Politechnika Warszawska

WYDZIAŁ ELEKTRYCZNY

Instytut Elektrotechniki Teoretycznej
i Systemów Informacyjno-Pomiarowych
Zakład Elektrotechniki Teoretycznej
i Informatyki Stosowanej

Praca dyplomowa magisterska

na kierunku Informatyka
w specjalności Inżynieria oprogramowania

Problemy współbieżności w algorytmach węzła sieci czujnikowej na przykładzie modelu w języku Go.

Jakub Młynarczyk
nr albumu 288226

promotor
dr inż. Łukasz Makowski

WARSZAWA 2019

Problemy współbieżności w algorytmach węzła sieci czujnikowej na przykładzie modelu w języku Go.

Streszczenie

Praca składa się z krótkiego wstępu jasno i wyczerpująco opisującego oraz uzasadniającego cel pracy. Rozdział drugi 'Wykorzystane technologie' opisuje technologie, narzędzia oraz systemu wykorzystane w celu wykonania pracy. Rozdział trzeci 'Bezprzewodowe sieci czujnikowe' przedstawia podstawowe zagadnienia związane z bezprzewodowymi sieci czujników, algorytmami i protokołami stosowanymi w celach uzyskania lepszej efektywności procesu wymiany informacji. Rozdział czwarty 'Architektura systemu' opisuje wymagania oraz implementację autorskiego systemu środowiska symulacyjnego. Rozdział piąty 'Opracowanie wyników eksperymentów' definiuje zakres testów, prezentuje uzyskane wyniki oraz krótko opisuje uzyskane wartości. Ostatni rozdział pracy 'Podsumowanie' to holistyczny opis uzyskanych wyników, zaobserwowanych zależności w bezprzewodowych sieciach czujnikowych oraz możliwościach dalszego rozwoju projektu.

Słowa kluczowe: wsn, sieci czujnikowe, golang, LEACH, PEGASIS

Challenges of concurrency in wireless sensor network, based on a model developed in Go.

Abstract

This thesis presents a novel way of using a novel algorithm to present complex problems of concurrency in wireless sensor networks. In the first chapter briefly presents presents the objectives and goals of the document. The second chapter describes all available tools, technologies and utilities used in the process of writing. The third chapter presents the fundamentals of wireless sensor networks and technologies. The fourth chapter presents requirements and implementation details of custom-made simulation environment written in Go programming language. The fifth chapter defines a set of tests and sub-tests, as well as presents results of those tests, which enable to compare existing wireless sensor network protocols and proves the correctness of end-to-end simulator. Final chapter summarizes all the tests results, discovered dependencies and points some new possibilities of further development of the simulator.

Keywords: wsn, wireless sensor networks, golang, LEACH, PEGASIS

WARSZAWA, 31 marca 2019

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa magisterska pt. Problemy współbieżności w algorytmach węzła sieci czujnikowej na przykładzie modelu w języku Go.:

- została napisana przeze mnie samodzielnie,
- nie narusza niczyich praw autorskich,
- nie zawiera treści uzyskanych w sposób niezgodny z obowiązującymi przepisami.

Oświadczam, że przedłożona do obrony praca dyplomowa nie była wcześniej podstawą postępowania związanego z uzyskaniem dyplomu lub tytułu zawodowego w uczelni wyższej. Jestem świadom, że praca zawiera również rezultaty stanowiące własności intelektualne Politechniki Warszawskiej, które nie mogą być udostępniane innym osobom i instytucjom bez zgody Władz Wydziału Elektrycznego.

Oświadczam ponadto, że niniejsza wersja pracy jest identyczna z załączoną wersją elektroniczną.

Jakub Młynarczyk.....

Spis treści

Podziękowania

Dziękuję bardzo serdecznie wszystkim pracownikom uczelni, rodzinie oraz pracodawcy za poświęcony czas i energię.

Jakub Młynarczyk

Rozdział 1

Wstęp

W ciąg ostatnich lat obserwujemy szybki rozwój technologii informatycznych i teleinformatycznych w zakresie bezprzewodowych sieci czujnikowych. Pierwsze implementacje bezprzewodowych czujników wykorzystano w celach zbrojeniowych. Aktualnie, możliwości oferowane przez bezprzewodowe czujniki znajdują zastosowanie w nieskończonej liczbie aplikacji, zarówno w przemyśle (np. medycynie), jak również w sektorze prywatnym (np. inteligentny dom).

1.1 Cel pracy

Celem pracy są badania problemów współbieżności w algorytmach sieci czujnikowej. Praca wykorzystuje ogólnie znane i udokumentowane algorytmy umożliwiające transmitowanie danych w sieci, w których najważniejszym ograniczeniem jest skończona ilość energii czujnika.

Istotnym elementem pracy jest autorski symulator sieci sensorowej, która stanowi podstawowe narzędzie umożliwiająca implementację oraz testowanie nowych algorytmów. Aplikacja umożliwia tworzenie symulacji porównujących efekty zastosowania różnych algorytmów transmisji danych, bez potrzeby wykorzystania fizycznych urządzeń.

1.2 Tworzenie projektu

Koncepcja projektu informatycznego została opracowana przez autora pracy. Początkowy etap tworzenia oprogramowania polegał na zebraniu wymagań systemu. Niezbędnym było zdefiniowanie danych wejściowych, oraz oczekiwanych danych wyjściowych będących rezultatem działania oprogramowania symulacyjnego. W momencie zatwierdzenia założeń projektu przez

promotora pracy, autor przystąpił do stworzenia prototypu, który miał na celu zweryfikowanie założeń, przetestowanie fundamentów architektury systemu, oraz oszacowanie czasu niezbędnego do ukończenia całości projektu. Pierwszy prototyp uwidocznył istotne braki w funkcjonalności (np. brak wielokrotnego odpalania scenariuszy), które zostały zaimplementowane w finalnej wersji. Projekt zrealizowano w języku programowania Go, wraz z systemem kontroli wersji git, którego zdalna kopia dostępna jest na serwisie GitHub na profilu autora pracy (github.com/keadwen).

Rozdział 2

Wykorzystane technologie

Rozdział ten zawiera opis technologii oraz narzędzi wykorzystanych w pracy dyplomowej. Przedstawiony zostanie język oprogramowania Go, w którym napisanym zostały najważniejsze komponenty pracy dyplomowej. Rozdział zawiera również metody i technologie niezbędne do wygenerowania danych wejściowych, technik serializacji danych oraz ich reprezentacji przy wykorzystaniu Protocol Buffers, GNUPlot, itd.

2.1 Język programowania Go

Go (Golang) to język programistyczny stworzony jako wolne oprogramowanie (open-source) na potrzeby firmy Google, Inc. Głównymi architektami języka są Robert Griesemer, Rob Pike i Ken Thompson. Golang umożliwia tworzenie komercyjnego oprogramowania i jest wspierany na wielu systemach operacyjnych (Linux, Windows, Mac OS X).

Język Go należy do kategorii języków kompilowanych, z statycznym definiowaniem typu zmiennych. Poniżej przedstawiony został fragment kodu źródłowego napisane w języku Go.

```
package main

import (
  "fmt"

  "github.com/golang/example/stringutil"
)

type Label struct {
```

```
  Text string
  X, Y int
}

func (l *Label) ReverseText() {
  l.Text = stringutil.Reverse(l.Text)
}

func main() {
  var x = 100
  fmt.Printf("<%T>: %v", x, x)
  // Expected output: <int>: 100
  hello := Label{
    Text: "Hello",
    X:    100,
    Y:    200,
  }

  fmt.Println(hello.Text)
  // Expected output: hello
  hello.ReverseText()
  fmt.Println(hello.Text)
  // Expected output: olleh
}
```

W powyższym przykładzie przedstawione zostało kilka innowacyjnych funkcjonalności języka Go. Składnia języka oraz funkcjonalności zbliżone są do C/C++ czy Python, jednakże występuje kilka cech unikatowych dla Go.

Porównanie Go z Python

- Go jest językiem wspomagającym tworzenie aplikacji wielowątkowych.
- Go jest językiem statycznym, co pozwala wyeliminować błędy typu runtime wynikające z typu zmiennej.
- Go jest językiem samodokumentującym. Określony odgórnie format komentarzy umożliwia automatyczne tworzenie przejrzystej dokumentacji.
- Go jest językiem kompilowanym, co pozwala na szybsze uruchamianie i egzekucję oprogramowania.
- Go wykorzystuje mniej pamięci. Przykład: W Go zmienna typu int32 wymaga 4 bajty pamięci, w Python 24 bajty.

- Python umożliwia runtime reflection.
- Python posiada większy bazę publicznych bibliotek.

Porównanie Go z C++

- Go posiada system zarządzania pamięcią (garbage collector).
- Go jest językiem wspomagającym tworzenie aplikacji wielowątkowych.
- Go jest językiem samodokumentującym. Nie wymaga tworzenia plików typu header.
- Go nie jest językiem obiektowym. Zdolność dziedziczenia (inheritance) została zastąpiona osadzaniem (embedding).
- Go posiada możliwość użycia (zaimportowania) dowolnej biblioteki C, C++.
- C++ posiada osiągnąć szybszą egzekucję oprogramowania.
- C++ posiada tworzenie kodu niezależnie od typu zmiennej (generics).
- C++ nie posiada systemu zarządzania pamięcią (garbage collector), co umożliwia większą kontrolę nad zasobami pamięci (np. w mikro kontrolerach).

Unikatowe cechy Go

- Produktem kompilacji jest plik egzekucyjny posiadający wszystkie niezbędne zależności.
- Zarządzanie pakietami pozwala na importowanie rozwiązań bezpośrednio z GitHub (lub innych serwisów zewnętrznych).
- Dynamiczna alokacja typu zmiennej statycznej.
- Natywne wspieranie tworzenie oprogramowania wykorzystującego wielowątkowość i współbieżność procesów.
- Natywna metoda testowania funkcjonalności i bibliotek.
- Pełny zestaw wbudowanych narzędzi pozwalających na testowanie wydajności kodu (benchmarking), formatowania składni kodu zgodnie ze standardem (gofmt), oraz wiele innych funkcjonalności.

2.2 Serializacja danych Protocol Buffers

Protocol Buffers (proto, protobuf) to mechanizm serializacji danych stworzony na potrzeby firmy Google, Inc. Protocol Buffers to mechanizm współpracującym niezależnie od języka oprogramowania aplikacji czy platformy na którym uruchamiana jest aplikacja. Technologia ta definiuje strukturę danych (proto schema) za pomocą dedykowanego języka, składającego się z prostych zmiennych (np.: int64, string) oraz złożonych komunikatów (message). Poniżej przedstawiona została przykładowa struktura.

```
// example.proto
syntax = "proto3";

// Citizen represents a single citizen of Poland.
message Citizen {
  // The name of a citizen.
  string name = 1;
  // The surname of a citizen.
  string surname = 2;
  // (required) Unique Polish national identification number.
  PESEL pesel = 3;
}

// PESEL represents Polish Universal Electronic System for
// Registration of the Population.
message PESEL {
  // (required) Unique Polish national identification number.
  uint64 number = 1;
  bool active = 2;
}
```

Struktura ta przechowywana jest w plikach o rozszerzeniu ‘.proto‘, które są następnie kompilowane do dowolnego z wspieranych języków oprogramowania (w przypadku Protocol Buffers w wersji 3, wspierana jest generacja kodu w Java, C++, Python, Java Lite, Ruby, JavaScript, Objective-C, C# oraz PHP). Następnie, dane zserializowane zostają zapisane w formacie binarnym (wire format), który umożliwia na uzyskanie wyższego poziomu kompresji danych oraz transmisję danych bez potrzeby wykonania dalszego kodowania. Wynikiem kompilacji plików ‘.proto‘ jest zestaw bibliotek zawierający wygenerowany kod źródłowy, wraz z gotowymi strukturami, funkcjami i metodami niezbędnymi do operowania danymi w sposób natywny dla wybranego

języka programowania.

2.3 Narzędzia dodatkowe

Sekcja ta przedstawia zestaw narzędzi których funkcjonalności umożliwiły stworzenie oraz ułatwiły zarządzanie oprogramowaniem stworzonym w celach pracy dyplomowej.

2.3.1 System kontroli wersji git

Git to rozproszony system kontroli wersji stworzony jako wolne oprogramowanie (open source). Głównymi architektem narzędzia jest Linus Torvalds. Git to oprogramowanie powszechnie stosowanym w przypadku zarządzania oprogramowaniem. Narzędzie to umożliwia tworzenie pobocznych gałęzi (branch) niezależnych od głównej gałęzi. Funkcjonalność ta pozwala na niezależne wprowadzanie zmian w kodzie na określonej wersji kontrolnej, które mogą następnie zostać wprowadzone ponownie do gałęzi głównej (merge). Architektura rozproszona git (w przeciwieństwie do scentralizowanych systemów kontroli wersji) umożliwia programistom na posiadanie lokalnej kopii repozytorium, której zmiany mogą zostać następnie wprowadzone do gałęzi głównej.

2.3.2 GNUPlot

GNUPlot to narzędzie do generowania wykresów funkcji w oparciu o dane wejściowe. Program dostępny jest niemal na każdym systemie operacyjnym. Przy pomocy GNUPlot generować można dwu- oraz trójwymiarowe wykresy, które zapisane mogą zostać w różnych formatach t.j. PNG, SVG czy JPEG.

2.3.3 Docker

Docker to narzędzie stworzone jako wolne oprogramowanie (open source) napisane w języku Go przez firmę Docker, Inc. Narzędzie to pozwala na tworzenie kontenerów, które izolują aplikację na poziomie systemu operacyjnego. W przeciwieństwie do maszyn wirtualnych, kontener nie wymaga wirtualizowania systemu operacyjnego dla każdego z kontenerów. Wszystkie równolegle działające kontenery aplikacji działające na pojedynczym urządzeniu współdzielą parametry fizyczne maszyny oraz jądro systemu operacyjnego (np. Linux). Izolacja kontenerów widoczna jest na poziomie zależności (depen-

dency) do określonych wersji bibliotek (libraries), narzędzi (binaries), plików konfiguracyjnych czy parametrów.

Rozdział 3

Bezprzewodowe sieci czujnikowe

W tym rozdziale przedstawiona zostanie najważniejsza część teoretyczna bezprzewodowych sieci sensorowych, która jest niezbędna w celu zrozumienia problemu rozwiązywanego w pracy dyplomowej.

3.1 Wstęp do bezprzewodowych sieci sensorowych

Bezprzewodowe sieci sensorowe (wireless sensor network) nazywa się również bezprzewodowymi sieciami czujników. Sieć czujnikowa składa się z urządzeń, których funkcją jest realizowanie określonego zadania. Pierwsze aplikacje i zastosowania bezprzewodowych sieci czujników zostały wdrożone na potrzeby wojskowe, jednakże przeciągu ostatnich lat, technologie te znalazły wiele nowych zastosowań w przemyśle (np. pomiary meteorologiczne) i aplikacjach codziennych (np. systemy domów inteligentnych).

Rozwój technologii, malejący koszt elektroniki oraz dostępność produktów bezprzewodowej sieci czujnikowej, umożliwia tworzenie dedykowanych aplikacji. Na rynku dostępnych jest wiele urządzeń oraz rozwiązań, a proces wyboru uzależniony jest przede wszystkim od wymogów projektu oraz budżetu. Dla uproszczenia przyjąć można, że pojedynczy czujnik powinien składać się z procesora zdolnego wykonywać określone zadanie, pamięci zdolnej do przechowywania informacji oraz anteny, która umożliwia nadawanie i odbieranie informacji.

Proces wymiany informacji między urządzeniami zależy od protokołów i implementacji. Niezależnie jednak od protokołu i implementacji, cel pozostaje niezmienny. Informacje posiadane przez węzeł w sieci (np. pomiar temperatury otoczenia) muszą zostać przekazane z węzła pomiarowego do węzła głównego. Węzeł główny, zwany również ‘sink‘ jest odpowiedzialnym

za agregację wszystkich informacji oraz ich dalsze przetwarzanie. W dalszej części pracy, przedstawione zostaną dwa protokoły trasowania (routingu), których zadaniem jest zoptymalizowanie energetyczne procesu komunikacji, podwyższenie niezawodności systemu oraz umożliwienie zautomatyzowanej organizacji topologii sieci. By móc zrozumieć istotę i korzyści wykorzystania protokołów trasowania, niezbędnym jest przedstawienie najprostszego schematu komunikacji, komunikacji bezpośredniej.

3.2 Model transmisji

W pracy wykorzystany został model transmisji przedstawiony w pracy [2]. Model jest powszechnie wykorzystywany w planowaniu oraz symulowaniu aplikacji bezprzewodowych sieci czujników. Model przedstawia zależności transmisji danych w wolnej przestrzeni pomiędzy dwoma węzłami (nadawcą i odbiorcą) uwzględniając wielotorowość sygnału. Poniższy wzór pozwala na wyznaczenie całkowitego kosztu transmisji k-bitów do węzła oddalonego o d-metrów:

$$E_{TX}(k,d) = E_{TX-elec}(k) + E_{TX-amp}(k,d) = \begin{cases} k \times E_{elec} + k \times \varepsilon_{fs} \times d^2 & d < d_{d_0} \\ k \times E_{elec} + k \times \varepsilon_{fs} \times d^4 & d \geqslant d_{d_0} \end{cases}$$

gdzie E_{elec} jest energią wykorzystywaną przez nadajnik (np. praca mikroprocesora), d_0 wyznaczamy przez $\sqrt{\varepsilon_{fs}/\varepsilon_{mp}}$. W zależności od odległości na której odbywa się komunikacja, ε_{fs} reprezentuje transmisję w wolnej przestrzeni, natomiast ε_{mp} wielotorowość transmisji.

Poniższy wzór pozwala na wyznaczenie całkowitego kosztu odbioru k-bitów przez węzeł:

$$E_{RX}(k,d) = E_{RX}(k) = k \times E_{elec}$$

3.3 Protokoły

Warstwa sieciowa w bezprzewodowych sieciach czujników jest elementem krytycznym działania całości systemu. Warstwa ta w przeciwieństwie do warstwy fizycznej, odpowiedzialna jest za logiczne skonfigurowanie węzłów i zaimplementowanie algorytmu trasowania danych pomiędzy węzłami.

Istnieje wiele protokołów WSN, których dobór zależy od przede wszystkim od naszej aplikacji oraz parametrów sieci które wymagają optymalizacji.

Poniższej przedstawione zostały cztery aspekty, których dokładne określenie pozwoli na dopasowanie najbardziej optymalnego algorytmu:

- koszty energetyczne - w sieci w której ilość energii węzła jest ograniczona, zalecanym jest stosowanie algorytmów pozwalających na efektywną wymianę danych.
- adresowania i rozpoznawania węzłów - w sieci w której występuje duża ilość węzłów, należy określić metodę adresacji. Metoda statyczna (np. wykorzystanie pliku konfiguracyjnego) lub dynamiczna (np. wygenerowanie unikatowego numeru identyfikującego).
- skalowalności sieci - w sieci w której występuje duża ilość węzłów wymiana informacji może być utrudniona zarówno pod kątem fizycznym (np. brak dostępnej przepustowości w paśmie bezprzewodowym), jak i logicznym (np. długi czas detekcji węzłów i budowania topologii sieci).
- niezawodności wymiany danych - w sieci w której dane generowane przez poszczególne węzły nie mogą zostać utracone, zalecane jest wykorzystanie algorytmu umożliwiającego detekcję brakujących fragmentów danych i retransmisję.

