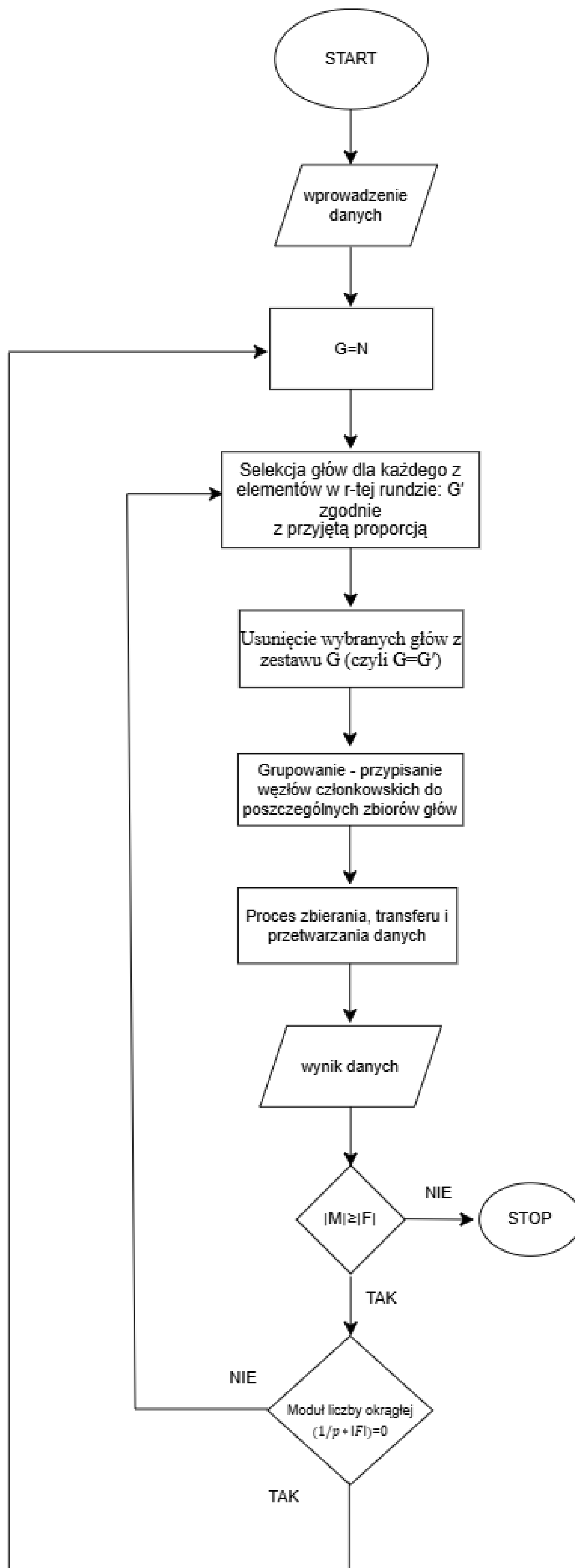
Gdzie:
E_{RX} – zużycie mocy na bit przy odbiorze danych,
E_{DA} – zużycie energii podczas procesu fuzji danych.

$$d_0 = \sqrt{\frac{E_{fs}}{E_{mp}}}$$ [4]

[1] A.Hazra, P. Rana, M.Adhikari, and T. Amgoth, “Fogcomputing for next-generation Internet of Things: Fundamental, state-of the-art and research challenges,” Comput. Sci. Rev.
[2] V.K.Prasad,M.D.Bhavsar,andS.Tanwar,“InfluenceofMontor ing: Fog and Edge Computing,” Scalable Comput.-Pract. Exp., vol. 20,no.2,pp.365–376,2019,doi:10.12694/scpe.v20i2.1533.
[3] R. Basir, N.A. Chughtai, M. Ali, S. Qaisar, and A. Hashimi, “Mode selection, caching and physical layer security for fog net works,” Bull. Pol. Acad. Sci. Tech. Sci., vol. 70,

[4] S. Chen and L. Tang, “Flexible English Learning Platform using Collaborative Cloud-Fog-Edge Networking,” Scalable Comput. Pract. Exp, vol. 24, no. 3, pp. 3

9. Schemat blokowy



[5]

[5] A. Paszkiewicz, C. Ćwikła, M. Bolanowski, M. Ganzha, M. Paprzycki, M. Hodoň: Multifunctional clustering based on the LEACH algorithm for edge-cloud continuum ecosystem.

