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 [16]. Pierwsze implementacje bezprzewodowych czujników wykorzystano w celach zbrojeniowych (np. system wykrywania jednostek wojskowych i snajperów Boomerang sniper detection system) [1]. Wzrost popularności rozwiązań bezprzewodowych powiększyła gamę możliwości oferowanych przez bezprzewodowe czujniki, które dzisiaj znajdują zastosowanie w nieskończonej liczbie aplikacji [8]. Branża inżynierii środowiska umożliwiła zbudowanie wielu efektywnych systemów umożliwiających pomiar i kolekcjonowanie danych w miejscach wcześniej niedostępnych ze względu na surowe warunki atmosferyczne. System *Volcano Monitoring* zbudowany został na potrzeby monitorowania aktywności wulkanicznej w Ekwadorze w lata 2004-2005 [4]. Korzyści naukowe płynące z informacji uzyskanych w ramach tego projektu zainicjowały dalsze modernizacje i wdrożenia w systemie, pozwalające na monitorowanie większej liczby wulkanów na terenie kraju. System *ZebraNet* pozwala na kolekcjonowanie aktywności oraz lokalizacji zwierząt w terenie. Zebrane dane pozwalają na zaobserwowanie wcześniej nieznanych schematów migracji zebr oraz ich interakcji z innymi zwierzętami [**?**]. Występuje wiele innych systemów, które odpowiedzialne są za monitorowania poziomu wody, poziomu zanieczyszczenia powietrza, itp. [1] [12] Inne sektory przemysłu również wykorzystały korzyści płynące z aplikacji WSN. Współczesne budownictwo jest pełne rozwiązań domów inteligentnych w których mam możliwość pomiar temperatury i wilgoci, poboru mocy, zdalnego odczytu wartości liczników gazu i prądu [7]. Przyszłość technologii bezprzewodowej sieci czujników zdecydowanie dostarczy wiele nowych rozwiązań, zarówno w nowych branżach jak i tych aktualnie wykorzystujących podobne rozwiązania [3].

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.

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 uwidocznił 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). Projekt chroniony jest licencją publiczną typu *Apache License 2.0*, która pozwala na wykorzystywanie aplikacji w celach komercyjnych, z możliwością dystrybucji, modyfikacji oprogramowania. Licencja ta nie stanowi gwarancji poprawności działania oprogramowania. Oprogramowanie dostępne na GitHub umożliwia weryfikację uzyskanych wyników opisanych w pracy dyplomowej, jak również analizę stworzonego oprogramowania.

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 [27]. 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.

```
1 package main
2
3 import (
4   "fmt"
5
6   "github.com/golang/example/stringutil"
7 )
8
9 type Label struct {
```

```
10   Text string
11   X, Y int
12 }
13
14 func (l *Label) ReverseText() {
15   l.Text = stringutil.Reverse(l.Text)
16 }
17
18 func main() {
19   var x = 100
20   fmt.Printf("<%T>: %v", x, x)
21   // Expected output: <int>: 100
22   hello := Label{
23     Text: "Hello",
24     X:    100,
25     Y:    200,
26   }
27
28   fmt.Println(hello.Text)
29   // Expected output: hello
30   hello.ReverseText()
31   fmt.Println(hello.Text)
32   // Expected output: olleh
33 }
```

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

Cechy języka Go i Python Go jest językiem wspomagającym tworzenie aplikacji wielowątkowych. Zaliczany jest do języków statycznych, co pozwala wyeliminować błędy typu *runtime* wynikające z typu zmiennej. Go jest językiem kompilowanym, co pozwala na szybsze uruchamianie i wykonanie poleceń oprogramowania oraz wykorzystuje mniej pamięci (np. w Go zmienna typu int32 wymaga 4 bajty pamięci, w Python 24 bajty). Python od samego początku umożliwiał *runtime reflection*, które jest już dostępne w Golang przy wykorzystaniu biblioteki *reflect*). W pierwszych latach funkcjonowania Go, język Python posiada zdecydowanie większą bazę publicznych bibliotek.