W następnej części pracy przedstawione zostaną 3 protokoły trasowania danych, które zostały zaimplementowane w symulatorze. Wybór ukierunkowany był na ograniczenie kosztów energetycznych w sieci, które umożliwia na wydłużenie czasu życia każdego z węzłów.

3.3.1 Komunikacja bezpośrednia

Komunikacja bezpośrednia jest rozwiązaniem najprostszym z perspektywy implementacji, jednakże nieefektywnym z poziomu energetycznego. Węzły znajdujące się sieci nie są zaangażowane w podejmowanie decyzji dotyczących optymalizacji kosztów transmisji. Adres węzła głównego ('sink') może być zaprogramowany na poziomie pliku konfiguracyjnego. Dane wysyłane przez węzeł nadawane są bezpośrednio do węzła głównego. Model środowiska symulacyjnego definiuje koszt transmisji komunikacji. Wartość ta zależna jest od odległości między dwoma węzłami. Czas życia węzłów (posiadających identyczną ilość energii początkowej oraz wielkość przesyłanych informacji) wydłuża się, wraz z malejącym dystansem do węzła głównego.

Poniższa ilustracja przedstawia trzy węzły w sieci, które komunikują się bezpośrednio z węzłem głównym:

3.3.2 LEACH

LEACH (Low-Energy Adaptive Clustering Hierarchy) jest algorytmem wykorzystywanych w protokołach routingu bezprzewodowej sieci czujnikowej. W przeciwieństwie do komunikacji bezpośredniej, LEACH jest protokołem w którym występuje hierarchia.

Zestaw operacji w protokole LEACH nazywa się rundami (round). Każda z rund podzielona jest na dwie fazy. W pierwszej fazie, fazie konfiguracyjnej (setup), węzły dokonują podziału sieci na niezależne klastry (cluster). W drugiej fazie, fazie komunikacyjnej (steady state), węzły uczestniczące w sieci dokonują wymiany informacji za pomocą agregacji danych w klastrze, które następnie przesłane są do stacji bazowej.

Faza konfiguracji (setup) zbudowana jest z trzech etapów. W pierwszym etapie węzły dokonują wyboru roli w rundzie. W LEACH występują dwa rodzaje węzłów:

- węzły typu cluster head (CH)
- węzły standardowe (non-CH)

Każdy z węzłów generuje losową wartość ‘n‘ z zakresu [0, 1]. Wartość ta podstawiona w poniższe równanie:

$$T(n) = \begin{cases} \frac{P}{1-P[r*mod(1/P)]} & \text{jeżeli } n \in G \\ 0 & \text{w innym przypadku} \end{cases}$$

gdzie P oznacza wartość z zakresu (0, 1] definiująca prawdopodobieństwa wyznaczenia węzła typu cluster head. G jest zbiorem węzłów które nie pełniły roli cluster head w ostatnich 1/P rundach.

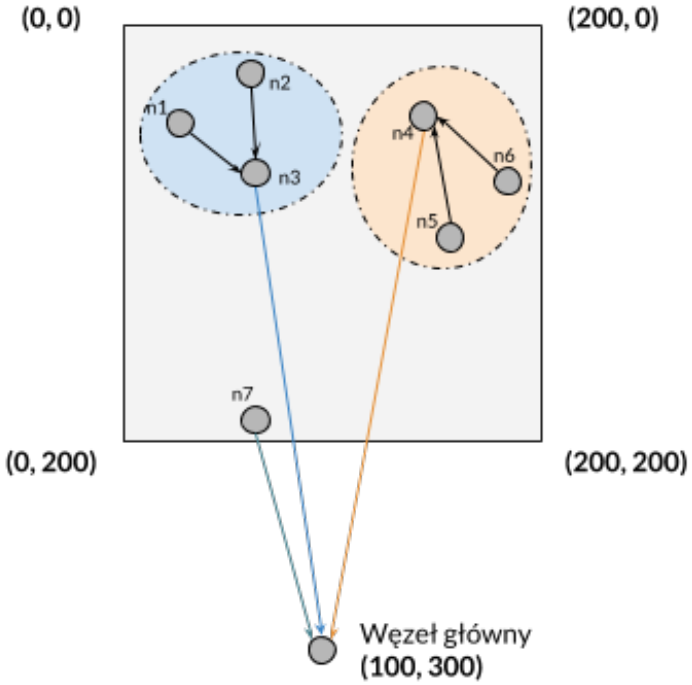
W momencie ustalenia roli przez węzły, każdy z węzłów informuje pozostałe węzły uczestniczące w sieci o swojej roli w rundzie.

Kolejnym etapem fazy konfiguracji jest wyznaczenie klastra do którego zostaną przyłączone węzły standardowe. Każdy z węzłów standardowych wybiera jeden węzeł typu cluster head. W przypadku otrzymania informacji od kilku węzłów typu cluster head, węzeł standardowy wybiera węzeł CH znajdujący się najbliższej (węzeł którego sygnał jest najmocniejszy).

Ostatnim etapem jest ukończenie formowania klasterów w którym znajduje się dokładnie jeden węzeł typu cluster head. Wystąpić może sytuacja w której węzeł typu cluster head nie posiada przynależących węzłów standardowych. Pozytywnie zakończona faza formowania klastrów tworzy stabilną sieci wymiany informacji.

W tym momencie rozpoczyna się faza komunikacji (‘steady state‘) w której węzły standardowe dokonują transmisji danych do węzła typu cluster head. Węzły CH dokonują agregacji zebranych informacji oraz bezpośrednie przekazanie ich do węzła głównego (‘sink‘). Stworzenie środowiska w którym węzły dokonują pośredniczenia informacji, pozwala na ograniczenie kosztów energetycznych transmisji danych przez standardowe węzły (dystans do węzła typu cluster head jest mniejszy niż do ‘sink‘). Należy jednak pamiętać, że koszt energetyczny transmisji węzłów typu cluster head rośnie, gdyż stają się one odpowiedzialne za odbieranie informacji, przetworzenie ich, a następnie wysłanie do całości do ‘sink‘. Wielkość informacji (mierzona w bajtach), przetransmitowana z węzłów typu cluster head jest zależna od ilości standardowych węzłów przynależących do CH oraz metody agregacji informacji.

Poniższa ilustracja przedstawia zestaw węzłów w sieci, które wykorzystują protokół LEACH:



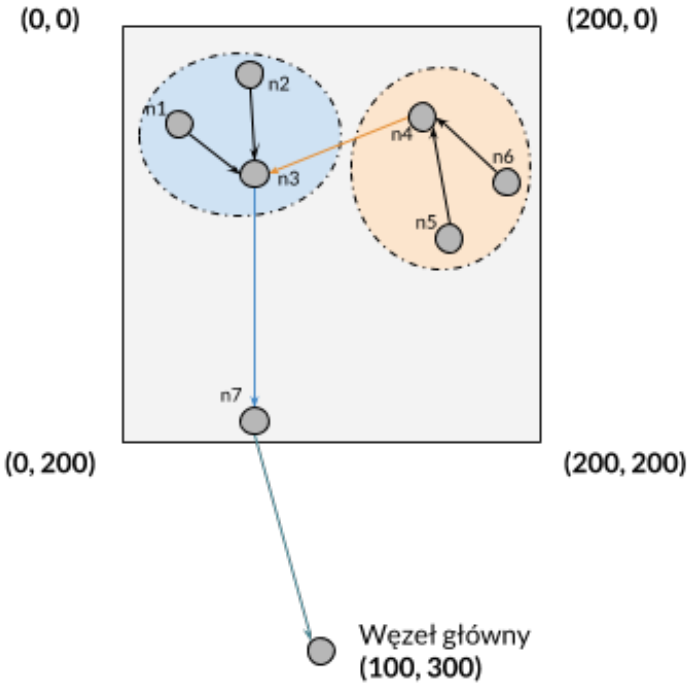
3.3.3 PEGASIS

PEGASIS (Power-Efficient GAathering in Sensor Information Systems) jest algorytmem wykorzystywanym w protokołach routingu bezprzewodowej sieci czujnikowej. W przeciwieństwie do komunikacji bezpośredniej, PEGASIS podobnie jak LEACH jest protokołem w którym występuje hierarchia.

PEGASIS jest protokołem dokonującym usprawnień w protokole LEACH. W przypadku sieci czujnikowej, urządzenia pełniące rolę węzłów posiadają ograniczone ilości energii, wynikające np. z zasilania bateryjnego. Przebudowana metoda przekazywania danych zebranych przez węzły typu cluster head, pozwala na ukończenie rundy przy wykorzystaniu mniejszej ilości energii niż LEACH.

Pierwsza faza, faza komunikacji, w której ustalane są role węzłów jest identyczna. Przebudowana została metoda transmisji agregacji informacji od węzłów typu cluster head to węzła głównego (‘sink‘). Węzły typu cluster head formują łańcuch, w którym najdalej oddalony węzeł typu cluster head, przekazuje zebrane i dane agregowane do innego węzła typu cluster head znajdującego się w najbliższym sąsiedztwie. Algorytm trasowania najmniejszym kosztem nazywana się typu ‘greedy‘.

Efektem niekorzystnym w protokole PEGASIS jest wydłużenie czasu każdej z rund, wynikające z sekwencyjnej wymiany danych w łańcuchu. Generowane w ten sposób są dodatkowe koszty energetyczne związane z utrzymaniem węzłów w stanie pracy, kosztem odbioru i przetworzenia informacji. Należy jednak podkreślić, że zysk energetyczny wynikający z transmisji danych na krótsze dystanse kompensuje poniesione straty, tym samym pozwalając na wydłużenie czasu pracy każdego z węzłów.



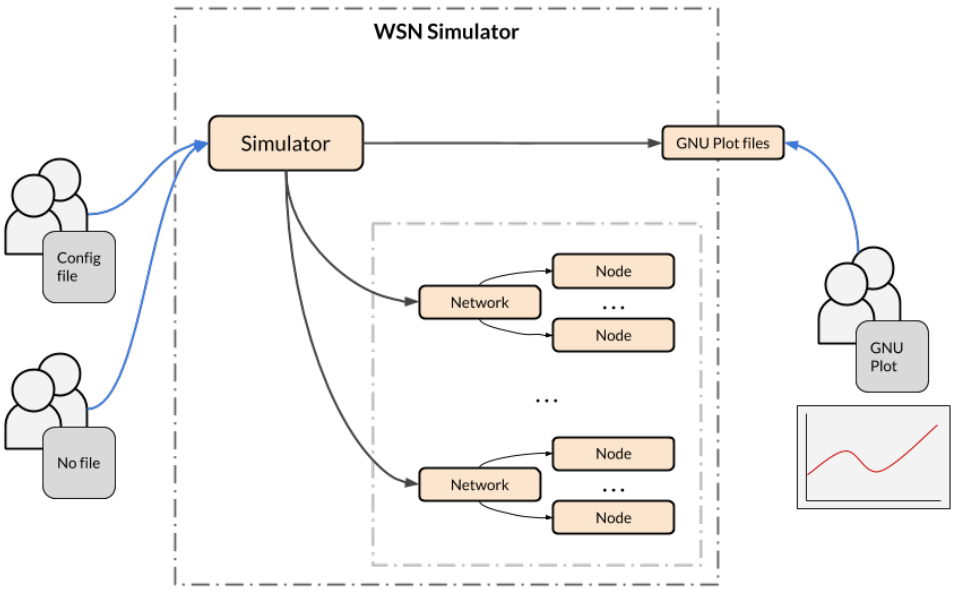
Rozdział 4

Architektura systemu

System informatyczny stworzony na potrzeby pracy dyplomowej ma za zadanie dostarczenie wyników oraz wykresów niezbędnych do zbadania poniższych zależności sprawności energetycznej węzłów dla wybranych algorytmów w sieci WSN (komunikacji bezpośredniej, LEACH, PEGASIS).

4.1 Komponenty systemu

Poniższy diagram przedstawia dekompozycję modułów, całkowitą architekturę systemu oraz kierunki interakcji i przepływu informacji.



Użytkownik (znajdujący się po lewej stronie) posiada możliwość uruchomienia oprogramowania na dwa sposoby:

- symulacja wstępnie konfigurowana za pomocą pliku konfiguracyjnego
- symulacja bez pliku konfiguracyjnego

Opcja symulacji bez pliku konfiguracyjnego powoduje wygenerowanie pliku konfiguracyjnego w którym znajduje się 200 węzłów, rozproszonych w sposób pseudolosowy na przestrzeni 200 x 200.

Symulator tworzy wewnętrzną strukturę sieci i węzłów, które poddawane są symulacji. Wynik końcowy poszczególnych sieci zwracany jest do symulatora, który generuje pliki w formacie kompatybilnym z narzędziem GNU Plot.

4.1.1 Plik konfiguracyjny

Plik konfiguracyjny (‘config file‘) posiada dane wejściowe pozwalające na zbudowanie środowiska testowego. Informacje zawarte w pliku konfiguracyjnym można podzielić na dwie sekcje:

- konfiguracja symulatora i sieci
- konfiguracja węzła

Konfiguracja symulatora i sieci

Konfiguracja symulatora i sieci składa się z pięciu zmiennych w komunikacie ‘Config‘. Każda z tych wartości może być modyfikowana bez potrzeby ponownej kompilacji oprogramowania symulacyjnego. Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych parametrów konfiguracyjnych:

- protokoł (‘protocol‘) - zmienna ta zdefiniowana za pomocą komunikatu typu ‘enum E_Protocol‘ pozwala symulatorowi wybrać odpowiedni protokół sterujący symulacją. Dokładny opis działania protokołów zostanie przedstawiony w dalszej części pracy.
- wartości maksymalnej rund pomiarowych (‘max_rounds‘) - zmienna ta pozwala określić wartość rund pomiarowych aktywnych węzłów w pojedynczej symulacji, po której całkowity przebieg symulacji zostanie zatrzymany. Wyznaczenie tej wartości umożliwia użytkownikowi określenie dowolnej granicy, bez potrzeby oczekiwania na zakończenie symulacji (wykorzystanie całkowitej energii dostępnej przez węzły pomiarowe).

- proporcji węzłów typu klaster do wszystkich węzłów (‘p_cluster_heads‘) - zmienna ta pozwala określić stosunek ilości węzłów pomiarowych odpowiedzialnych za pośredniczenie w przesyłaniu danych pomiarowych (klastrów) do ilości wszystkich węzłów w sieci. Parametr ten wykorzystywany jest zależnie od protokołu.
- długość wiadomości (‘msg_length‘) - zmienna ta określa całkowity rozmiar wiadomości generowanych przez pojedynczy węzeł pomiary. Długość wyrażona jest w bajtach i jest sumą dwóch elementów, danych pomiarowych oraz dodatkowych danych generowanych w procesie enkapsulacji (np. adresowanie, preambuły, itd.)
- konfiguracja węzłów (‘nodes‘) - zmienna ta zdefiniowana za pomocą listy komunikatów typu Node. Symulacja musi składać się przynajmniej z dwóch węzłów. Kolejność listy ma znaczenie, gdyż pierwszy węzeł pełni rolę głównego odbiornika danych (‘sink‘). Dokładniejszy opis parametrów poszczególnych węzłów przedstawiony został w kolejnym podrozdziale ‘Konfiguracja węzła‘.

Poniżej przedstawiony został fragment pliku konfiguracyjnego dla konfiguracji symulatora i sieci:

```
enum E_Protocol {
  UNSET = 0;
  DIRECT = 1;
  LEACH = 2;
  APTEEN = 3;
  PEGASIS = 4;
}

message Config {
  // Simulation protocol.
  E_Protocol protocol = 1;
  // Number of maximum rounds in simulation.
  int64 max_rounds = 2;
  // Percentage of cluster heads among all nodes [0, 1].
  double p_cluster_heads = 3;
  // Size of data sent by individual node (in Bytes).
  int64 msg_length = 4;
  // Nodes points to configuration for each node.
  repeated Node nodes = 5;
}
```

Konfiguracja węzła

Konfiguracja węzła składa się z pięciu zmiennych w komunikacje ‘Node‘. Każda z tych wartości może być modyfikowana bez potrzeby ponownej kompilacji oprogramowania symulacyjnego. Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych parametrów konfiguracyjnych:

- indeks (‘id‘) - zmienna ta określa unikatowy identyfikator węzła. Parametr ten umożliwia identyfikację węzła podczas symulacji.
- energia początkowa (‘initial_energy‘) - zmienna ta określa ilość energii (mierzonej w [J]), którą posiada węzeł w momencie rozpoczęcia symulacji. W przypadku zdefiniowania zerowej energii początkowej, węzeł nie będzie brał udziału w komunikacji ze względu na brak zasobów energetycznych na przeprowadzenia jakiejkolwiek operacji.
- pozycja (‘location‘) - zmienna ta zdefiniowana za pomocą komunikatu typu ‘Location‘ pozwala symulatorowi na umieszczenie węzła w dwuwymiarowej przestrzeni. Pozycja węzła wykorzystywana jest do określania odległości między węzłami i aplikowania kosztów energetycznych operacji (np. transmisji danych).
- koszt energetyczny (‘energy_cost‘) - zmienna ta zdefiniowana za pomocą komunikatu typu ‘EnergyCost‘ pozwala na wprowadzenie dodatkowych kosztów energetycznych operacji dla poszczególnych węzłów. Podstawowy model transmisji posiada zdefiniowane koszty energetyczne operacji. W przypadku zdefiniowana zmiennej ‘EnergyCost‘ dla węzła, koszt operacji (np. transmisji, odbioru, pomiaru i przetwarzania danych) ulega zmianie.
- opóźnienia czasowe (‘time_delay‘) - zmienna ta zdefiniowana za pomocą komunikatu typu ‘TimeDelay‘ pozwala symulatorowi na wprowadzenie dodatkowych parametrów czasowych dla operacji wykonywanych przez węzeł. Iloczyn zmiennej (np. czasu przetwarzania danych węzła [ns]) i kosztu energetycznego działania węzła [J/s] reprezentuje dodatkowe parametry symulacji, które umożliwiają tworzenie zaawansowanych i precyzyjnych scenariuszy.

Poniżej przedstawiony został fragment pliku konfiguracyjnego dla konfiguracji węzła:

```
// Node defines a configuration for a single node.
message Node {
  // Unique ID for a node.
```

```
  int64 id = 1;
  // Initial value of energy (in Joules).
  double initial_energy = 2;

  // Location of a node in 2D space.
  Location location = 3;
  // Energy consumption of node operations.
  EnergyCost energy_cost = 4;
  // Time delays introduced by node operations
  TimeDelay time_delay = 5;
}
```

Poniżej przedstawiony został fragment pliku konfiguracyjnego dla konfiguracji lokalizacji węzła:

```
// Location defines a X, Y coordinates of a node.
message Location {
  double X = 1;
  double Y = 2;
}
```

Poniżej przedstawiony został fragment pliku konfiguracyjnego dla konfiguracji kosztów energetycznych pracy węzła:

```
// EnergyCost defines energy consumption for common node operations.
message EnergyCost {
  // Energy required to transmit one byte (in Joules).
  double transmit = 1;
  // Energy required to receive one byte (in Joules).
  double receive = 2;
  // Energy required to listen the channel for a second (in Joules).
  double listen = 3;
  // Energy required to process sensor data (in Joules).
  double sensor_data_process = 4;
  // Energy required to wake up MCU (in Joules).
  double wake_up_mcu = 5;
}
```

Poniżej przedstawiony został fragment pliku konfiguracyjnego dla konfiguracji opóźnień czasowych pracy węzła:

```
// TimeDelay defines time delays for common node operations.
message TimeDelay {
  // Time required to process sensor data (in nanoseconds).
  int64 sensor_data_process = 1;
  // Time required to wake up MCU (in nanoseconds).
  int64 wake_up_mcu = 2;
}
```

4.1.2 Moduł główny (Core)

Moduł główny posiada najważniejsze cechy i funkcjonalności umożliwiające modelowanie systemu symulatora WSN. Wszystkie zaimplementowane struktury (‘structs‘) znajdują się w jednej bibliotece (‘package‘) o nazwie ‘core‘.