10. Pseudokod

```
START
   Wczytaj dane
   G = N

   // Pętla dla selekcji cech i przetwarzania danych
   Powtarzaj dopóki |M| >= |F|:
      // Selekcja głów dla każdej cechy w rundzie r
      Wybierz głowy dla każdej cechy: G' zgodnie z przyjętym proporcją

      // Usuwanie wybranych głów z zestawu G
      G = G \ G'

      // Klasteryzacja: przypisanie węzłów członkowskich do poszczególnych
zestawów głów
      Przypisz węzły członkowskie do zestawów głów

      // Proces zbierania, przesyłania i przetwarzania danych
      Zbieraj i przetwarzaj dane

   KONIEC pętli

   Wyjście danych
   Jeśli |M| >= |F|:
      Numer rundy modulo (1/p * |F|) = 0
   KONIEC IF

STOP
```

11. Zastosowania algorytmu LEACH

Monitorowanie środowiska
Zbieranie danych meteorologicznych: W sieciach sensorów wykorzystywanych do monitorowania warunków atmosferycznych, takich jak temperatura, wilgotność, ciśnienie atmosferyczne itp. Algorytm LEACH umożliwia efektywną transmisję tych danych do stacji bazowej, zmniejszając zużycie energii przez węzły w sieci. Monitorowanie jakości powietrza: Sieci sensorów do monitorowania zanieczyszczenia powietrza, które wymagają przesyłania danych w czasie rzeczywistym. LEACH pomaga w redukcji zużycia energii, co przedłuża czas życia sensorów rozmieszczonych w dużych obszarach.

Rolnictwo precyzyjne
Zbieranie danych o glebie i roślinach: W zastosowaniach rolniczych, np. w systemach monitorowania wilgotności gleby, temperatury, pH gleby, algorytm LEACH może pomóc w

minimalizacji zużycia energii przez węzły w rozproszonych sieciach sensorów, co jest szczególnie istotne w dużych obszarach uprawnych. Inteligentne systemy nawadniania: W systemach, które automatycznie dostosowują poziom nawadniania w zależności od wilgotności gleby, LEACH pozwala na długoterminowe monitorowanie parametrów środowiskowych z minimalnym zużyciem energii.

Zarządzanie miastami inteligentnymi (Smart Cities)
Monitorowanie infrastruktury miejskiej: Algorytm LEACH może być wykorzystany w sieciach czujników monitorujących mosty, drogi, systemy kanalizacyjne czy oświetlenie uliczne w inteligentnych miastach. Pomaga to w efektywnym zarządzaniu miastem, zapewniając oszczędność energii, a jednocześnie umożliwiając zbieranie danych na temat stanu infrastruktury. Inteligentne parkowanie: W systemach monitorujących dostępność miejsc parkingowych, LEACH może być używany do przesyłania informacji o wolnych miejscach z czujników do centralnego systemu, minimalizując zużycie energii w sieci czujników.

Monitorowanie zdrowia (WBAN - Wireless Body Area Networks)
Zbieranie danych o stanie zdrowia pacjentów: W sieciach sensorów obszaru ciała (WBAN), które monitorują funkcje życiowe pacjentów, takie jak tętno, temperatura ciała, poziom tlenu we krwi itp., LEACH pozwala na efektywną transmisję danych, zapewniając długoterminowe działanie systemu bez szybkiego wyczerpywania energii węzłów. Monitorowanie sportowców: W zastosowaniach sportowych, gdzie monitorowane są parametry fizjologiczne, LEACH może pomóc w wydajnym przesyłaniu danych o stanie organizmu sportowca, przy minimalnym zużyciu energii.