Węzeł (Node)

Węzeł (‘Node‘) reprezentuje model węzła, będącego elementem podstawowym w sieci. Struktura ta przechowuje dane konfiguracyjne węzła, poziom energii czy dane historyczne, pozwalające na monitorowanie pracy oraz tworzenie grafów.

Poniżej przedstawiona została struktura węzła oraz definicje funkcji przynależące do struktury.

```
type Node struct {
  Conf   config.Node
  Ready  bool
  nextHop *Node  // As a default set to Base Station.
  Energy float64 // Energy level of a node.

  transmitQueue int64
  receiveQueue int64
  // Statistics and aggregation variables.
  dataSent    int64
  dataReceived int64
}

func (n *Node) Transmit(msg int64, dst *Node) error
func (n *Node) Receive(msg int64, src *Node) error
func (n *Node) Info() string
```

```
func (n *Node) distance(dst *Node) float64
func (n *Node) consume(e float64) error
```

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych zmiennych struktury węzła (Node):

- konfiguracja (‘Conf‘) - zmienna publiczna przechowuje konfigurację węzła w formacie protocol buffer (szczegółowe informacje dostępne w ‘Konfiguracja węzła‘).
- gotowość (‘Ready‘) - zmienna publiczna przechowuje stan gotowości węzła. Wartość ‘true‘ oznacza, że węzeł jest gotowy do nadawania i odbierania informacji.
- następny skok (‘nextHop‘) - zmienna prywatna przechowuje adres do zmiennej węzła, będącego odbiorcą informacji nadawanych przez węzeł. Wartością domyślna w momencie rozpoczęcia symulacji jest adres głównego odbiornika danych (‘sink‘).
- energia (‘Energy‘) - zmienna publiczna przechowuje aktualny stan energetyczny węzła. W przypadku wyczerpania energii, zmienna Ready zostaje ustawiona na ‘false‘.
- kolejka nadawania (‘transmitQueue‘) - zmienna prywatna przechowuje informację o ilości bajtów gotowych do przekazania do następnego węzła na końcu rundy.
- kolejka odbioru (‘receiveQueue‘) - zmienna prywatna przechowuje informację o ilości bajtów odebranych przez węzeł na początku rundy.
- dane nadane (‘dataSent‘) - zmienna prywatna przechowuje sumę bajtów nadanych przez węzeł.
- dane odebrane (‘dataReceived‘) - zmienna prywatna przechowuje sumę bajtów odebranych przez węzeł.

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych funkcji węzła (Node):

- Transmit - funkcja pozwala na transmisję danych do węzła.
- Receive - funkcja pozwala na odbieranie danych przez węzeł.
- Info - funkcja generuje ciąg znaków w podstawowymi informacjami na temat węzła.
- distance - funkcja wyznacza wartość odległości pomiędzy dwoma węzłami.
- consume - funkcja obciąża energetycznie węzeł.

Sieć (Network)

Sieć (‘Network‘) reprezentuje model środowiska w którym odbywa się symulacja. Struktura ta przechowuje kontroluje przepływ informacji pomiędzy węzłami, zbiera i eksportuje informacje z poszczególnych rund.

Poniżej przedstawiona została struktura sieci oraz definicje funkcji przynależące do struktury.

```
type Network struct {
  Protocol   Protocol
  BaseStation *Node
  Nodes      sync.Map

  Round    int64
  MaxRounds int64
  MsgLength int64

  GNUPlotNodes     []string
  GNUPlotTotalEnergy []string

  PlotTotalEnergy *plot.Plot // An amount of total energy in the
     network per Round.
  PlotNodes       *plot.Plot // A number of alive nodes in the
     network per Round.
  NodesAlivePoints plotter.XYs
  NodesEnergyPoints map[int64]plotter.XYs
}

func (net *Network) AddNode(n *Node) error
func (net *Network) Simulate() error
func (net *Network) CheckNodes() int
func (net *Network) PopulateEnergyPoints()
func (net *Network) PopulateNodesAlivePoints()
```

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych zmiennych struktury sieci (Network):

- protokół (‘Protocol‘) - zmienna publiczna przechowuje obiekt definiujący protokół komunikacji pomiędzy węzłami.
- stacja bazowa (‘BaseStation‘) - zmienna publiczna przechowuje obiekt węzła głównego (‘sink‘).

- węzły (‘Nodes‘) - zmienna publiczna przechowuje obiekty węzłów w sieci. Implementacja przy wykorzystaniu dziennika (hashmap), który umożliwia operacje zapisu i odczytu w procesach równoległych. Kluczem dziennika jest unikatowy identyfikator węzła.
- runda (‘Round‘) - zmienna publiczna przechowuje numer aktualnej rundy symulacji.
- maksymalna ilość rund (‘MaxRounds‘) - zmienna publiczna przechowuje maksymalną ilość rund symulacji. W przypadku osiągnięcia wartości ‘Round‘ równej ‘MaxRounds‘, symulacja zostanie przerwana.
- długość wiadomości (‘MsgLength‘) - zmienna publiczna przechowuje informację o całkowitej wielkości wiadomości (mierzonej w bajtach) jaka generowana jest podczas rundy przez każdy z węzłów. W skład tej wartości wchodzą dane pomiarowe i dodatkowy nakład informacji powstały w wyniku enkapsulacji.
- GNUPlotNodes, GNUPlotTotalEnergy - zmienne publiczne przechowujące parametry rund wykorzystywane do generowania grafów przy użyciu narzędzia GNUPlot.
- PlotTotalEnergy, PlotNodes, NodesAlivePoints, NodesEnergyPoints - zmienne publiczne przechowujące parametry rund wykorzystywane do generowania grafów przy użyciu biblioteki plotter.

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych funkcji węzła (Node):

- AddNode - funkcja pozwala na dodanie węzła do sieci.
- Simulate - funkcja umożliwia rozpoczęcie symulacji.
- CheckNodes - funkcja sprawdza ilość sprawnych węzłów w sieci.
- PopulateEnergyPoints - funkcja sprawdza oraz przechowuje stany energetyczne węzłów.
- PopulateNodesAlivePoints - funkcja sprawdza oraz przechowuje ilość sprawnych węzłów w sieci.

Koszty transmisji

Transmisja i odbiór danych w sieci obarczony jest kosztem energetycznym. Model transmisji przedstawiony w rozdziale ‘Model transmisji‘ wymaga zdefiniowania wartości liczbowych dla czterech parametrów:

```
const (
  // Energy values measured in [J/byte].
  E_ELEC = 40e-9
  E_RX   = 4e-9
  E_MP   = 0.0104e-12
  E_FS   = 80e-12
)
```

Protokół (Protocol)

Protokół (‘Protocol‘) reprezentuje model protokołu komunikacji między węzłami.

Poniżej przedstawiona został interfejs protokołu oraz definicje funkcji przynależące do interfejsu.

```
type Protocol interface {
  Setup(net *Network) ([]int64, error)
  SetNodes(int)
  SetClusters(int)
}
```

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych funkcji węzła (Node):

- Setup - funkcja konfiguruje węzły w sieci zgodnie z zaimplementowanym protokołem.
- SetNodes - funkcja definiuje ilość węzłów w sieci. Wartość ta jest niezbędna do wyznaczania parametrów w protokołach (np. LEACH, PEGASIS).
- SetClusters - funkcja definiuje ilość klastrów w sieci. Wartość ta jest niezbędna do wyznaczania parametrów w protokołach (np. LEACH, PEGASIS).

Poniższa lista przedstawia zaimplementowane protokoły:

- Komunikacja bezpośrednia (Direct Communication) - wszystkie węzły w sieci komunikują się bezpośrednio z węzłem głównym (‘sink‘).
- LEACH (LEACH) - wszystkie węzły w sieci komunikują się zgodnie z topologią ustaloną w procesie konfiguracji LEACH.

- PEGASIS (PEGASIS) - wszystkie węzły w sieci komunikują się zgodnie z topologią ustaloną w procesie konfiguracji PEGASIS.

4.1.3 Moduł symulatora (Simulator)

Symulator (‘Simulator‘) odpowiada za tworzenie środowisk symulacyjnych. Pojedynczy obiekt symulatora pozwala na zbudowanie wielu scenariuszy symulacji, a następnie ich uruchomienie oraz wygenerowanie metryk. Dane wyjściowe mogą zostać podane dalszej analizie przy wykorzystaniu zewnętrznych narzędzi (np. GNUPlot).

```
type Simulator struct {
  namespace map[string]bool
  config    map[string]*config.Config
  network   map[string]*core.Network

  plotTotalEnergy *plot.Plot // An amount of total energy in the
      network per round.
  plotNodes      *plot.Plot // A number of alive nodes in the
      network per round.
}

func Create() (*Simulator, error)

func (s *Simulator) AddScenario(name string, conf *config.Config)
    error
func (s *Simulator) Run() error
func (s *Simulator) ExportPlots(filepath string) error
func (s *Simulator) ExportGNUPlots(filepath string) error

func createAndPopulateFile(filepath string, data []string) error
func (s *Simulator) create(name string, conf *config.Config) error
func createPlot(title, x, y string) (*plot.Plot, error)
func (s *Simulator) plotter() error
```

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych zmiennych struktury symulatora (‘Simulator‘):

- przestrzeń nazw (‘namespace‘) - zmienna prywatna przechowuje nazwy symulacji, które w jednoznaczny sposób identyfikują sieć i konfigurację.
- przestrzeń konfiguracji (‘config‘) - zmienna prywatna przechowuje obiekt

konfiguracji (Config) dla każdej symulacji. Kluczem dziennika jest nazwa symulacji.

- przestrzeń sieci (‘network‘) - zmienna prywatna przechowuje obiekt sieci (Network) dla każdej symulacji. Kluczem dziennika jest nazwa symulacji.

Poniższa lista przedstawia oraz opisuje znaczenie poszczególnych funkcji symulatora (‘Simulator‘):

- Setup - funkcja konfiguruje węzły w sieci zgodnie z zaimplementowanym protokołem.
- SetNodes - funkcja definiuje ilość węzłów w sieci. Wartość ta jest niezbędna do wyznaczania parametrów w protokołach (np. LEACH, PEGASIS).
- SetClusters - funkcja definiuje ilość klastrów w sieci. Wartość ta jest niezbędna do wyznaczania parametrów w protokołach (np. LEACH, PEGASIS).

4.2 Obsługa systemu

4.2.1 Konfiguracja środowiska i kompilacja

Poprawna konfiguracja środowiska wymaga instalacji niezbędnych bibliotek i pakietów.

1. Instalacja kompilatora i bibliotek Golang w wersji 1.10.1, lub wyższej.
2. Instalacja kompilatora Protocol Buffer w wersji 3.6, lub wyżej.

Weryfikacja konfiguracji Golang:

```
! Poprawna konfiguracja Golang.
$ which go
/usr/local/go/bin/go
$ go version
go version go1.10.1 linux/amd64
```

Weryfikacja konfiguracji protoc:

```
! Poprawna konfiguracja protoc.
$ which protoc
/usr/local/bin/protoc
$ protoc -version
libprotoc 3.6.0
```

Proces kompilacji (z poziomu folderu z kodem projektu):

```
$ pwd
/home/<username>/go/src/github.com/keadwen/msc_project
$ go build .
```

4.2.2 Generowanie konfiguracji

Wcześniej opisany plik konfiguracyjny jest elementem niezbędnym do uruchomienia symulacji.

Przykładowy plik konfiguracyjny (example.pbtxt) znajdują się w folderze proto:

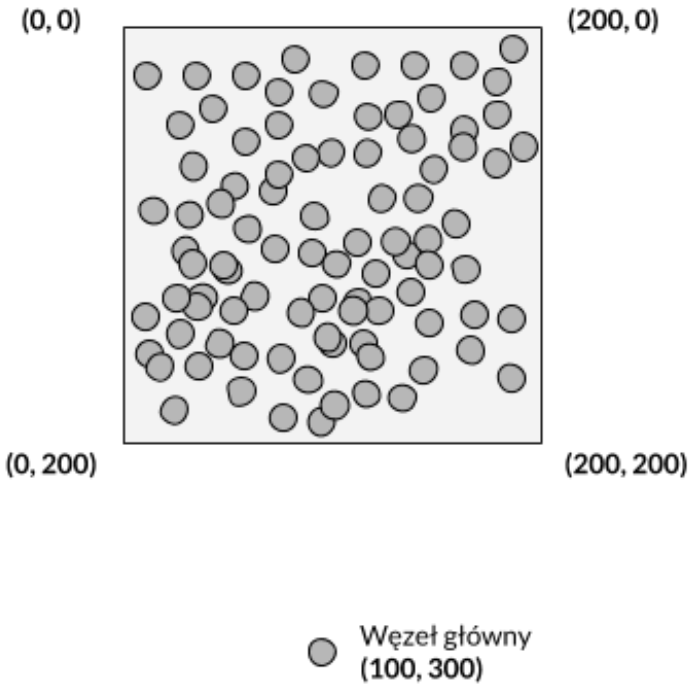
```
protocol: 1
nodes: <
  id: 0
  initial_energy: 1.0e4
  location: <
    X: 0
    Y: 0
  >
  energy_cost <
  >
  time_delay: <
  >
>
nodes: <
  id: 1
  initial_energy: 10.0e-6
  location: <
    X: 300.0
    Y: 0
  >
  energy_cost <
```

```
  >
  time_delay: <
  >
>
nodes <
  id: 2
  initial_energy: 20.0e-6
  location: <
    X: 500.0
    Y: 0
  >
  energy_cost: <
  >
  time_delay: <
  >
>
```

Oprogramowanie umożliwia również generowanie scenariuszy w dynamiczny sposób. W tym przypadku podczas uruchamiania oprogramowania (proces opisany w sekcji 'Uruchamianie scenariuszy') użytkownik nie podaje pliku konfiguracyjnego. Oprogramowanie stworzy identyczny zestaw konfiguracyjny dla każdego zaimplementowanego protokołu. Parametry wygenerowanej konfiguracji dostępne poniżej:

- Protokół: Bezpośrednia komunikacja, LEACH, PEGASIS
- Maksymalna ilość rund: 25000
- Proporcja węzłów typu klaster do wszystkich węzłów: 0.15
- Liczba węzłów głównych ('sink'): 1
- Energia węzła głównego: 100 [J]
- Lokalizacja węzła głównego na osi X: 100
- Lokalizacja węzła głównego na osi Y: 300
- Liczba węzłów: 200
- Energia węzłów: 1 [J]
- Lokalizacja węzłów na osi X: [0, 200]
- Lokalizacja węzłów na osi Y: [0, 200]

Poniższa ilustracja przedstawia rozstawienie węzłów w sieci wygenerowanych dynamicznie:



4.2.3 Uruchamianie scenariuszy

Przykład uruchamiania projektu z dwoma plikami konfiguracyjnymi:

```
$ /go/src/github.com/keadwen/msc_project/msc_project \
    --config_file=example1.proto,example2.proto
```

Przykład uruchamiania projektu bez pliku konfiguracyjnego:

```
$ /go/src/github.com/keadwen/msc_project/msc_project
```

4.2.4 Generowanie wykresów

Generowanie wykresów odbywa się przy wykorzystaniu narzędzia GNUPlot. Proces tworzenia wykresów nie jest zautomatyzowany. Jest to dodatkowa czynność, która musi zostać wykonana manualnie po pomyślnie zakończonym procesie symulacji. W początkowym etapie tworzenia oprogramowania, zastosowane zostało rozwiązanie przy wykorzystaniu biblioteki plotter, która generowała dwa wykresy:

- Wykres ilości aktywnych węzłów w każdej rundzie.

- Wykres energii całkowitej posiadanej przez wszystkie węzły w każdej rundzie.

Rozwiązanie to pomimo zalet związanych z automatycznym tworzeniem wykresów, nie pozwalało na generowanie ich w jakości spełniającej wymagania pracy dyplomowej.

W momencie poprawnego zakończenia symulacji, folder plotdata powinien posiadać zestaw plików w formacie GNU plot. Pliki te należy następnie przekierować do GNUPlot. Poniżej przedstawiona została lista operacji umożliwiająca wygenerowanie wykresów:

```
~/go/src/github.com/keadwen/msc_project$ gnuplot
gnuplot> set grid
gnuplot> plot \
  "< cat ./latex/gnuplot_data/test_1/leach200_m32/*" using 1:2
     smooth sbezier ls 1 title "LEACH (m=32B)", \
  "< cat ./latex/gnuplot_data/test_1/pegasis200_m32/*" using 1:2
     smooth sbezier ls 2 title "PEGASIS (m=32B)", \
  "< cat ./latex/gnuplot_data/test_1/direct200_m32/*" using 1:2
     smooth sbezier ls 4 title "DIRECT (m=32B)"
```

4.2.5 Dodawanie nowych protokołów

Architektura systemu umożliwia tworzenie nowych protokołów wymiany informacji pomiędzy węzłami. Poprawna implementacja wymaga modyfikacji oprogramowania w kilku miejscach:

1. Rozszerzenie definicji enum E_Protocol w proto/config.proto.
2. Rozszerzenie mapProtocol w simulator/simulator.go
3. Stworzenie nowego pliku .go w folderze core. Struktura reprezentująca nowy protokołu musi posiadać zestaw funkcji zgodny z interfejsem Protocol.

Po wykonaniu wszystkich z powyższych kroków, protokół może zostać wykorzystany w symulacji. Należy pamiętać, że oprogramowanie i protocol buffers będą wymagały ponownej kompilacji. W przeciwnym wypadku zmiany nie będą widoczne.

Rozdział 5

Opracowanie wyników eksperymentów

W tym rodziale przedstawione zostaną wyniki uzyskane w procesie symulacji przy wykorzystaniu autorskiego oprogramowania.

5.1 Dane wejściowe

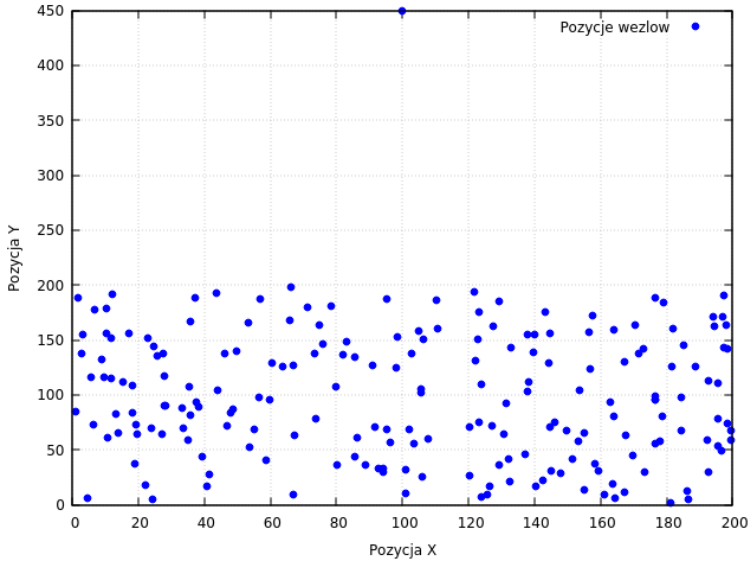
Oprogramowanie posiada implementacje trzech algorytmów: komunikację bezpośrednią, LEACH oraz PEGASIS (dokładniejszy opis działania protokołów został przedstawiony w poprzednich rozdziałach). W celu porównania efektywności energetycznej pomiędzy trzema rozwiązaniami, niezbędne jest precyzyjne zdefiniowanie testów. W systemie posiadamy cztery parametry, których wartości będą modyfikowane i efekty zmian będą porównywane:

- Wielkość wiadomości - parametr ten definiuje wielkość wiadomości wysyłanej pomiędzy węzłami (mierzone w Bajtach).
- Ilość węzłów w sieci - parametr ten definiuje liczbę węzłów w pojedynczej sieci i symulacji.
- Wielkość obszaru sieci - parametr ten definiuje wielkość obszaru wymiany danych w sieci.
- Parametr ‘p‘ - wartość prawdopodobieństwa wypromowania węzła na węzeł typu cluster head.