Przemysł i monitorowanie środowiskowe
Monitorowanie warunków przemysłowych: W przemysłowych systemach monitorowania, gdzie używa się czujników do monitorowania temperatury, ciśnienia, wilgotności itp., LEACH pozwala na długotrwałe działanie sieci, co jest szczególnie przydatne w dużych zakładach przemysłowych, gdzie rozproszona sieć sensorów jest wykorzystywana do wykrywania awarii lub monitorowania parametrów produkcji. Monitorowanie systemów energetycznych: LEACH jest używany w systemach monitorujących sieci energetyczne (np. linie przesyłowe, stacje transformatorowe), gdzie zużycie energii musi być monitorowane w czasie rzeczywistym, ale konieczne jest oszczędzanie energii w sieci czujników.

Zastosowania wojskowe
Bezprzewodowe sieci sensoryczne w terenach wojskowych: LEACH jest wykorzystywany w bezprzewodowych sieciach sensorowych dla celów wojskowych, takich jak monitorowanie granic, detekcja intruzów, a także w systemach śledzenia obiektów. Oszczędność energii w takich warunkach jest kluczowa, zwłaszcza w długoterminowych misjach w trudnym terenie.

Zastosowania w logistyce
Śledzenie towarów i zasobów: Algorytm LEACH może być używany w systemach monitorowania i śledzenia towarów w magazynach lub podczas transportu. Sensory w tych systemach mogą monitorować temperaturę, wilgotność i inne parametry, a LEACH zapewnia efektywne przesyłanie danych, co jest istotne w przypadku dużych przestrzeni magazynowych lub rozproszonych zasobów.

Zastosowania w sieciach sensorów zdrowia i bezpieczeństwa publicznego

Monitorowanie wypadków: LEACH może być wykorzystywany w systemach monitorowania wypadków, takich jak wykrywanie wstrząsów sejsmicznych, pożarów, powodzi itp., gdzie sieci sensorów muszą działać przez dłuższy czas, a jednocześnie minimalizować zużycie energii.

12. Implementacja algorytmu w Arduino

Nasza implementacja wykorzystuje cztery urządzenia Arduino, z których każde pełni rolę czujnika podłączonego do fotorezystora. Węzły zbierają dane oświetlenia i komunikują się z bazą, którą jest komputer bazowy (laptop). Komunikacja między czujnikami oraz między czujnikami a bazą odbywa się za pomocą protokołu UDP, z wykorzystaniem przypisanych adresów IP. Poszczególne czujniki wyznaczono na lidera, które realizują proces klasteryzacji. Liderzy odpowiadają za przypisanie pozostałych czujników do klastrów oraz za zarządzanie przesyłaniem danych do bazy. Pozostałe czujniki pełnią funkcję węzłów zbierających dane, które są przesyłane do lidera lub bazy w zależności od konfiguracji.

```
#include <WiFiNINA.h>
#include <WiFiUdp.h>
#include <limits>
#include <math.h>

#define node_number 4

bool useHEED = true;  // false - LEACH, true - HEED
const char *Nazwa_sieci = /* "……………" */
IPAddress COM6 (192, 168, 33, 20);
IPAddress COM8 (192, 168, 33, 30); const char *COM8_str  = "192.168.33.30";
IPAddress COM9 (192, 168, 33, 40); const char *COM9_str  = "192.168.33.40";
IPAddress COM10(192, 168, 33, 50); const char *COM10_str = "192.168.33.50";
const int Zlacze_UDP = 1234; const int Zlacze_czujnika = A0; WiFiUDP udp;
int Klaster_COM6, Klaster_COM8, Klaster_COM9, Klaster_COM10;

struct node {
   double energy, pos_x, pos_y, distance_to_head;
   int    id, is_head = 0, assigned_to_cluster = false, reports_to = -127;
   bool   is_dead = false;
};
node nodes[node_number], base;

// Energetyczne parametry
const double E_TX = 50e-9, E_RX = 50e-9, E_DA = 5e-9, E_fs = 10e-12, E_mp = 0.0013e-
12;
const int L = 4000;
const double d0 = sqrt(E_fs / E_mp), ENERGY_DEAD = 0.0;

int runda = 0; unsigned long poprzedni_czas = 0; const unsigned long interwal = 5000; // (ms)

// --- Funkcje pomocnicze ---
```

```
double calculate_distance(node a, node b) {
   return sqrt(sq(a.pos_x - b.pos_x) + sq(a.pos_y - b.pos_y));
}
double energyTransmit(double d) {
   return (d < d0) ? (E_TX * L + E_fs * L * d * d) : (E_TX * L + E_mp * L * pow(d, 4));
}
double energyReceive() {
   return (E_RX + E_DA) * L;
}

// --- HEED (z pełnym modelem energii) ---
void heedClustering() {

  /* ---------- 1.  RESET FLAG ---------- */
  for (int i = 0; i < node_number; i++) {
    nodes[i].is_head = 0;
    nodes[i].assigned_to_cluster = false;
    nodes[i].distance_to_head = INFINITY;
  }

  /* ---------- 2.  WYBÓR HEADÓW ---------- */
  bool unassigned_nodes_exist = true;
  while (unassigned_nodes_exist) {
    int head_node   = -1;
    double max_energy = -1;

    for (int i = 0; i < node_number; i++) {
      if (!nodes[i].assigned_to_cluster && !nodes[i].is_dead &&
          nodes[i].energy > max_energy) {
        head_node   = i;
        max_energy  = nodes[i].energy;
      }
    }

    if (head_node == -1) break;                // nie ma już żywych kandydatów

    nodes[head_node].is_head           = 1;
    nodes[head_node].assigned_to_cluster = true;
    nodes[head_node].reports_to        = nodes[head_node].id;

    /*  przydział węzłów do bieżącego head-a */
    for (int i = 0; i < node_number; i++) {
      if (!nodes[i].is_head && !nodes[i].assigned_to_cluster && !nodes[i].is_dead) {
        double dist_to_head  = calculate_distance(nodes[i], nodes[head_node]);
        double dist_to_base  = calculate_distance(nodes[i], base);
        if (dist_to_head < dist_to_base) {
          nodes[i].distance_to_head   = dist_to_head;
          nodes[i].assigned_to_cluster = true;
          nodes[i].reports_to          = nodes[head_node].id;
        }
```

```
    }