W celu zapewnienia poprawnej jakości porównywania testów, każdy z scenariuszy posiada dedykowany plik konfiguracyjny. W ten sposób zapewnione jest stworzenie identycznego środowiska testowego, niezależnego od testowanego protokołu.

5.2 Scenariusze

Każdy z testów oraz wartości przedstawione na grafie są wynikami uzyskanymi w procesie wielokrotnego uruchomienia identycznego scenariusza. Ilość uruchomień zdefiniowana dla każdego z testów (domyślenie wynosi 100). W celach testów wygenerowana została ‘konfiguracja domyślna‘, w której zdefiniowana jest lokalizacja 200 unikatowych węzłów o energii początkowej wynoszącej 2 [J], rozdystrybuowanych w sposób pseudolosowy na przestrzeni X: [0, 200], Y: [0, 200]. Lokalizacja węzła typu ‘sink‘ to X:100, Y: 450. Scenariusze wykorzystujące np. 50 węzłów, korzystają z podzbioru węzłów domyślnych, czyli z pierwszych 50 węzłów konfiguracji domyślnej. Poniższy rysunek przedstawia rozmieszczenie 200 węzłów standardowych oraz 1 węzła głównego na płaszczyźnie sieci.



Rysunek 5.1: Rozmieszczenie 200 węzłów w sieci.

5.2.1 Test 1 - Wielkość wiadomości

Test ‘Wielkość wiadomości‘ ma na celu przedstawienie zależności efektywności energetycznej dla trzech algorytmów dla różnych wielkości wiadomości wysyłanej pomiędzy węzłami.

Lista plików konfiguracyjnych:

```
~/go/src/github.com/keadwen/msc_projec$ tree testdata/test_1/
testdata/test_1/
```

```
  direct200_m128.pbtxt
  direct200_m256.pbtxt
  direct200_m32.pbtxt
  direct200_m64.pbtxt
  leach200_m128.pbtxt
  leach200_m256.pbtxt
  leach200_m32.pbtxt
  leach200_m64.pbtxt
  pegasis200_m128.pbtxt
  pegasis200_m256.pbtxt
  pegasis200_m32.pbtxt
  pegasis200_m64.pbtxt

0 directories, 12 files
```

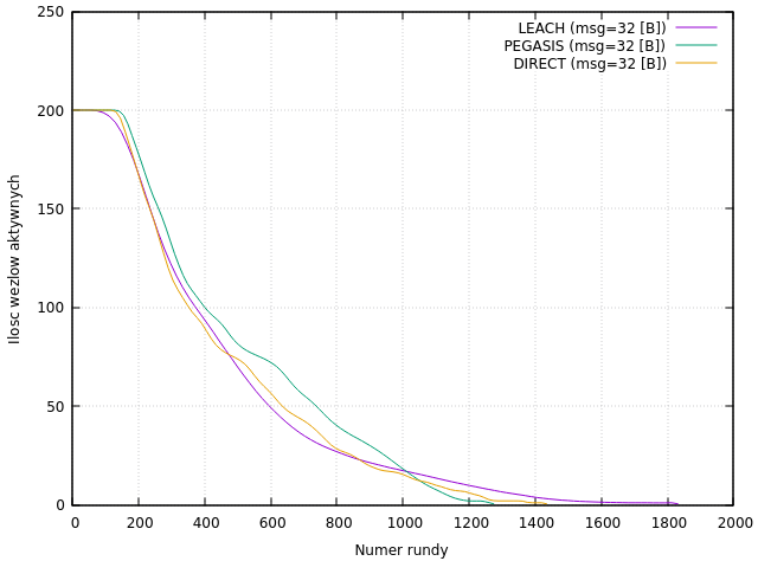
Uruchomienie oprogramowania:

```
~/go/src/github.com/keadwen/msc_project$ ./msc_project
    --repeat_config=100 --config_files=\
testdata/test_1/direct200_m32.pbtxt,\
testdata/test_1/direct200_m64.pbtxt,\
testdata/test_1/direct200_m128.pbtxt,\
testdata/test_1/direct200_m256.pbtxt,\
testdata/test_1/leach200_m32.pbtxt,\
testdata/test_1/leach200_m64.pbtxt,\
testdata/test_1/leach200_m128.pbtxt,\
testdata/test_1/leach200_m256.pbtxt,\
testdata/test_1/pegasis200_m32.pbtxt,\
testdata/test_1/pegasis200_m64.pbtxt,\
testdata/test_1/pegasis200_m128.pbtxt,\
testdata/test_1/pegasis200_m256.pbtxt
```

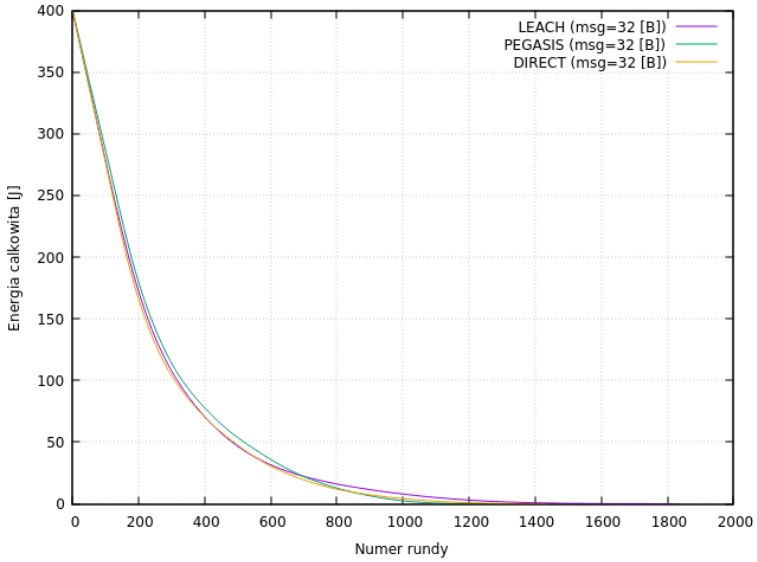
Tabela uzyskanych wyników:

Uzyskane rezultaty			
Protokół	Wielkość wiadomości (B)	Kwantyl	Wartość
DIRECT	32	-	1434

Uzyskane rezultaty			
Protokół	Wielkość wiadomości (B)	Kwantyl	Wartość
LEACH	32	1	1431
		5	1499
		10	1518
		25	1571
		50	1635
		75	1697
		90	1765
		95	1783
		99	1805
		100	1831
PEGASIS	32	1	1136
		5	1147
		10	1152
		25	1166
		50	1185
		75	1203
		90	1224
		95	1234
		99	1262
		100	1274



Rysunek 5.2: Funkcja ilości węzłów aktywnych w rundzie (m=32B).



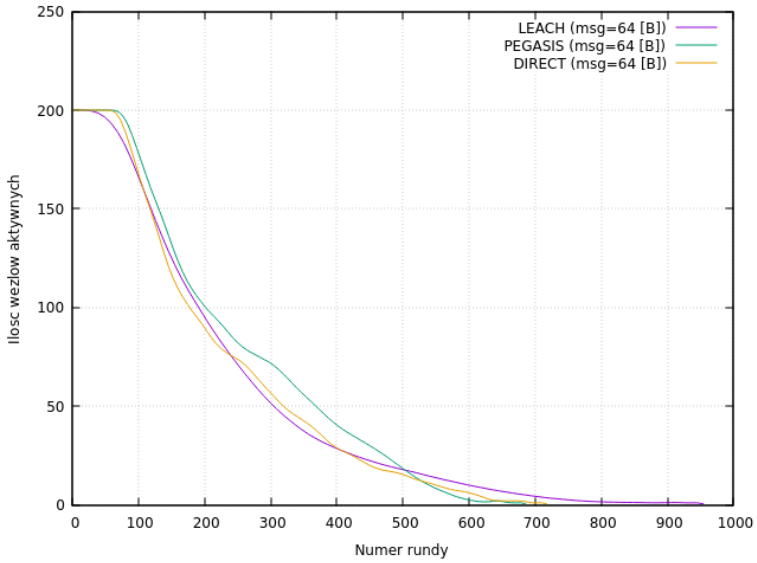
Rysunek 5.3: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (m=32B).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 1434. Dla LEACH uzyskano skrócenie czasu pracy na poziomie -1.3% dla wartości minimalnej (1416), oraz wydłużenie czasu pracy na poziomie +14.0% dla mediany (1635), +27.7% dla wartości maksymalnej (1636). Dla PEGASIS uzyskano skrócenie czasu pracy na po-

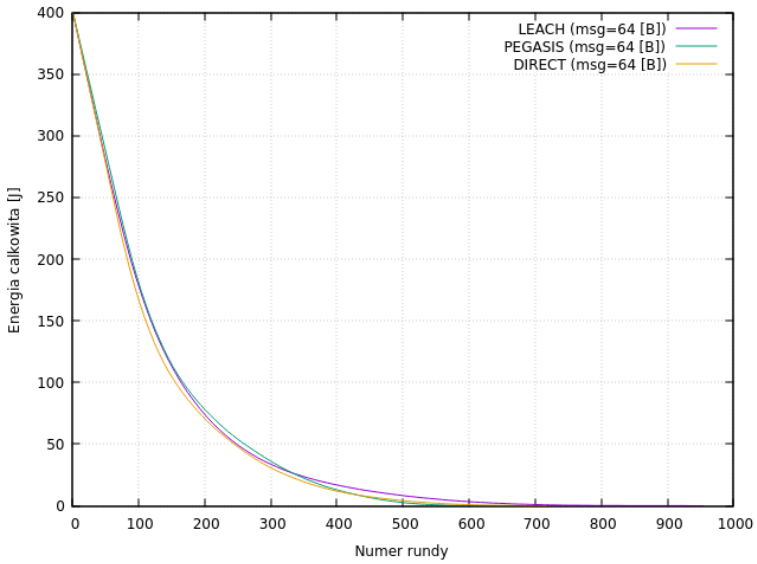
ziomie -22.5% dla wartości minimalnej (1112), -17.4% dla mediany (1184), -11.2% dla wartości maksymalnej (1274). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest zbliżony. Zauważyć należy jednak, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 1000 rund.

Uzyskane rezultaty			
Protokół	Wielkość wiadomości (B)	Kwantyl	Wartość
DIRECT	64	-	717
LEACH	64	1	775
		5	760
		10	770
		25	801
		50	831
		75	872
		90	893
		95	908
		99	935
		100	954
PEGASIS	64	1	576
		5	584
		10	587
		25	594
		50	608
		75	622
		90	629
		95	635
		99	669
		100	685



Rysunek 5.4: Funkcja ilości węzłów aktywnych w rundzie (m=64B).



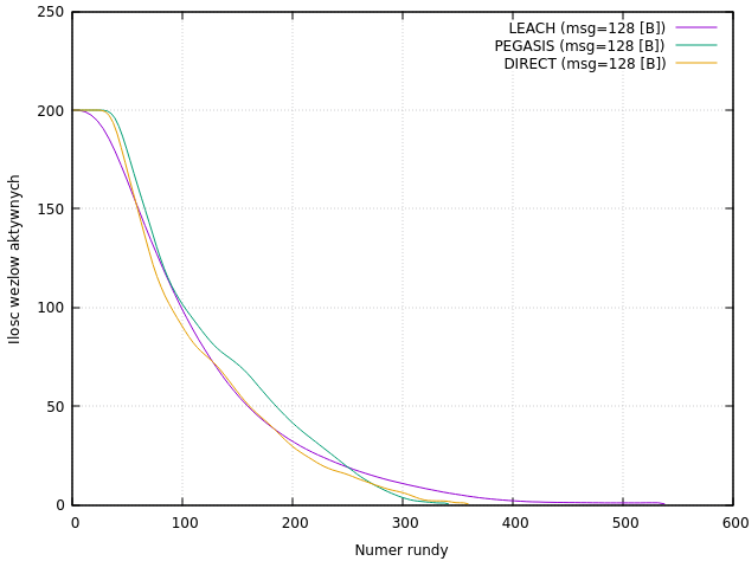
Rysunek 5.5: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (m=64B).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 717. Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +2.1% dla wartości minimalnej (732), +15.8% dla mediany (831), +31.1% dla wartości maksymalnej (954). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -20.4% dla wartości minimalnej (571), -15.3%

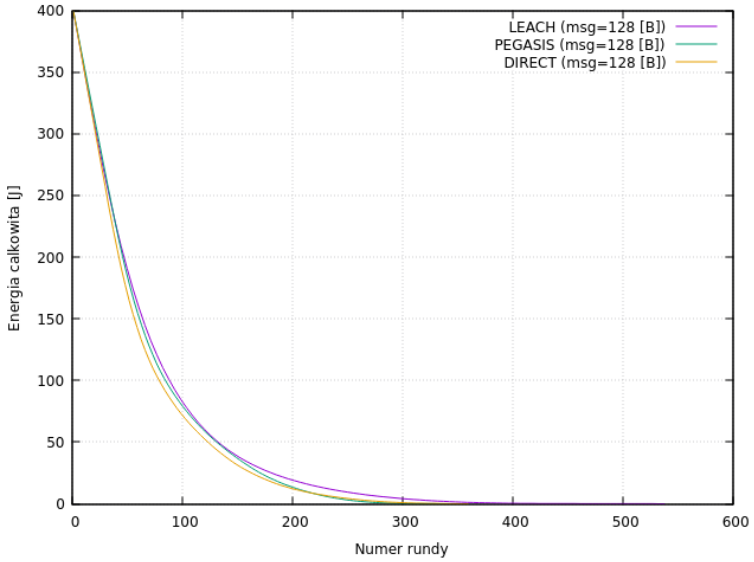
dla mediany (608), -4.5% dla wartości maksymalnej (685). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest zbliżony. Zauważyć należy jednak, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 500 rund.

Uzyskane rezultaty			
Protokół	Wielkość wiadomości (B)	Kwantyl	Wartość
DIRECT	128	-	359
LEACH	128	1	388
		5	392
		10	399
		25	409
		50	432
		75	453
		90	479
		95	492
		99	508
		100	537
PEGASIS	128	1	291
		5	295
		10	300
		25	305
		50	312
		75	318
		90	327
		95	330
		99	336
		100	341



Rysunek 5.6: Funkcja ilości węzłów aktywnych w rundzie (m=128B).



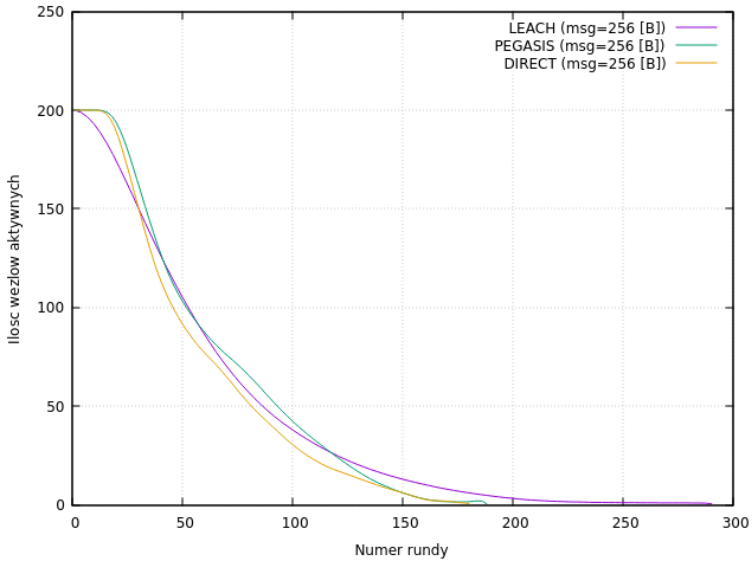
Rysunek 5.7: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (m=128B).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359. Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +3.9% dla wartości minimalnej (373), +20.3% dla mediany (432), +49.6% dla wartości maksymalnej (537). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -19.2% dla wartości minimalnej (290), -13.1%

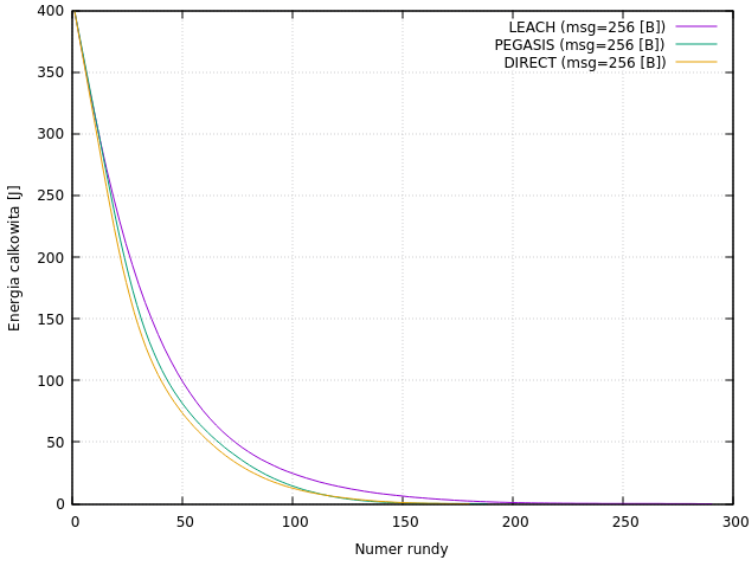
dla mediany (312), -5.0% dla wartości maksymalnej (341). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest zbliżony. Zauważyć należy jednak, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 250 rund.

Uzyskane rezultaty			
Protokół	Wielkość wiadomości (B)	Kwantyl	Wartość
DIRECT	256	-	180
LEACH	256	1	207
		5	212
		10	215
		25	224
		50	239
		75	252
		90	267
		95	281
		99	289
		100	290
PEGASIS	256	1	155
		5	157
		10	158
		25	162
		50	166
		75	170
		90	175
		95	179
		99	181
		100	188



Rysunek 5.8: Funkcja ilości węzłów aktywnych w rundzie (m=256B).



Rysunek 5.9: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (m=256B).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 180. Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +8.3% dla wartości minimalnej (195), +32.8% dla mediany (239), +61.1% dla wartości maksymalnej (290). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -15.6% dla wartości minimalnej (152), -7.8%

dla mediany (166), oraz wydłużenie czasu pracy na poziomie +4.4% dla wartości maksymalnej (188). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest zbliżony. Zauważyć należy jednak, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 120 rund.

5.2.2 Test 2 - Ilość węzłów w sieci

Test 'Ilość węzłów w sieci' ma na celu przedstawienie zależności efektywności energetycznej dla trzech algorytmów względem ilości węzłów znajdujących się w sieci.