  }

  /*  sprawdź, czy zostały nieprzydzielone węzły  */
  unassigned_nodes_exist = false;
  for (int j = 0; j < node_number; j++)
    if (!nodes[j].assigned_to_cluster && !nodes[j].is_dead) {
      unassigned_nodes_exist = true;
      break;
    }
}

/*  ---------- 3.  Ewentualne przełączenie na bliższy head ----------  */
for (int i = 0; i < node_number; i++) {
  if (!nodes[i].is_head && !nodes[i].is_dead) {
    for (int j = 0; j < node_number; j++) {
      if (nodes[j].is_head && !nodes[j].is_dead) {
        double new_dist = calculate_distance(nodes[i], nodes[j]);
        if (new_dist < nodes[i].distance_to_head) {
          nodes[i].distance_to_head = new_dist;
          nodes[i].reports_to       = nodes[j].id;
        }
      }
    }
  }
}

/* ---------- 4. IDENTYCZNE ZUŻYCIE ENERGII JAK W LEACH ---------- */
for (int i = 0; i < node_number; i++) {
  if (nodes[i].is_dead) continue;

  if (nodes[i].is_head) {
    /* head nadaje bezpośrednio do bazy */
    double dBS = calculate_distance(nodes[i], base);
    nodes[i].energy -= energyTransmit(dBS);
  } else {
    /* zwykły węzeł nadaje do swojego head-a */
    nodes[i].energy -= energyTransmit(nodes[i].distance_to_head);

    /* head odbiera pakiet */
    for (int j = 0; j < node_number; j++) {
      if (nodes[j].id == nodes[i].reports_to && !nodes[j].is_dead) {
        nodes[j].energy -= energyReceive();
        break;
      }
    }
  }

  /* sprawdź, czy węzeł „umarł” */
  if (nodes[i].energy <= ENERGY_DEAD) {
```

```
      nodes[i].energy      = 0.0;
      nodes[i].is_dead     = true;
      nodes[i].reports_to = -127;       // jasny sygnał w monitorze
    }
  }

  /* ---------- 5.  ZWROT WYNIKÓW DO PÓL GLOBALNYCH ---------- */
  Klaster_COM6  = nodes[0].reports_to;
  Klaster_COM8  = nodes[1].reports_to;
  Klaster_COM9  = nodes[2].reports_to;
  Klaster_COM10 = nodes[3].reports_to;
}

// --- LEACH ---
void leachClustering(int round) {
  const double p = 0.5;

  auto thresholdT = [&](int round, int nodeID) -> double {
    int r_mod = round % static_cast<int>(1.0 / p);
    double denominator = (1 - p * r_mod);
    if (denominator <= 0.0) denominator = 1e-9;
    return p / denominator;
  };

  // Reset stanu
  for (int i = 0; i < node_number; i++) {
    nodes[i].is_head = 0;
    nodes[i].assigned_to_cluster = false;
    nodes[i].distance_to_head = INFINITY;
    nodes[i].reports_to = -127;
  }

  // Losowy wybór głów
  for (int i = 0; i < node_number; i++) {
    if (nodes[i].is_dead) continue;
    double randVal = static_cast<double>(random(10000)) / 10000.0;
    if (randVal < thresholdT(round, nodes[i].id)) {
      nodes[i].is_head = 1;
      nodes[i].assigned_to_cluster = true;
      nodes[i].reports_to = nodes[i].id;
    }
  }

  // Przypisanie zwykłych węzłów
  for (int i = 0; i < node_number; i++) {
    if (nodes[i].is_dead || nodes[i].is_head) continue;
    double min_dist = INFINITY;
    int closest_head = -1;
    for (int j = 0; j < node_number; j++) {
      if (!nodes[j].is_dead && nodes[j].is_head) {
```

```
      double dist = calculate_distance(nodes[i], nodes[j]);
      if (dist < min_dist) {
        min_dist = dist;
        closest_head = j;
      }
    }
  }

  if (closest_head != -1) {
    nodes[i].distance_to_head = min_dist;
    nodes[i].reports_to = nodes[closest_head].id;
    nodes[i].assigned_to_cluster = true;
  }
}

//Awaryjne przypisanie (jeśli jakiś węzeł żyje, ale nie ma reports_to)
for (int i = 0; i < node_number; i++) {
  if (!nodes[i].is_dead && nodes[i].reports_to == -127) {
    // przypisz siebie jako head
    nodes[i].is_head = 1;
    nodes[i].reports_to = nodes[i].id;