List plików konfiguracyjnych:

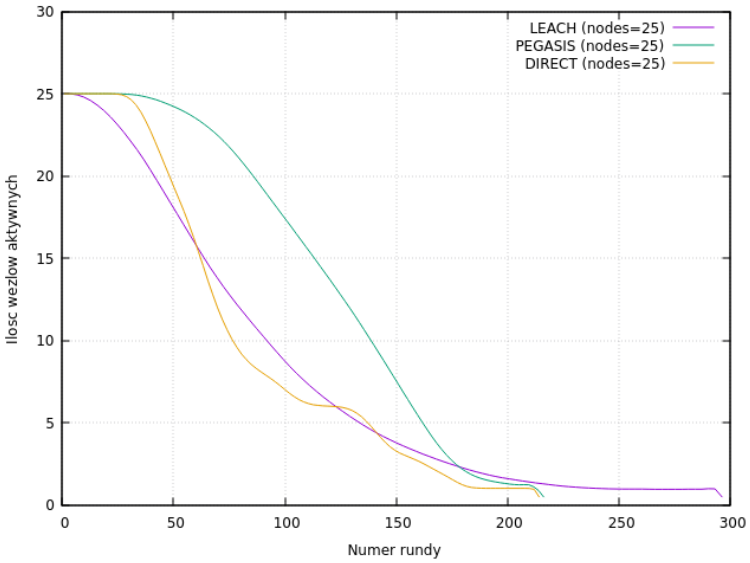
```
~/go/src/github.com/keadwen/msc_project$ tree testdata/test_2/
testdata/test_2/
  direct100.pbtxt
  direct150.pbtxt
  direct200.pbtxt
  direct25.pbtxt
  direct50.pbtxt
  leach100.pbtxt
  leach150.pbtxt
  leach200.pbtxt
  leach25.pbtxt
  leach50.pbtxt
  pegasis100.pbtxt
  pegasis150.pbtxt
  pegasis200.pbtxt
  pegasis25.pbtxt
  pegasis50.pbtxt

0 directories, 15 files
```

Uruchomienie oprogramowania:

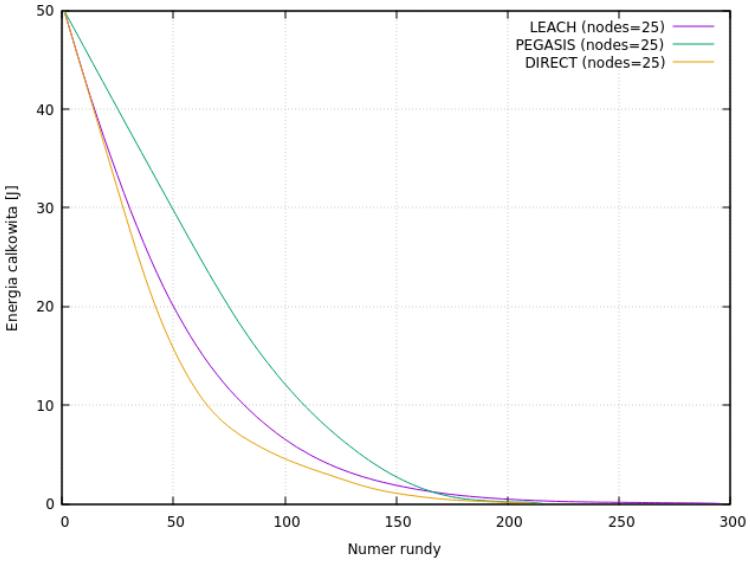
```
~/go/src/github.com/keadwen/msc_project$ ./msc_project
    --repeat_config=100 --config_files=\
testdata/test_2/direct25.pbtxt,\
```

```
testdata/test_2/direct50.pbtxt,\
testdata/test_2/direct100.pbtxt,\
testdata/test_2/direct150.pbtxt,\
testdata/test_2/direct200.pbtxt,\
testdata/test_2/leach25.pbtxt,\
testdata/test_2/leach50.pbtxt,\
testdata/test_2/leach100.pbtxt,\
testdata/test_2/leach150.pbtxt,\
testdata/test_2/leach200.pbtxt,\
testdata/test_2/pegasis25.pbtxt,\
testdata/test_2/pegasis50.pbtxt,\
testdata/test_2/pegasis100.pbtxt,\
testdata/test_2/pegasis150.pbtxt,\
testdata/test_2/pegasis200.pbtxt
```

Uzyskane rezultaty			
Protokół	Ilość węzłów	Kwantyl	Wartość
DIRECT	25	-	214
LEACH	25	1	178
		5	186
		10	197
		25	212
		50	230
		75	246
		90	263
		95	272
		99	287
		100	296
PEGASIS	25	1	170
		5	172
		10	176
		25	182
		50	187
		75	195
		90	201
		95	206
		99	215
		100	216



Rysunek 5.10: Funkcja ilości węzłów aktywnych w rundzie (n=25).



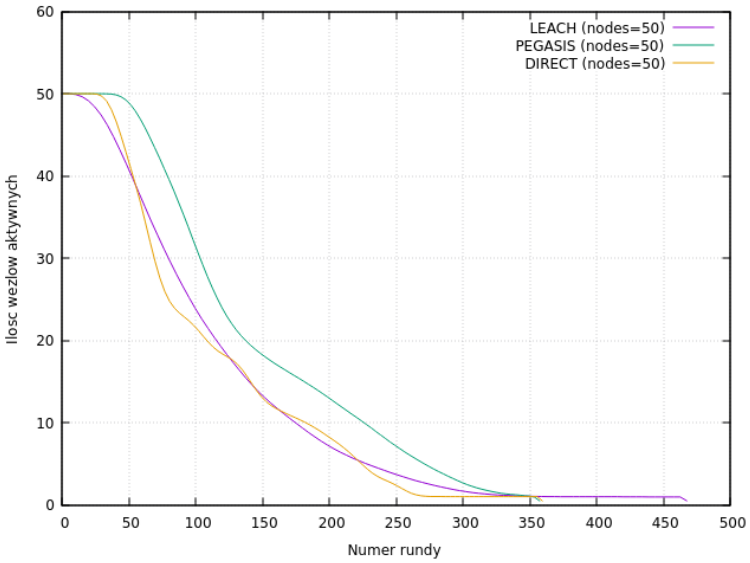
Rysunek 5.11: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (n=25).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 214 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -16.8% dla wartości minimalnej (178), oraz wydłużenie czasu pracy na poziomie +7.2% dla mediany (229), +38.3% dla wartości maksymalnej

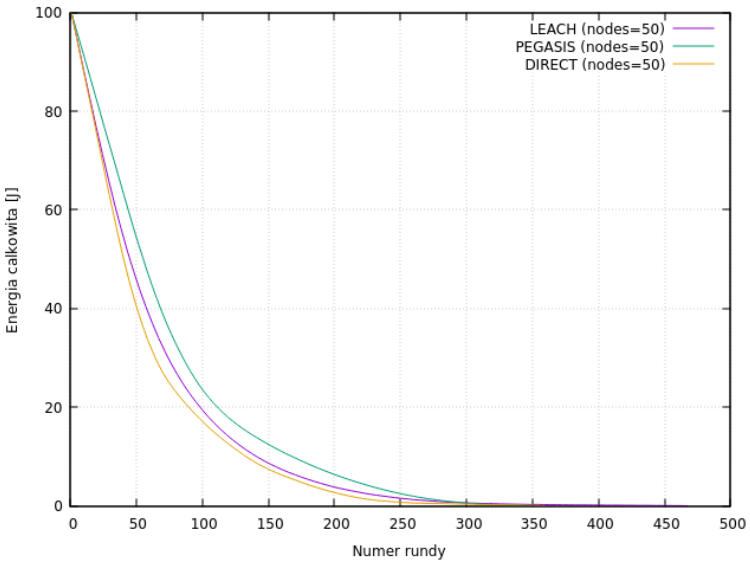
(296). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -32.9% dla wartości minimalnej (165), -12.9% dla mediany (186), oraz wydłużenie czasu pracy na poziomie +0.9% dla wartości maksymalnej (216). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów uwidacznia różnice pomiędzy efektywnością energetyczną protokołów. Zauważyć można, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 150 rund.

Uzyskane rezultaty			
Protokół	Ilość węzłów	Kwantyl	Wartość
DIRECT	50	-	359
LEACH	50	1	294
		5	307
		10	321
		25	357
		50	382
		75	410
		90	435
		95	443
		99	461
		100	467
PEGASIS	50	1	285
		5	292
		10	302
		25	310
		50	320
		75	332
		90	342
		95	348
		99	354
		100	357



Rysunek 5.12: Funkcja ilości węzłów aktywnych w rundzie (n=50).



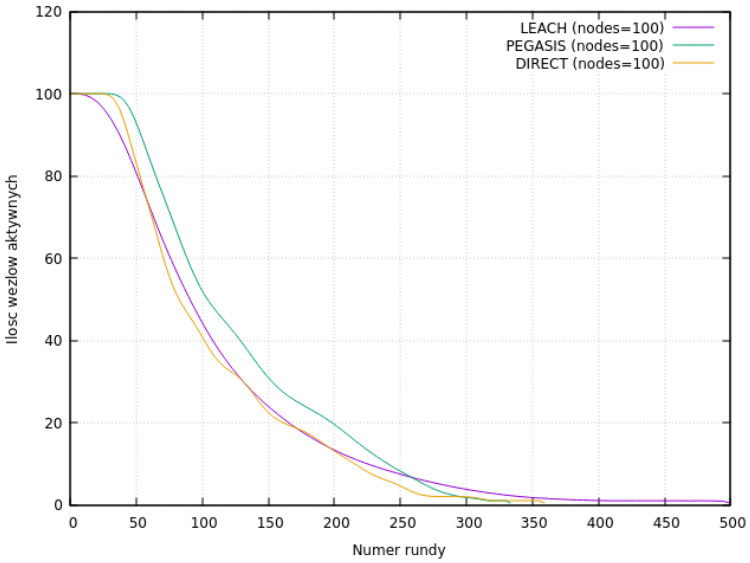
Rysunek 5.13: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (n=50).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -22.3% dla wartości minimalnej (279), oraz wydłużenie czasu pracy na poziomie +6.4% dla mediany (382), +30.1% dla wartości maksymalnej

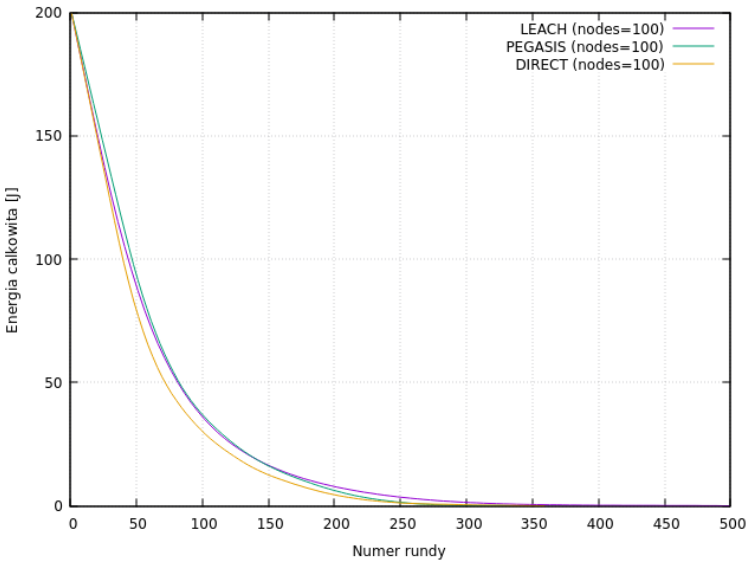
(467). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -34.2% dla wartości minimalnej (272), -10.9% dla mediany (320), -0.6% dla wartości maksymalnej (357). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest zbliżony. Zauważyć można, że PEGASIS cechuje się lepszą sprawnością energetyczną przez pierwsze 250 rund. Komunikacja bezpośrednia jest widocznie mniej efektywna.

	Uzyskane rezultaty		
Protokół	Ilość węzłów	Kwantyl	Wartość
DIRECT	100	-	359
LEACH	100	1	326
		5	347
		10	356
		25	385
		50	413
		75	439
		90	464
		95	477
		99	493
		100	499
PEGASIS	100	1	267
		5	274
		10	278
		25	285
		50	291
		75	301
		90	311
		95	314
		99	319
		100	333



Rysunek 5.14: Funkcja ilości węzłów aktywnych w rundzie (n=100).



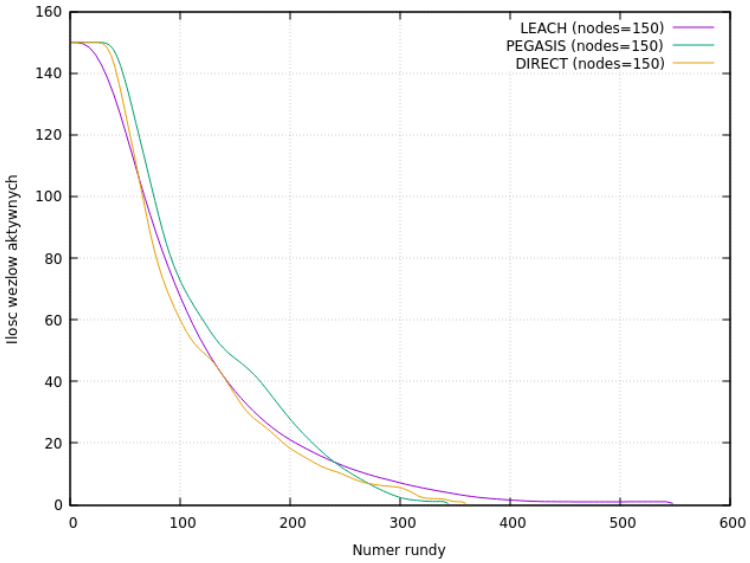
Rysunek 5.15: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (n=100).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -9.7% dla wartości minimalnej (324), oraz wydłużenie czasu pracy na poziomie +14.9% dla mediany (413), +39.0% dla wartości maksymalnej

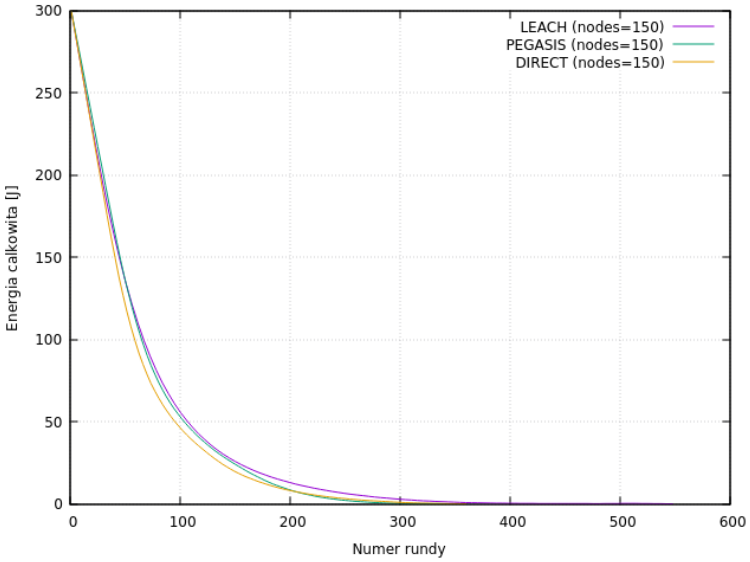
(499). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -25.6% dla wartości minimalnej (267), -18.9% dla mediany (291), -7.2% dla wartości maksymalnej (333). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie dla LEACH i PEGASIS jest niemal identyczny. Komunikacja bezpośrednia jest widocznie mniej efektywna.

Uzyskane rezultaty			
Protokół	Ilość węzłów	Kwantyl	Wartość
DIRECT	150	-	359
LEACH	150	1	369
		5	378
		10	384
		25	399
		50	425
		75	447
		90	468
		95	484
		99	499
		100	547
PEGASIS	150	1	287
		5	293
		10	297
		25	304
		50	312
		75	322
		90	330
		95	333
		99	339
		100	343



Rysunek 5.16: Funkcja ilości węzłów aktywnych w rundzie (n=150).



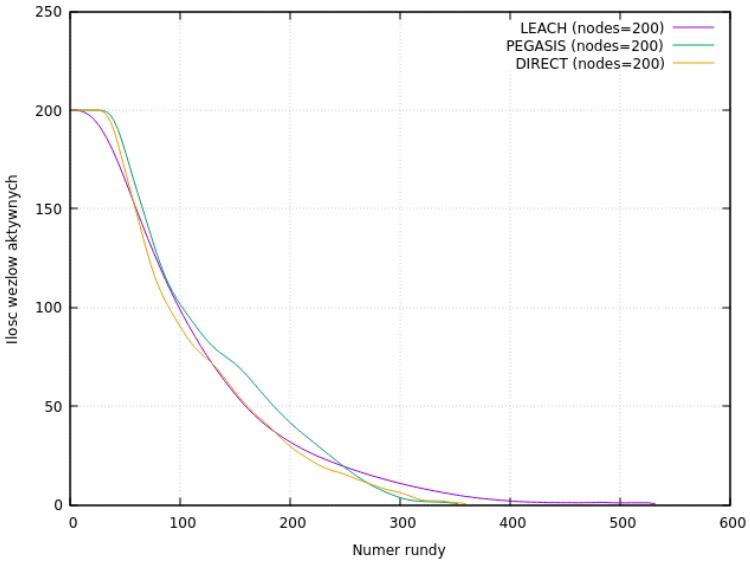
Rysunek 5.17: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (n=150).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +0.3% dla wartości minimalnej (360), +18.2% dla mediany (425), +52.4% dla wartości maksymalnej (547). Dla PEGASIS uzyskano skrócenie

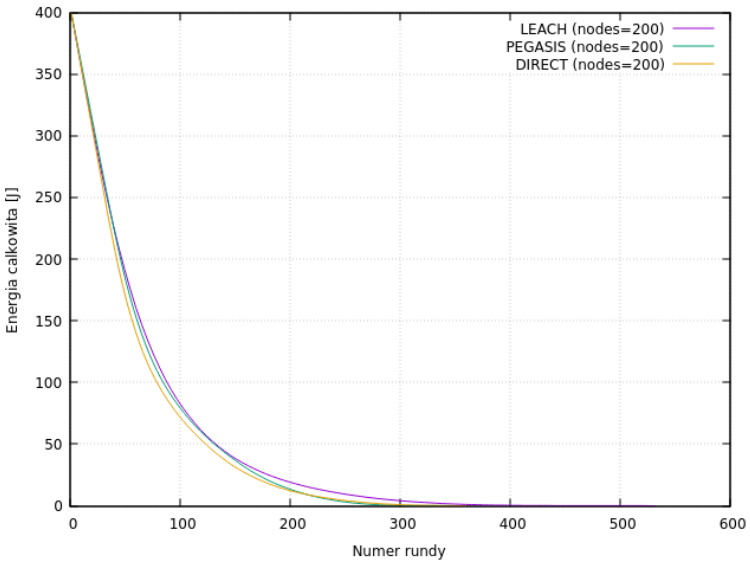
czasu pracy na poziomie -20.3% dla wartości minimalnej (286), -13.1% dla mediany (312), -4.5% dla wartości maksymalnej (343). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest niemal identyczny.

Uzyskane rezultaty			
Protokół	Ilość węzłów	Kwantyl	Wartość
DIRECT	200	-	359
LEACH	200	1	382
		5	398
		10	405
		25	415
		50	430
		75	449
		90	467
		95	475
		99	499
		100	531
PEGASIS	200	1	297
		5	299
		10	302
		25	307
		50	313
		75	321
		90	329
		95	335
		99	349
		100	352



Rysunek 5.18: Funkcja ilości węzłów aktywnych w rundzie (n=200).