    nodes[i].assigned_to_cluster = true;
    nodes[i].distance_to_head = 0;
  }
}

// Zużycie energii
for (int i = 0; i < node_number; i++) {
  if (nodes[i].is_dead) continue;

  if (nodes[i].is_head) {
    double dBS = calculate_distance(nodes[i], base);
    nodes[i].energy -= energyTransmit(dBS);
  } else {
    if (isfinite(nodes[i].distance_to_head))
      nodes[i].energy -= energyTransmit(nodes[i].distance_to_head);

    for (int j = 0; j < node_number; j++) {
      if (nodes[j].id == nodes[i].reports_to && !nodes[j].is_dead) {
        nodes[j].energy -= energyReceive();
        break;
      }
    }
  }

  if (nodes[i].energy <= ENERGY_DEAD || !isfinite(nodes[i].energy)) {
    nodes[i].energy = 0.0;
    nodes[i].is_dead = true;
    nodes[i].reports_to = -127;
  }
```

```
  }

  // Zwrot do globalnych zmiennych
  Klaster_COM6 = nodes[0].reports_to;
  Klaster_COM8 = nodes[1].reports_to;
  Klaster_COM9 = nodes[2].reports_to;
  Klaster_COM10 = nodes[3].reports_to;
}

void setup() {
  Serial.begin(19600);
  randomSeed(analogRead(Zlacze_czujnika));
  base.id = -1; base.pos_x = 0; base.pos_y = 0;
  nodes[0].id = -6; nodes[1].id = -8; nodes[2].id = -9; nodes[3].id = -10;

  for (int i = 0; i < node_number; i++) {
    // Zakres energii: 0,5 J - 0,55 J
    nodes[i].energy = random(101) / 20000.0;
    nodes[i].pos_x = random(101);
    nodes[i].pos_y = random(101);
    nodes[i].distance_to_head = INFINITY;
  }

  WiFi.config(COM6);
  while (WiFi.begin(Nazwa_sieci, Haslo_sieci) != WL_CONNECTED) delay(1000);
  Serial.println("Połączono z siecią.");
  udp.begin(Zlacze_UDP);
}

void loop() {
  unsigned long aktualny_czas = millis();

  if (aktualny_czas - poprzedni_czas >= interwal) {
    poprzedni_czas = aktualny_czas;

    if (useHEED) heedClustering();
    else leachClustering(runda++);

    Serial.print("Energia: ");
    for (int i = 0; i < node_number; i++) {
        Serial.print(nodes[i].energy, 5);
        Serial.print(" ");
    }
    Serial.println();

    udp.beginPacket(COM8_str, Zlacze_UDP);
    udp.write((uint8_t *)&Klaster_COM8, sizeof(Klaster_COM8));
    udp.endPacket();
    udp.beginPacket(COM9_str, Zlacze_UDP);
    udp.write((uint8_t *)&Klaster_COM9, sizeof(Klaster_COM9));
```

```
    udp.endPacket();
    udp.beginPacket(COM10_str, Zlacze_UDP);
    udp.write((uint8_t *)&Klaster_COM10, sizeof(Klaster_COM10));
    udp.endPacket();

    for (int i = 0; i < node_number; i++)
      if (nodes[i].reports_to == -127) Serial.print("(-) ");
      else Serial.print(String(nodes[i].reports_to) + " ");
    Serial.println();
  }

  int Pomiar = analogRead(Zlacze_czujnika);
  int Odbior;
  int Rozmiar_odbioru = udp.parsePacket();

  switch (Klaster_COM6) {
    case -6:
      Serial.println("COM6: " + String(Pomiar));
      break;
    case -8:
      udp.beginPacket(COM8_str, Zlacze_UDP);
      udp.write((uint8_t *)&Pomiar, sizeof(Pomiar));
      udp.endPacket();
      break;
    case -9:
      udp.beginPacket(COM9_str, Zlacze_UDP);
      udp.write((uint8_t *)&Pomiar, sizeof(Pomiar));
      udp.endPacket();
      break;
    case -10:
      udp.beginPacket(COM10_str, Zlacze_UDP);
      udp.write((uint8_t *)&Pomiar, sizeof(Pomiar));
      udp.endPacket();
      break;
  }

  if (Rozmiar_odbioru == sizeof(Odbior)) {
    udp.read((char *)&Odbior, sizeof(Odbior));
    IPAddress Zdalny_adres = udp.remoteIP();
    if (Zdalny_adres == COM8) Serial.print("COM8: ");
    else if (Zdalny_adres == COM9) Serial.print("COM9: ");
    else if (Zdalny_adres == COM10) Serial.print("COM10: ");
    Serial.println(String(Odbior));
  }

  delay(1000);
}
```