Rysunek 5.19: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (n=200).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +3.9% dla wartości minimalnej (373), +19.6% dla mediany (430), +47.9% dla wartości maksymalnej (531). Dla PEGASIS uzyskano skrócenie

czasu pracy na poziomie -18.4% dla wartości minimalnej (293), -12.8% dla mediany (313), -1.9% dla wartości maksymalnej (352). Pomimo niższej wartości maksymalnej uzyskanych rund pracy węzłów w PEGASIS, ilość węzłów aktywnych w przeciągu symulacji jest wyższa niż w przypadku DIRECT czy LEACH. Zjawisko to pozwala na uzyskanie danych od większej puli węzłów aktywnych.

Rozkład ilości energii całkowitej wszystkich węzłów w rundzie jest niemal identyczny.

5.2.3 Test 3 - Wielkość obszaru sieci

Test 'Wielkość obszaru sieci' ma na celu przedstawienie zależności efektywności energetycznej dla trzech algorytmów względem wielkości obszaru sieci na której znajdują się węzły w sieci.

Istotna informacja: lokalizacja węzłów w scenariuszu 200 jest niezmienna względem pozostałych testów. Wielkość obszaru sieci była manipulowana przy użyciu skali (0.25, 0.50, 2.00, 4.00). Oznacza to, że koordynaty (X, Y) węzłów zostały przemnożone przez wartość skali. Proporcja odległości pomiędzy węzłami zostaje taka sama, jednakże zmianie ulega koszt komunikacji, gdyż funkcja kosztów energetycznych jest zależna od dystansu.

List plików konfiguracyjnych:

```
~/go/src/github.com/keadwen/msc_project$ tree testdata/test_3/
testdata/test_3/
  direct_xy050.pbtxt
  direct_xy100.pbtxt
  direct_xy200.pbtxt
  direct_xy400.pbtxt
  direct_xy800.pbtxt
  leach_xy050.pbtxt
  leach_xy100.pbtxt
  leach_xy200.pbtxt
  leach_xy400.pbtxt
  leach_xy800.pbtxt
  pegasis_xy050.pbtxt
  pegasis_xy100.pbtxt
  pegasis_xy200.pbtxt
  pegasis_xy400.pbtxt
  pegasis_xy800.pbtxt
```

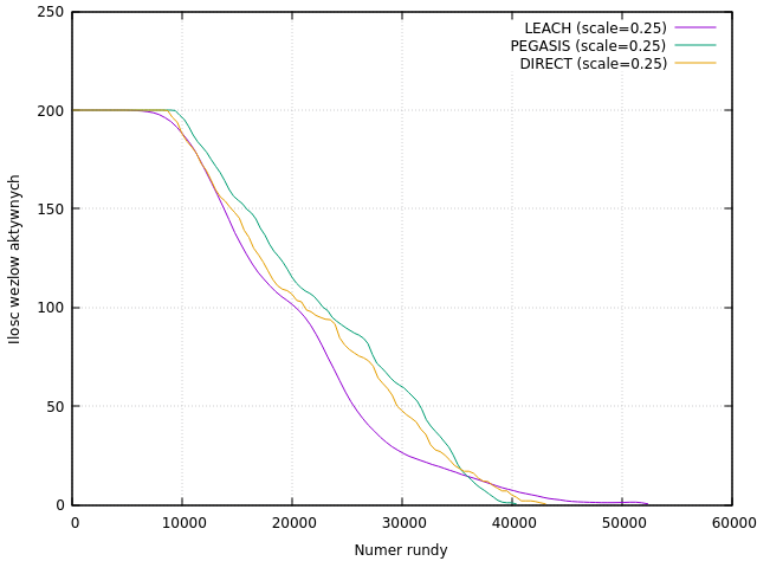
Uruchomienie oprogramowania:

```
~/go/src/github.com/keadwen/msc_project$ ./msc_project
   --repeat_config=100 --config_files=\
testdata/test_3/direct_xy050.pbtxt,\
testdata/test_3/direct_xy100.pbtxt,\
testdata/test_3/direct_xy200.pbtxt,\
testdata/test_3/direct_xy400.pbtxt,\
testdata/test_3/direct_xy800.pbtxt,\
testdata/test_3/leach_xy050.pbtxt,\
testdata/test_3/leach_xy100.pbtxt,\
testdata/test_3/leach_xy200.pbtxt,\
testdata/test_3/leach_xy400.pbtxt,\
testdata/test_3/leach_xy800.pbtxt,\
testdata/test_3/pegasis_xy050.pbtxt,\
testdata/test_3/pegasis_xy100.pbtxt,\
testdata/test_3/pegasis_xy200.pbtxt,\
testdata/test_3/pegasis_xy400.pbtxt,\
testdata/test_3/pegasis_xy800.pbtxt
```

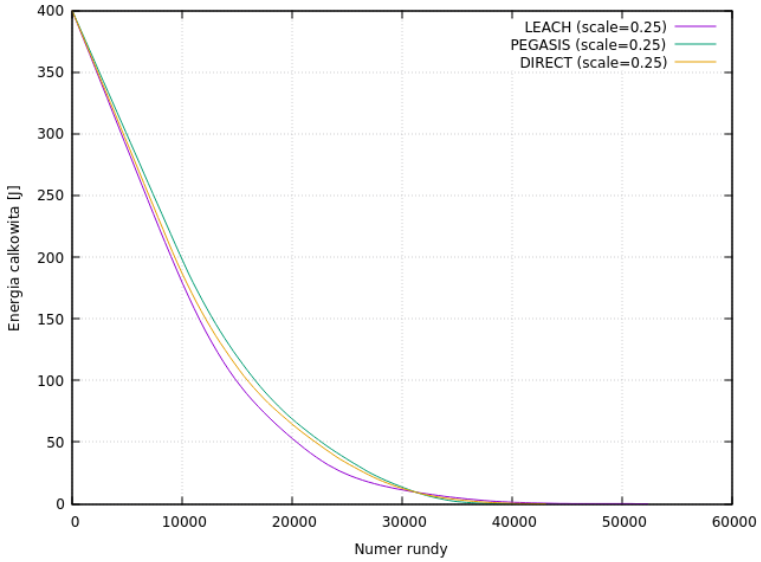
Poniższa tabela przedstawia wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Uzyskane rezultaty			
Protokół	Skala sieci	Kwantyl	Wartość
DIRECT	0.25	-	42976
LEACH	0.25	1	43417
		5	44197
		10	45130
		25	45790
		50	46910
		75	48151
		90	49310
		95	50097
		99	51909
		100	52278

Uzyskane rezultaty			
Protokół	Skala sieci	Kwantyl	Wartość
PEGASIS	0.25	1	38521
		5	38724
		10	38847
		25	39050
		50	39279
		75	39544
		90	39756
		95	39828
		99	40038
		100	40316



Rysunek 5.20: Funkcja ilości węzłów aktywnych w rundzie.

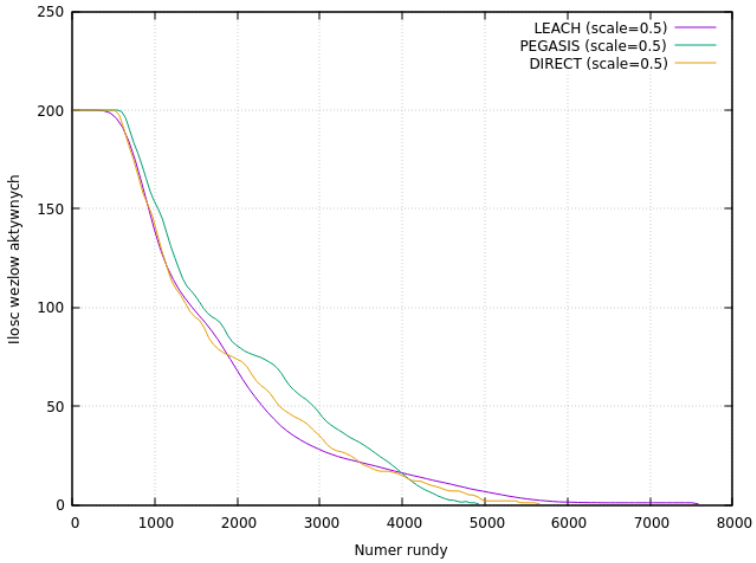


Rysunek 5.21: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie.

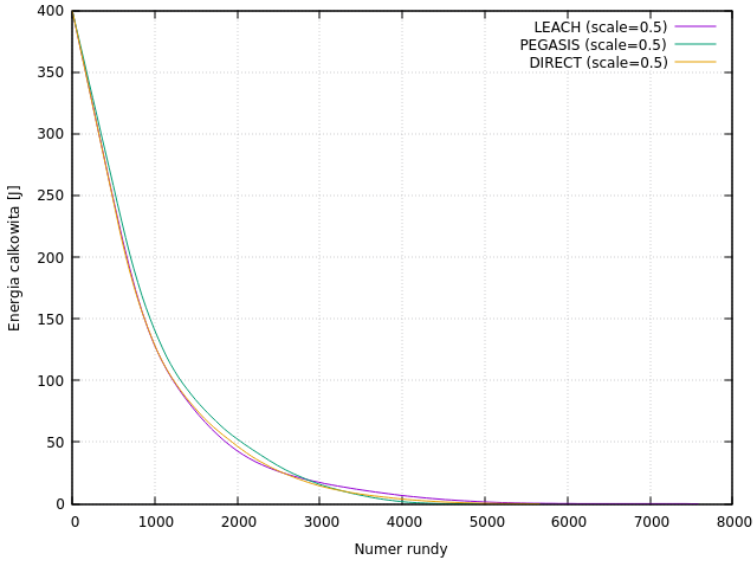
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 42976 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +0.9% dla wartości minimalnej (43360), +9.2% dla mediany (46910), +21.6% dla wartości maksymalnej (52278). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -10.7% dla wartości minimal-

nej (38362), -8.6% dla mediany (39279), -6.2% dla wartości maksymalnej (40316). Odległości węzłów do 'sink' są czterokrotnie mniejsze niż w konfiguracji podstawowej. Czas pracy węzłów uległ znaczącemu wydłużeniu, gdyż koszt energetyczny transmisji jest niższy.

Protokół	Skala sieci	Kwantyl
DIRECT	0.50	-
LEACH	0.50	1
		5
		10
		25
		50
		75
		90
		95
		99
		100
PEGASIS	0.50	1
		5
		10
		25
		50
		75
		90
		95
		99
		100



Rysunek 5.22: Funkcja ilości węzłów aktywnych w rundzie.

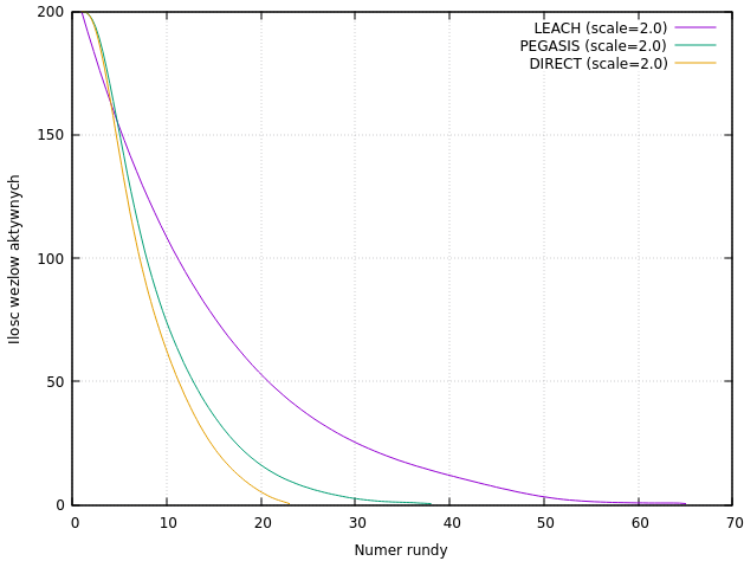


Rysunek 5.23: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie.

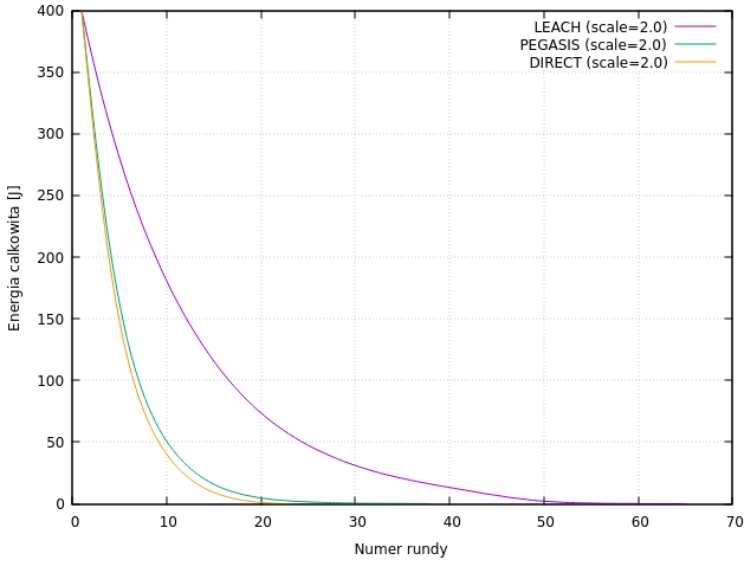
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 5657 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -2.3% dla wartości minimalnej (5529), oraz wydłużenie czasu pracy na poziomie +11.2% dla mediany (6293), +34.0% dla wartości maksymalnej (7580). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie

-22.1% dla wartości minimalnej (4406), -19.1% dla mediany (4577), -13.0% dla wartości maksymalnej (4920). Odległości węzłów do 'sink' są dwukrotnie mniejsze niż w konfiguracji podstawowej. Czas pracy węzłów uległ znaczącemu wydłużeniu, gdyż koszt energetyczny transmisji jest niższy.

Uzyskane rezultaty			
Protokół	Skala sieci	Kwantyl	Wartość
DIRECT	2.00	-	23
LEACH	2.00	1	49
		5	51
		10	51
		25	53
		50	55
		75	58
		90	60
		95	61
		99	63
		100	65
PEGASIS	2.00	1	27
		5	29
		10	29
		25	30
		50	32
		75	33
		90	34
		95	36
		99	37
		100	38



Rysunek 5.24: Funkcja ilości węzłów aktywnych w rundzie.

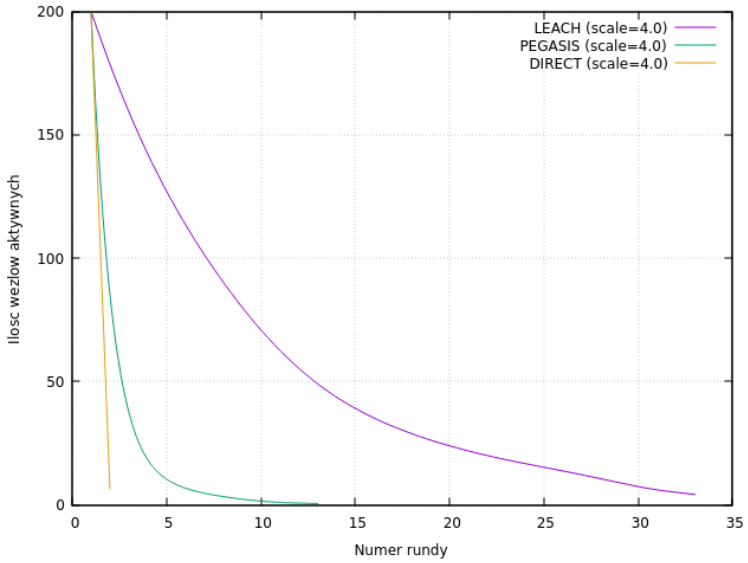


Rysunek 5.25: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie.

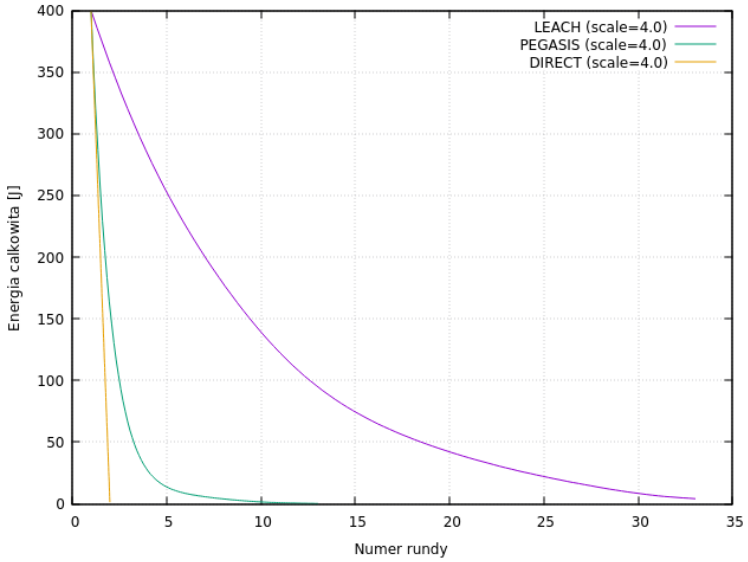
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 23 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +213.0% dla wartości minimalnej (49), +239.1% dla mediany (55), +282.6% dla wartości maksymalnej (65). Dla PEGASIS uzyskano wydłużenie czasu pracy na poziomie +117.4% dla wartości minimalnej (27), +139.1% dla

mediany (32), +165.2% dla wartości maksymalnej (38). Odległości węzłów do 'sink' są dwukrotnie większa niż w konfiguracji podstawowej. Czas pracy węzłów uległ znaczącemu skróceniu, gdyż koszt energetyczny transmisji jest wyższy.

Uzyskane rezultaty			
Protokół	Skala sieci	Kwantyl	Wartość
DIRECT	4.00	-	2
LEACH	4.00	1	27
		5	28
		10	29
		25	29
		50	30
		75	31
		90	32
		95	32
		99	33
		100	33
PEGASIS	4.00	1	7
		5	8
		10	9
		25	9
		50	10
		75	11
		90	11
		95	12
		99	13
		100	13



Rysunek 5.26: Funkcja ilości węzłów aktywnych w rundzie.



Rysunek 5.27: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie.

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 2 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +1350.0% dla wartości minimalnej (27), +1500.0% dla mediany (30), +1650.0% dla wartości maksymalnej (33). Dla PEGASIS uzyskano wydłużenie czasu pracy na poziomie +350.0% dla wartości minimalnej (7), +500.0%

dla mediany (10), +650.0% dla wartości maksymalnej (13). Odległości węzłów do 'sink' są czterokrotnie większa niż w konfiguracji podstawowej. Czas pracy węzłów uległ znaczącemu skróceniu, gdyż koszt energetyczny transmisji jest wyższy.

5.2.4 Test 4 - Parametr 'p'

Test 'Parametr p' ma na celu przedstawienie zależności efektywności energetycznej dla trzech algorytmów względem wartości prawdopodobieństwa wypromowania węzła na węzeł typu cluster head.

Istotna informacja: w przypadku algorytmu komunikacji bezpośredniej parametr 'p' nie istnieje. Wynika to z charakterystyki komunikacji, gdyż w protokole tym nie występuje koncept węzła typu cluster head i każdy z węzłów przesyła informacje bezpośrednio do węzła głównego (sink). Dokonać można jednak założenia, że transmisja w protokole komunikacji bezpośredniej jest specjalnym przypadkiem sieci typu LEACH, w której parametr

$$p = 1.0$$

Oznacza to, że każdy z węzłów pełni rolę węzła cluster head, który nie posiada przynależących węzłów standardowych i dokonuje transmisji tylko swoich dane.