13. Opis algorytmu

Algorytm zawarty w tym kodzie jest implementacją dwóch algorytmów grupowania: **LEACH** (Low-Energy Adaptive Clustering Hierarchy) i **HEED** (Hybrid Energy-Efficient Distributed Clustering). Sieć składa się z kilku węzłów, które zbierają dane (np. z czujników), a algorytmy pomagają w wyznaczaniu tzw. "głów klastrów" oraz przydzielaniu węzłów do odpowiednich klastrów. Algorytmy te pozwalają na oszczędność energii, co jest kluczowe w systemach, w których węzły zasilane są na przykład bateriami.

14.1 Struktura kodu:

1. **Biblioteki**:
   - WiFiNINA.h i WiFiUdp.h - do komunikacji przez Wi-Fi i protokół UDP, który umożliwia bezprzewodową wymianę danych.
   - limits i math.h - do obliczeń matematycznych, takich jak obliczanie odległości czy energii.
2. **Zmienne i struktury**:
   - **node**: Struktura reprezentująca węzeł w sieci. Każdy węzeł zawiera informacje o:
     - energii,
     - położeniu (współrzędne x i y),
     - stanie (czy jest martwy, czy jest głową klastra),
     - odległości do głowy klastra,
     - oraz identyfikatorze
     - id
     - przypisaniu do głowy klastra (czy jest przypisany czy nie).
   - **base**: Reprezentuje bazę (lub stację bazową) w sieci, do której węzły przesyłają dane.
   - **Klaster_COM6, Klaster_COM8, Klaster_COM9, Klaster_COM10**: Zmienne przechowujące informacje o przypisaniu węzłów do poszczególnych klastrów.
3. **Parametry energetyczne**:
   - E_TX, E_RX, E_DA itp. to stałe związane z obliczeniami energii używanej przez węzły przy transmisji i odbiorze danych.
   - L: Długość pakietu danych.
   - d0: Próg odległości, który określa, czy energia transmitowania zależy od d^2 (model przestrzeni wolnej) czy od d^4 (model wielościeżkowy).
4. **Funkcje pomocnicze**:
   - calculate_distance(): Oblicza odległość między dwoma węzłami na podstawie ich współrzędnych.
   - energyTransmit(): Oblicza zużycie energii przez węzeł podczas transmisji w zależności od odległości.
   - energyReceive(): Oblicza zużycie energii podczas odbioru danych.

14.2 Algorytmy:

1. **HEED (Hybrid Energy-Efficient Distributed Clustering)**:
   - **Cel**: Efektywne przydzielanie węzłów do klastrów z uwzględnieniem ich energii.
   - **Działanie**:
     1. Resetuje flagi węzłów (czy są głowami, czy zostały przypisane do klastra).
     2. **Wybór głów klastrów**: Wybiera węzły z największą energią jako głowy klastrów. Węzły te mają największe szanse na zostanie liderami.
     3. **Przydział węzłów do klastrów**: Węzły, które nie są głowami, przydzielane są do najbliższej głowy klastra. Przydział do klastrów odbywa się na podstawie odległości do głowy klastra oraz do bazy.
     4. **Ewentualne przełączenie na bliższego lidera**: Jeśli węzeł jest zbyt daleko od swojej głowy, może przełączyć się na bliższego lidera.
     5. **Zużycie energii**: Węzły zużywają energię na podstawie odległości do głowy klastra lub bazy. Głowy klastrów nadają dane bezpośrednio do bazy, a zwykłe węzły przesyłają dane do swojej głowy klastra.

2. **LEACH (Low-Energy Adaptive Clustering Hierarchy)**:
   - **Cel**: Algorytm dążący do równomiernego rozłożenia energii w sieci przez cykliczną zmianę głów klastrów.
   - **Działanie**:
     1. Resetuje stan węzłów.
     2. **Wybór głów**: Na podstawie określonego prawdopodobieństwa, węzły stają się głowami klastrów. Węzły te nadają dane do bazy.
     3. **Przypisanie węzłów**: Węzły, które nie są głowami, przydzielane są do najbliższej głowy klastra.
     4. **Awaryjne przypisanie**: Jeśli jakiś węzeł nie jest przypisany do żadnej głowy, staje się głową klastra.
     5. **Zużycie energii**: Zgodnie z modelem LEACH, węzły zużywają energię na transmisję do swoich głów i odbiór od nich.