Lista plików konfiguracyjnych:

```
~/go/src/github.com/keadwen/msc_project$ tree testdata/test_4/
testdata/test_4/
  direct200.pbtxt
  leach200_p005.pbtxt
  leach200_p010.pbtxt
  leach200_p015.pbtxt
  leach200_p020.pbtxt
  leach200_p025.pbtxt
  leach200_p030.pbtxt
  pegasis200_p005.pbtxt
  pegasis200_p010.pbtxt
  pegasis200_p015.pbtxt
  pegasis200_p020.pbtxt
  pegasis200_p030.pbtxt

0 directories, 13 files
```

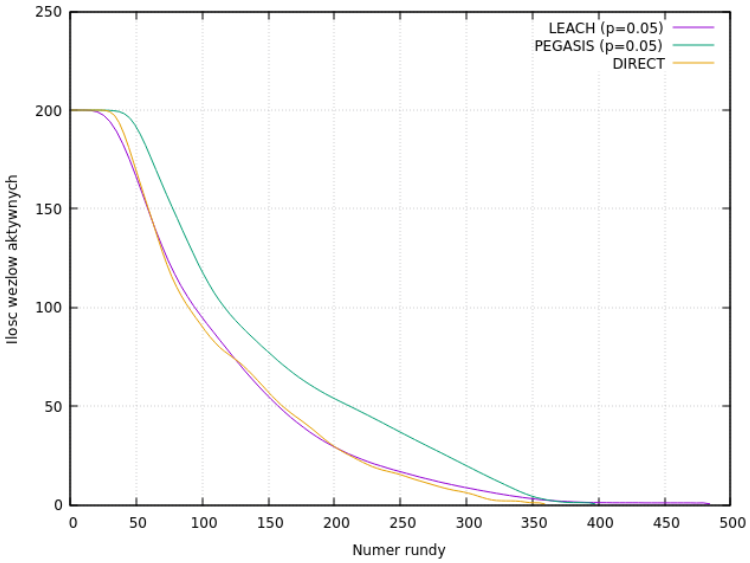
Uruchomienie oprogramowania:

```
~/go/src/github.com/keadwen/msc_project$ ./msc_project
   --repeat_config=100 --config_files=\
testdata/test_4/direct200.pbtxt,\
testdata/test_4/leach200_p005.pbtxt,\
testdata/test_4/leach200_p010.pbtxt,\
testdata/test_4/leach200_p015.pbtxt,\
testdata/test_4/leach200_p020.pbtxt,\
testdata/test_4/leach200_p025.pbtxt,\
testdata/test_4/leach200_p030.pbtxt,\
testdata/test_4/pegasis200_p005.pbtxt,\
testdata/test_4/pegasis200_p010.pbtxt,\
testdata/test_4/pegasis200_p015.pbtxt,\
testdata/test_4/pegasis200_p020.pbtxt,\
testdata/test_4/pegasis200_p025.pbtxt,\
testdata/test_4/pegasis200_p030.pbtxt
```

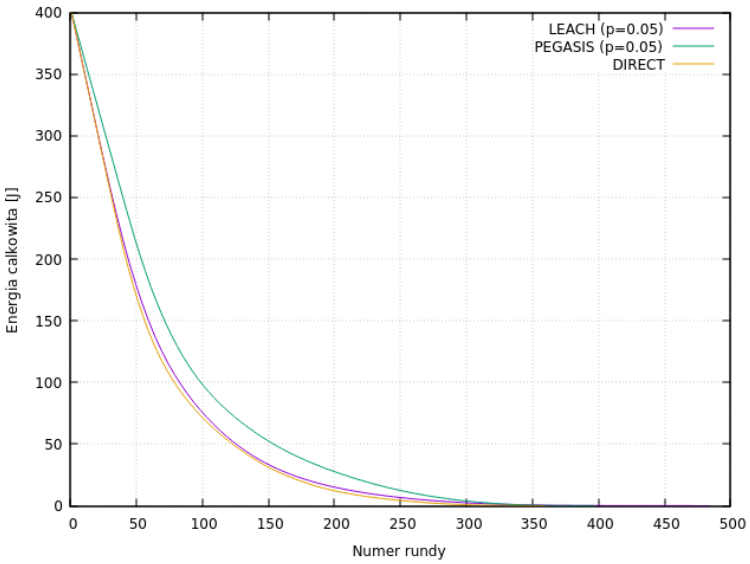
Poniższa tabela przedstawia wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.05	1	356
		5	363
		10	365
		25	383
		50	396
		75	417
		90	435
		95	455
		99	476
		100	484

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
PEGASIS	0.05	1	345
		5	351
		10	352
		25	356
		50	365
		75	372
		90	377
		95	380
		99	387
		100	389



Rysunek 5.28: Funkcja ilości węzłów aktywnych w rundzie (p=0.05).

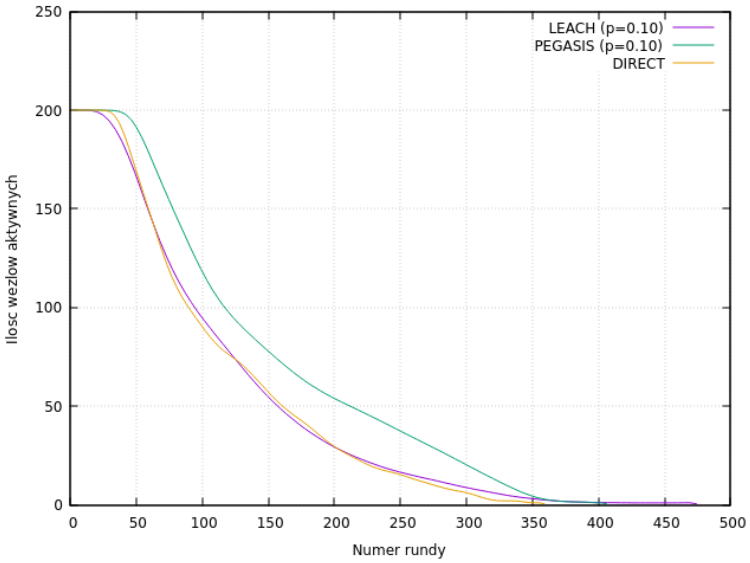


Rysunek 5.29: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.05).

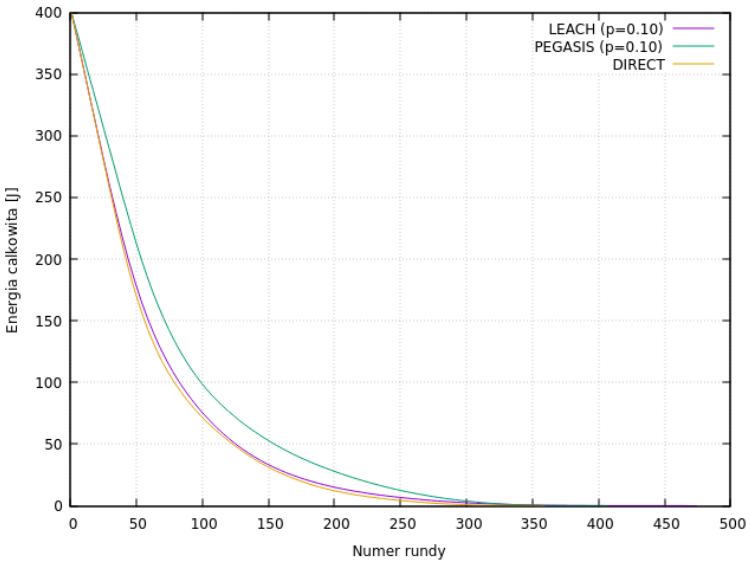
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego 'sink'). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -1.4% dla wartości minimalnej (354), oraz wydłużenie czasu pracy na poziomie +10.3% dla mediany (396), +34.8% dla wartości maksymalnej

(484). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -3.9% dla wartości minimalnej (345), oraz wydłużenie czasu pracy na poziomie +1.7% dla mediany (365), +8.4% dla wartości maksymalnej (389). Ilość węzłów typu cluster head jest dwukrotnie mniejsza niż w konfiguracji podstawowej.

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.10	1	364
		5	366
		10	373
		25	386
		50	400
		75	413
		90	430
		95	436
		99	455
		100	474
PEGASIS	0.10	1	341
		5	347
		10	351
		25	359
		50	364
		75	372
		90	377
		95	380
		99	388
		100	395



Rysunek 5.30: Funkcja ilości węzłów aktywnych w rundzie (p=0.10).

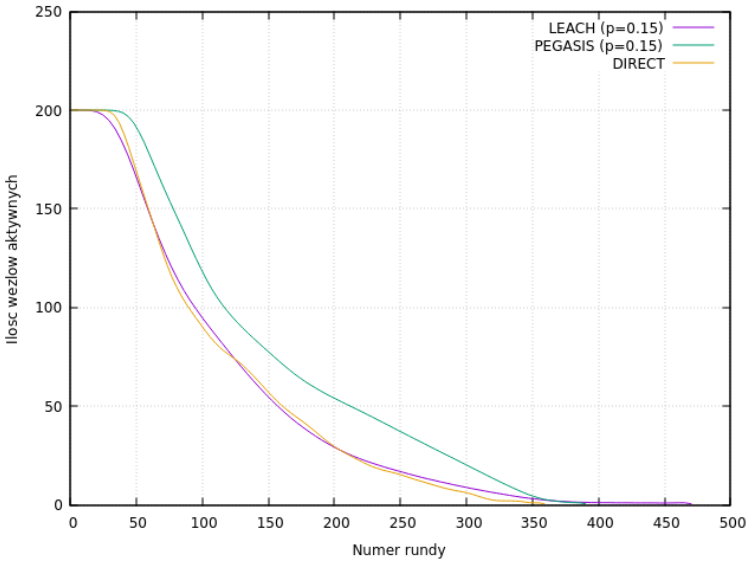


Rysunek 5.31: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.10).

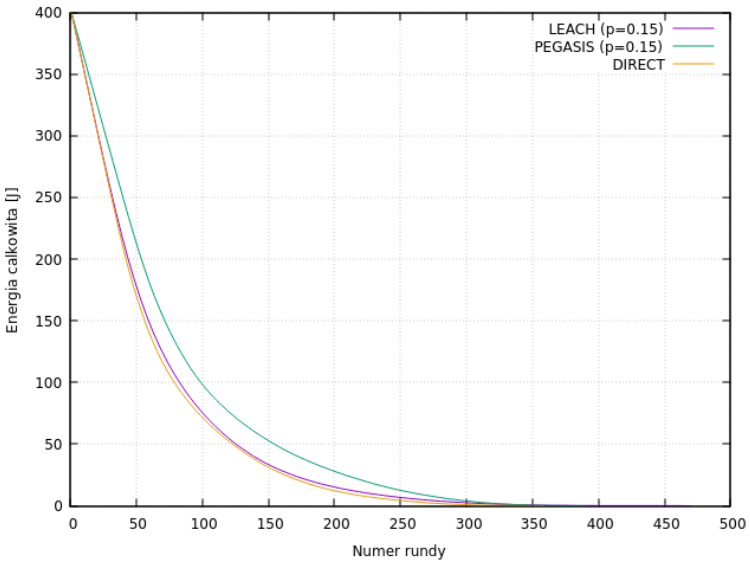
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +1.4% dla wartości minimalnej (365), +11.4% dla mediany (400), +32.0% dla wartości maksymalnej (474). Dla PEGASIS uzyskano skrócenie

czasu pracy na poziomie -5.3% dla wartości minimalnej (345), oraz wydłużenie czasu pracy na poziomie +1.3% dla mediany (364), +10.0% dla wartości maksymalnej (395).

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.15	1	357
		5	367
		10	371
		25	386
		50	399
		75	419
		90	434
		95	442
		99	455
		100	470
PEGASIS	0.15	1	340
		5	346
		10	350
		25	355
		50	365
		75	371
		90	382
		95	386
		99	402
		100	411



Rysunek 5.32: Funkcja ilości węzłów aktywnych w rundzie (p=0.15).

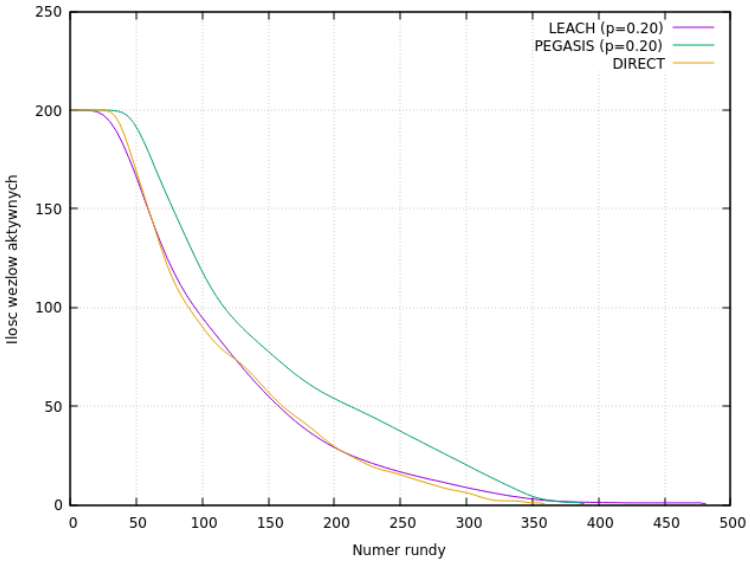


Rysunek 5.33: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.15).

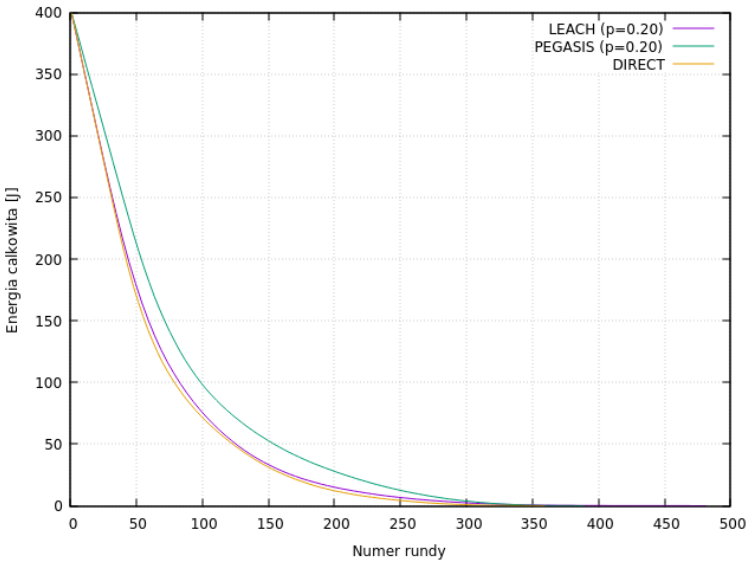
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -2.5% dla wartości minimalnej (350), oraz wydłużenie czasu pracy na poziomie +10.0% dla mediany (399), +30.9% dla wartości maksymalnej

(470). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -6.7% dla wartości minimalnej (335), oraz wydłużenie czasu pracy na poziomie +1.5% dla mediany (365), +14.5% dla wartości maksymalnej (411). Ilość węzłów typu cluster head jest o 50% większa niż w konfiguracji podstawowej.

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.20	1	362
		5	366
		10	372
		25	386
		50	403
		75	423
		90	442
		95	451
		99	471
		100	481
PEGASIS	0.20	1	347
		5	349
		10	352
		25	358
		50	367
		75	373
		90	378
		95	382
		99	388
		100	397



Rysunek 5.34: Funkcja ilości węzłów aktywnych w rundzie (p=0.20).

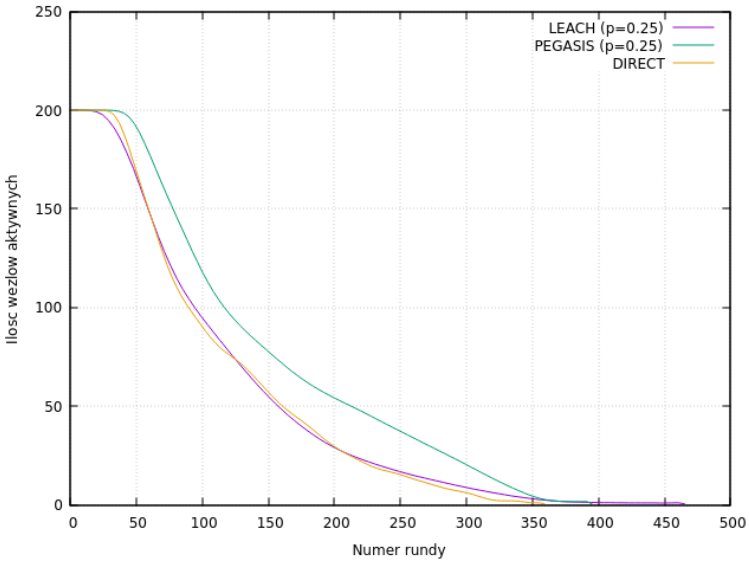


Rysunek 5.35: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.20).

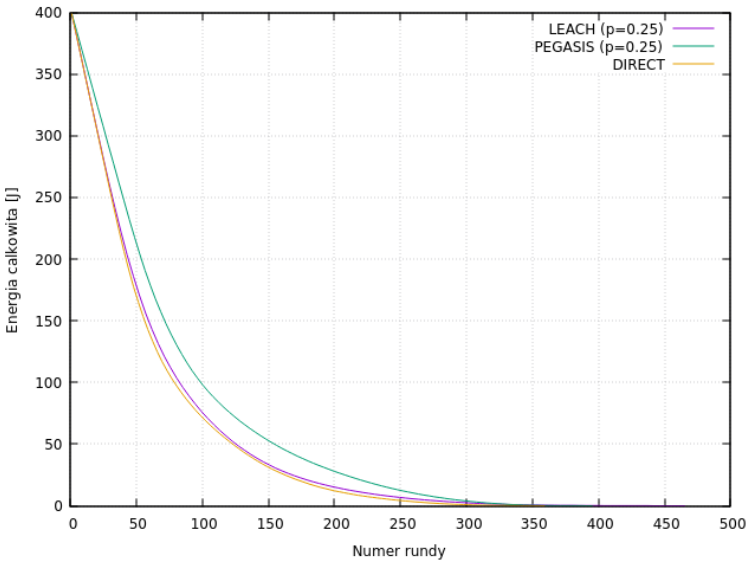
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +0.8% dla wartości minimalnej (362), +12.1% dla mediany (403), +34.0% dla wartości maksymalnej (481). Dla PEGASIS uzyskano skrócenie

czasu pracy na poziomie -4.2% dla wartości minimalnej (344), oraz wydłużenie czasu pracy na poziomie +2.2% dla mediany (367), +10.6% dla wartości maksymalnej (397). Ilość węzłów typu cluster head jest dwukrotnie większa niż w konfiguracji podstawowej.

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.25	1	359
		5	363
		10	371
		25	384
		50	404
		75	429
		90	439
		95	450
		99	454
		100	465
PEGASIS	0.25	1	339
		5	347
		10	345
		25	356
		50	362
		75	370
		90	381
		95	385
		99	397
		100	406



Rysunek 5.36: Funkcja ilości węzłów aktywnych w rundzie (p=0.25).