14.3 Działanie głównej pętli programu:

1. **Faza konfiguracji**: Co pewien czas (określony przez interwał) przeprowadzana jest zmiana konfiguracji sieci, gdzie wybrane są głowy klastrów, a węzły są przydzielane do odpowiednich klastrów.
2. **Wysyłanie danych**: Po przypisaniu do klastra, węzły przesyłają dane do swoich głów, a te dalej do bazy. Wartości energii i raporty są wysyłane za pomocą UDP do innych urządzeń w sieci (np. COM8, COM9, COM10).

3. **Monitorowanie energii**: Na bieżąco monitorowana jest energia węzłów, a po wyczerpaniu energii węzeł staje się martwy i nie bierze udziału w dalszej transmisji.

14. Porównanie obu algorytmów

15.1 Podobieństwa:

Cel:

- Celem obu algorytmów jest zmniejszenie zużycia energii w sieci sensorów bezprzewodowych (WSN), co pozwala na dłuższy czas życia całej sieci, poprzez wyznaczenie "głów" klastrów, które będą odpowiedzialne za agregację i transmisję danych.

Zasada działania (ogólny proces):

- W obu algorytmach sieć jest podzielona na klastry, w których wybierane są **główne węzły** (head nodes). Pozostałe węzły są przydzielane do tych głów.
- Węzły **główne** odpowiedzialne są za agregowanie danych i przesyłanie ich do stacji bazowej (bazy danych), a węzły **członkowskie** przesyłają dane do swoich głów.
- **Zarządzanie energią**: W obu algorytmach główną funkcjonalnością jest zarządzanie zużyciem energii w sieci, zarówno na poziomie głów klastrów, jak i węzłów członkowskich.

Cykliczność:

- Oba algorytmy działają w sposób cykliczny, tj. w regularnych odstępach czasu wykonują operacje związane z wyborem głów klastrów i przypisaniem węzłów do odpowiednich klastrów.

15.2 Główna różnica:

W algorytmie LEACH, przy wyborze głów klastrów brana jest pod uwagę wyłącznie energia.

W algorytmie HEED, podobnie jak w LEACH, głowy klastrów są wybierane na podstawie energii, ale proces przypisania jest bardziej złożony, ponieważ bierze pod uwagę zarówno odległość, jak i energię.

15. Wnioski

Ogólne wnioski: Algorytm HEED jest bardziej optymalny ze względu na to, że nie tylko ilość energii jest brana uwagę przy wybieraniu głów klastrów ale również odległość węzłów pomiędzy sobą oraz odległość od bazy. W przypadku algorytmu HEED, straty energii są o wiele mniejsze niż w przypadku algorytmu LEACH.

16. Bibliografia

Paszkiewicz, C. Ćwikła, M. Bolanowski, M. Ganzha, M. Paprzycki, M. Hodoň: Multifunctional clustering based on the LEACH algorithm for edge-cloud continuum ecosystem. Bulletin of the Polish Academy of Sciences Technical Sciences, vol. 72, no. 1, e147919, https://doi.org/10.24425/bpasts.2023.147919

Prasetyo, Y. H., Muladi, & Elmunsyah, H. (2024). Systematic literature review: Development of the LEACH protocol algorithm for efficient energy consumption in WSN. *Przegląd Elektrotechniczny*, 100(7), 148–157. https://doi.org/10.15199/48.2024.07.31

Autorzy nieznani. (n.d.). *Energy Efficient LEACH Protocol for Wireless Sensor Network*. ResearchGate. https://www.researchgate.net/publication/275405800_Energy_Efficient_LEACH_Protocol_for_Wireless_Sensor_Network

Tandel, R. I. (2016). *Leach protocol in wireless sensor network: A survey*. *International Journal of Computer Science and Information Technology*, 7(4), 449-455. https://www.ijcsit.com/docs/Volume%207/vol7issue4/ijcsit2016070449.pdf