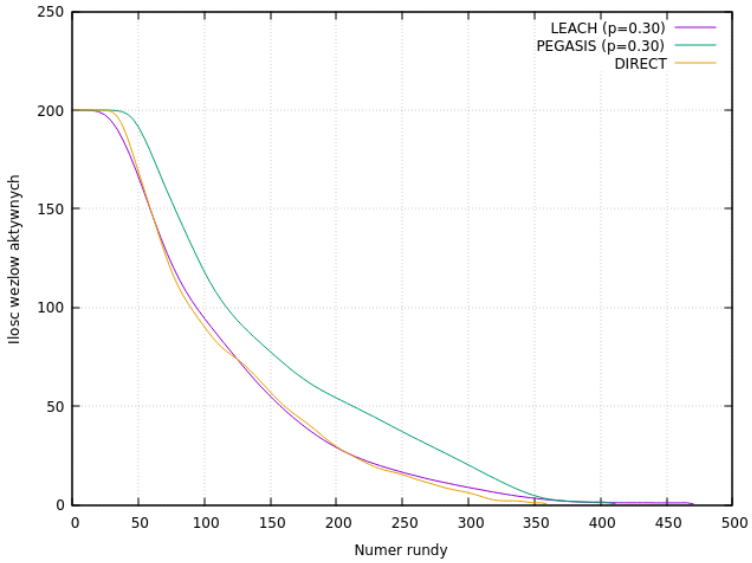


Rysunek 5.37: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.25).

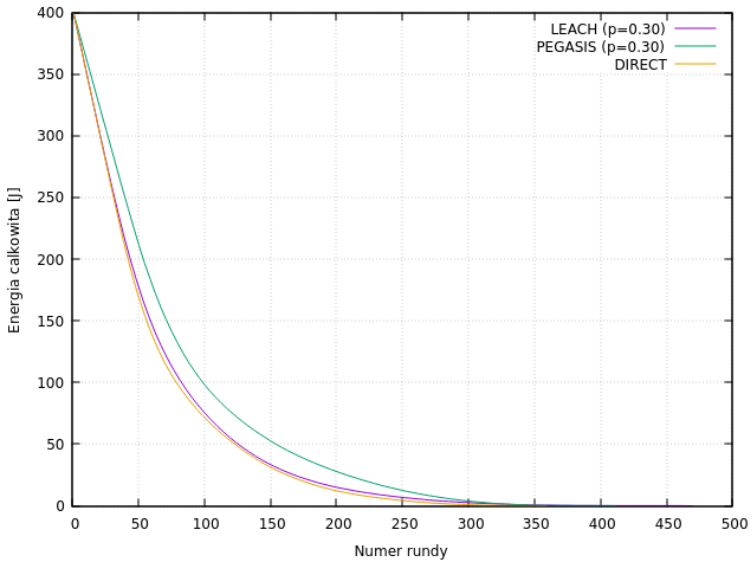
Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -0.3% dla wartości minimalnej (358), oraz wydłużenie czasu pracy na poziomie +12.4% dla mediany (404), +29.5% dla wartości maksymalnej

(465). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -4.2% dla wartości minimalnej (344), oraz wydłużenie czasu pracy na poziomie +2.2% dla mediany (367), +10.6% dla wartości maksymalnej (397). Ilość węzłów typu cluster head jest 150% większa niż w konfiguracji podstawowej.

Uzyskane rezultaty			
Protokół	Parametr ‘p‘	Kwantyl	Wartość
DIRECT	-	-	359
LEACH	0.30	1	352
		5	369
		10	373
		25	385
		50	399
		75	418
		90	431
		95	444
		99	456
		100	470
PEGASIS	0.30	1	348
		5	350
		10	354
		25	359
		50	366
		75	372
		90	377
		95	385
		99	389
		100	390



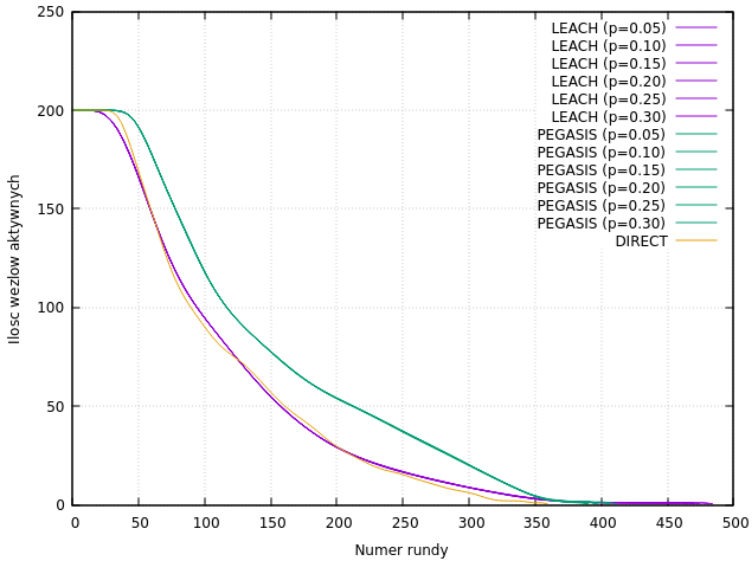
Rysunek 5.38: Funkcja ilości węzłów aktywnych w rundzie (p=0.30).



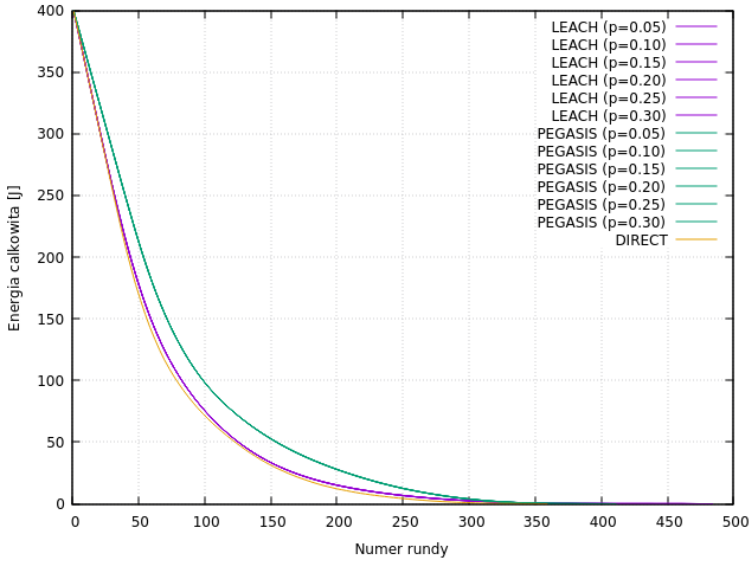
Rysunek 5.39: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.30).

Maksymalna ilość rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 359 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego ‘sink‘). Dla LEACH uzyskano skrócenie czasu pracy na poziomie -2.2% dla wartości minimalnej (351), oraz wydłużenie czasu pracy na poziomie +11.1% dla mediany (399), +30.9% dla wartości maksymalnej

(470). Dla PEGASIS uzyskano skrócenie czasu pracy na poziomie -4.7% dla wartości minimalnej (342), oraz wydłużenie czasu pracy na poziomie +1.7% dla mediany (365), +8.6% dla wartości maksymalnej (390). Ilość węzłów typu cluster head jest trzykrotnie większa niż w konfiguracji podstawowej.



Rysunek 5.40: Funkcje ilości węzłów aktywnych w rundzie (wszystkie konfiguracje parametru p).



Rysunek 5.41: Funkcje całkowitej ilości energii wszystkich węzłów w rundzie (wszystkie konfiguracje parametru p).

Powyższe dwa wykresy nakładają przebiegi funkcji uzyskane dla wszystkich sześciu scenariuszy testu. Zauważyć można zależność, że efektywność energetyczna LEACH oraz PEGASIS przynosi pozytywne rezultaty, pozwalające na wydłużenie czasu pracy węzłów.

Rozdział 6

Podsumowanie

Koncepcja i projekt systemu symulującego model bezprzewodowej sieci czujnikowej został opracowany przez autora pracy. Pierwszym etapem pracy było określenie wymagań oraz funkcjonalności systemu końcowego. Pierwsze szkice dokumentacji były regularnie konsultowane z promotorem pracy. Proces ten pozwolił na zebranie szczegółowych wymagań projektowych. W końcowym etapie zbierania wymagań, autor zdecydował się tworzeniem prototypu systemu symulującego. Prace te pozwoliły na wczesne przetestowanie założeń i pomysłów dotyczących dekompozycji problemu, których efekt zaowocował uproszczeniem całkowitej architektury systemu. Projekt zrealizowano w języku Go, przy wykorzystaniu rozszerzonego systemu kontroli wersji git.

Proces zbieranie wyników generowanych przez system symulacyjny został podzielony na cztery główne testy. W każdym z testów wybrany został jeden parametr konfiguracji, który podlegał wielokrotnym zmianom mającym na celu zaobserwowanie zależności czasu i efektywności pracy sieci.

Najistotniejszym wnioskiem zaobserwowanym zarówno w momencie tworzenia pracy, jak i procesie tworzenia wyników jest zależność między uzyskanymi wynikami maksymalnego czasu pracy, a konfiguracją sieci. Oznacza to, że niemalże niemożliwym jest określenie stałej (przewidywalnej) efektywności energetycznej sieci w momencie wdrożenia LEACH, PEGASIS czy innego algorytmu usprawniającego wymianę danych w bezprzewodowej sieci czujników. W przypadku scenariusza wygenerowanego w ramach pracy dyplomowej zauważyć można, że węzeł 171 (X: 4.508531144480791, Y: 6.256340981494829) znajduje się bardzo blisko ‘sink‘. Czas życia tego węzła będzie kilkukrotnie dłuższy niż w przypadku węzła najbardziej oddalonego, węzła 195 (X: 197.47820095322837, Y: 190.37079023341855).

Każdy z testów definiuje kilka wartości parametru modyfikowanego, np. domyślna wielkość wiadomości wynosi 128 [B], dlatego też test składa się

z mniejszych testów w których praca węzłów podlega symulacji przy wykorzystaniu identycznej lokalizacji węzłów zdefiniowanych w konfiguracji domyślnej. Jedyną wartością ulegającą modyfikacji była wielkości wiadomości wysyłanej pomiędzy węzłami. Wielkości te wynosiły 32, 64, 128 oraz 256 [B].

Najlepsze korzyści płynące z implementacji protokołów LEACH oraz PEGASIS uzyskano dla testu numer 3, w którym dystanse pomiędzy węzłami uległy powiększeniu. Dla sieci powiększone czterokrotnie, czyli o wielkości X: 800, Y: 800 (konfiguracja domyślna X: 200, Y: 200), uzyskano jedynie 2 rundy dla komunikacji bezpośredniej. Gdzie w przypadku identycznego scenariusza w wykorzystaniem PEGASIS uzyskano w najgorszym przypadku 7 rund, w najlepszym 13. Natomiast LEACH umożliwiło utrzymanie pracy węzłów w 27 rundach dla najgorszego przypadku, oraz 33 rundach dla przypadku najlepszego.

W początkowym etapie pracy oczekiwanym było, że algorytm PEGASIS będący usprawnieniem protokołu LEACH, umożliwi uzyskanie dłuższego czasu maksymalnego pracy sieci. Na pierwszy rzut oka, wyniki tego nie odzwierciedlają. Dystrybucja maksymalnego czas pracy w LEACH jest dłuższa, jednakże wykresy pozwalają zauważyć pewną, że ilość węzłów aktywnych w protokole PEGASIS jest wyższa w początkowej fazie symulacji scenariusza (zwłaszcza w pierwszej połowie uzyskanych rund). Oznacza to, że węzeł główny ‘sink‘ uzyskuje większą ilość danych z większej puli węzłów. W scenariuszach w których odległości pomiędzy węzłami zwiększają się, koszt przekazywania informacji PEGASIS przy pomocy algorytmu ‘greedy‘ staje się bardzo kosztowny. Każdy kolejny węzeł odpowiedzialny za przekazanie (‘relay‘) danych, musi ponieść spory koszt energetyczny transmisji, który w końcowych rundach bardzo szybko eliminuje węzły niezdolne do przekazywania danych (ze względu na niewystarczające zasoby energetyczne).

Bibliografia

[1] I.F. Akyildiz, W. Su, Y. Sankarasubramaniam, E. Cayirci, „Wireless sensor networks: a survey", Georgia Institute of Technology, 2001.prez

[2] W. R. Heinzelman, A. Chandrakasan, H. Balakrishnan, „EnergyEfficient Communication Protocol for Wireless Microsensor Networks", MIT, 2000.

[3] J. Yick, B. Mukherejee, D. Ghosal, „Wireless sensor network survey", Elsevier Computer Networks, Vol. 52, 2008.

[4] I. F. Akyildiz, M. Can Vuran, „Wireless Sensor Networks", Georgia Institute of Technology, Wiley, ISBN 9780470036013, 2010.

[5] G. J. Pottie, W. J. Kaiser, Wireless integrated network sensors. Communications of the ACM, 43:51–58, May 2000.

[6] C.Y. Chong and S.P. Kumar, „Sensor networks: evolution, opportunities, and challenges", In Proceedings of IEEE, Vol. 91, s. 1247-1256, 2003.

[7] K. Sohrabi, J. Gao, V. Ailawadhi, G.J. Pottie, „Protocols for self-organization of a wireless sensor network" IEEE Personal Communications, Vol. 7, s. 16-27, 2000.

[8] M. Bajelan, H. Bakhshi, „An adaptive LEACH-based clustering algorithm for wireless sensor networks.", J. Commun. Eng. Vol. 2(4), s. 351–365 2013.

[9] J. Ma, S. Wang, C. Meng, i in., Journal on Wireless Communications and Networking, s. 102, 2018

[10] G. Kavitha, R.S.D. Wahidabanu, „Improved cluster head selection for efficient data aggregation in sensor networks“ Research Journal of Applied Sciences, Engineering and Technology, Vol. 7(24), s. 5135–5142, 2014.

[11] D. Jia, H. Zhu, S. Zou, P. Hu, „Dynamic cluster head selection method for wireless sensor network.“, IEEE Sens. J. Vol. 16(8), s. 2746–2754, 2016.

[12] A. Donowan, B. Kerninghan, „The Go Programming Language“, Addison-Wesley, ISBN 9780134190440, 2015.

Opinia

o pracy dyplomowej magisterskiej wykonanej przez dyplomanta
Zdolnego Studenta i Pracowitego Kolegę
Wydział Elektryczny, kierunek Informatyka, Politechnika Warszawska
Temat pracy
TYTUŁ PRACY DYPLOMOWEJ

Promotor: **dr inż. Miły Opiekun**
Ocena pracy dyplomowej: **bardzo dobry**

Treść opinii

Celem pracy dyplomowej panów dolnego Studenta i Pracowitego Kolegi było opracowanie systemu pozwalającego symulować i opartego o oprogramowanie o otwartych źródłach (ang. Open Source). Jak piszą Dyplomanci, starali się opracować system, który łatwo będzie dostosować do zmieniających się dynamicznie wymagań, będzie miał niewielkie wymagania sprzętowe i umożliwiał dalszą łatwą rozbudowę oraz dostosowanie go do potrzeb. Przedstawiona do recenzji praca składa się z krótkiego wstępu jasno i wyczerpująco opisującego oraz uzasadniającego cel pracy, trzech rozdziałów (2-4) zawierających opis istniejących podobnych rozwiązań, komponentów rozpatrywanychjako kandydaci do tworzonego systemu i wreszcie zagadnień wydajności wirtualnych rozwiązań. Piąty rozdział to opis przygotowanego przez Dyplomantów środowiska obejmujący opis konfiguracji środowiska oraz przykładowe ćwiczenia laboratoryjne. Ostatni rozdział pracy to opis możliwości dalszego rozwoju projektu. W ramach przygotowania pracy Dyplomanci zebrali i przedstawili w bardzo przejrzysty sposób duży zasób informacji, co świadczy o dobrej orientacji w nowoczesnej i ciągle intensywnie rozwijanej tematyce stanowiącej zakres pracy i o umiejętności przejrzystego przedstawienia tych wyników. Praca zawiera dwa dodatki, z których pierwszy obejmuje wyniki eksperymentów i badań nad wydajnością, a drugi to źródła skryptów budujących środowisko.

Dyplomanci dość dobrze zrealizowali postawione przed nimi zadanie, wykazali się więc umiejętnością zastosowania w praktyce wiedzy przedstawionej w rozdziałach 2-4. Uważam, że cele postawione w założeniach pracy zostały pomyślnie zrealizowane. Proponuję ocenę bardzo dobrą (5).

(data, podpis)

Recenzja

pracy dyplomowej magisterskiej wykonanej przez dyplomanta
Zdolnego Studenta i Pracowitego Kolegę
Wydział Elektryczny, kierunek Informatyka, Politechnika Warszawska
Temat pracy
TYTUŁ PRACY DYPLOMOWEJ

Recenzent: **prof. nzw. dr hab. inż. Jan Surowy**
Ocena pracy dyplomowej: **bardzo dobry**

Treść recenzji

Celem pracy dyplomowej panów dolnego Studenta i Pracowitego Kolegi było opracowanie systemu pozwalającego symulować i opartego o oprogramowanie o otwartych źródłach (ang. Open Source). Jak piszą Dyplomanci, starali się opracować system, który łatwo będzie dostosować do zmieniających się dynamicznie wymagań, będzie miał niewielkie wymagania sprzętowe i umożliwiał dalszą łatwą rozbudowę oraz dostosowanie go do potrzeb. Przedstawiona do recenzji praca składa się z krótkiego wstępu jasno i wyczerpująco opisującego oraz uzasadniającego cel pracy, trzech rozdziałów (2-4) zawierających bardzo solidny i przejrzysty opis: istniejących podobnych rozwiązań (rozdz. 2), komponentów rozpatrywanychjako kandydaci do tworzonego systemu (rozdz. 3) i wreszcie zagadnień wydajności wirtualnych rozwiązań, zwłaszcza w kontekście współpracy kilku elementów sieci (rozdział 4). Piąty rozdział to opis przygotowanego przez Dyplomantów środowiska obejmujący opis konfiguracji środowiska oraz przykładowe ćwiczenia laboratoryjne (5 ćwiczeń). Ostatni, szósty rozdział pracy to krótkie zakończenie, które wylicza także możliwości dalszego rozwoju projektu. W ramach przygotowania pracy Dyplomanci zebrali i przedstawili w bardzo przejrzysty sposób duży zasób informacji o narzędziach, Rozdziały 2, 3 i 4 świadczą o dobrej orientacji w nowoczesnej i ciągle intensywnie rozwijanej tematyce stanowiącej zakres pracy i o umiejętności syntetycznego, przejrzystego przedstawienia tych wyników. Drobne mankamenty tej części pracy to zbyt skrótowe omawianie niektórych zagadnień technicznych, zakładające dużą początkową wiedzę czytelnika i dość niestaranne podejście do powołań na źródła. Utrudnia to w pewnym stopniu czytanie pracy i zmniejsza jej wartość dydaktyczną (a ta zdaje się być jednym z celów Autorów), ale jest zrekompensowane zawartością merytoryczną. Praca zawiera dwa dodatki, z których pierwszy obejmuje wyniki eksperymentów i badań nad wydajnością, a drugi to źródła skryptów budujących środowisko. Praca zawiera niestety dość dużą liczbę drobnych błędów redakcyjnych, ale nie wpływają one w sposób istotny na na jej czytelność i wartość. W całej pracy przewijają się samodzielne, zdecydowane wnioski

Autorów, które są wynikiem własnych i oryginalnych badań. Rozdział 5 i dodatki pracy przekonują mnie, że Dyplomanci dość dobrze zrealizowali postawione przed nimi zadanie. Pozwala to stwierdzić, że wykazali się więc także umiejętnością zastosowania w praktyce wiedzy przedstawionej w rozdziałach 2-4. Kończący pracę rozdział szósty świadczy o dużym (ale moim zdaniem uzasadnionym) poczuciu własnej wartości i jest świadectwem własnego, oryginalnego spojrzenia na tematykę przedstawioną w pracy dyplomowej. Uważam, że cele postawione w założeniach pracy zostały pomyślnie zrealizowane. Proponuję ocenę bardzo dobrą (5).

(data, podpis)