Cechy języka Go i C++ Go posiada system zarządzania pamięcią (*garbage collector*) i jest językiem wspomagającym tworzenie aplikacji wielowątkowych. Go nie wymaga tworzenia plików typu header. Go jest języ-

kiem obiektowym, niemniej zdolność dziedziczenia (*inheritance*) została zastąpiona osadzaniem (*embedding*). Dodatkowo Go posiada możliwość użycia (zaimportowania) dowolnej biblioteki C, C++. W przeciwieństwie, C++ pozwala osiągnąć krótszy czas wykonania poleceń oprogramowania, jak również tworzenie kodu niezależnie od typu zmiennej (*generics*). Możliwym jest, że Go w wersji 2.0 będzie wyposażony w zmienne typu *generics*. Istnieją również aplikacje w których brak systemu zarządzania pamięcią (*garbage collector*) w C++, umożliwia większą kontrolę nad zasobami pamięci (np. w mikro kontrolerach).

Unikatowe cechy Go Go posiada unikatową cechę samodokumentowania kodu. Odgórnie określony format komentarzy umożliwia automatyczne tworzenie przejrzystej dokumentacji. Dodatkowo, produktem kompilacji jest plik egzekucyjny posiadający wszystkie niezbędne zależności, a zarządzanie pakietami pozwala na importowanie rozwiązań bezpośrednio z GitHub (lub innych serwisów zewnętrznych). Na etapie pisania kodu źródłowego istnieje cecha dynamicznej alokacji typu zmiennej statycznej. Go wyposażony jest w system monitorujący zdrowie wątków i współbieżnych procesów, oraz natywnę metoda testowania funkcjonalności i bibliotek w sposób równoległy. Dodatkowo pełny zestaw wbudowanych narzędzi pozwalających na testowanie wydajności kodu (*benchmarking*), formatowania składni kodu zgodnie ze standardem (gofmt), oraz wiele innych komponentów.

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.

```
1 // example.proto
2 syntax = "proto3";
3
4 // Citizen represents a single citizen of Poland.
5 message Citizen {
6   // The name of a citizen.
7   string name = 1;
```

```
8   // The surname of a citizen.
9   string surname = 2;
10  // (required) Unique Polish national identification number.
11  PESEL pesel = 3;
12 }
13
14 // PESEL represents Polish Universal Electronic System for
15 // Registration of the Population.
16 message PESEL {
17  // (required) Unique Polish national identification number.
18  uint64 number = 1;
19  bool active = 2;
20 }
```

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 cechy ułatwiły proces projektowania, implementacji i zarządzania oprogramowaniem stworzonym na potrzeby pracy dyplomowej.

2.3.1 System kontroli wersji git

Git to rozproszony system kontroli wersji stworzony jako wolne oprogramowanie (open source) [30]. 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 [31]. 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 [29]. 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 (dependency) 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. Materiał przedstawiony w tym rozdziale pozwala na lepsze zrozumienie istniejących zjawisk fizycznych oraz 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 [7]. 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) [3].

Rozwój technologii, malejący koszt elektroniki oraz dostępność produktów bezprzewodowej sieci czujnikowej, umożliwia tworzenie dedykowanych aplikacji [11]. Na rynku dostępnych jest wiele urządzeń oraz rozwiązań, a proces wyboru uzależniony jest przede wszystkim od wymogów projektu oraz budżetu [3] [6]. 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 [12]. 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 (*routing*) [9], których zadaniem jest zoptymalizowanie energetyczne procesu komunikacji [2] [10], podwyższenie niezawodności systemu oraz umożliwienie zautomatyzowanej organizacji topologii sieci [15]. Korzyści płynące z wykorzystania protokołów trasowania są niemieżalne bez uprzedniego przedstawienia 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 [2]:

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

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 [1] [9].

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

Poniżej 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, będącym elementem pracy dyplomowej. 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 [9] [7]. 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 przede wszystkim od odległości między dwoma węzłami. Występują również koszty np. koszt energetyczny czasem pracy procesora odpowiedzialnego za przetwarzanie danych. 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 trasowania (*routing*) bezprzewodowej sieci czujnikowej [1]. 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*) [26]. 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 [19] [8].

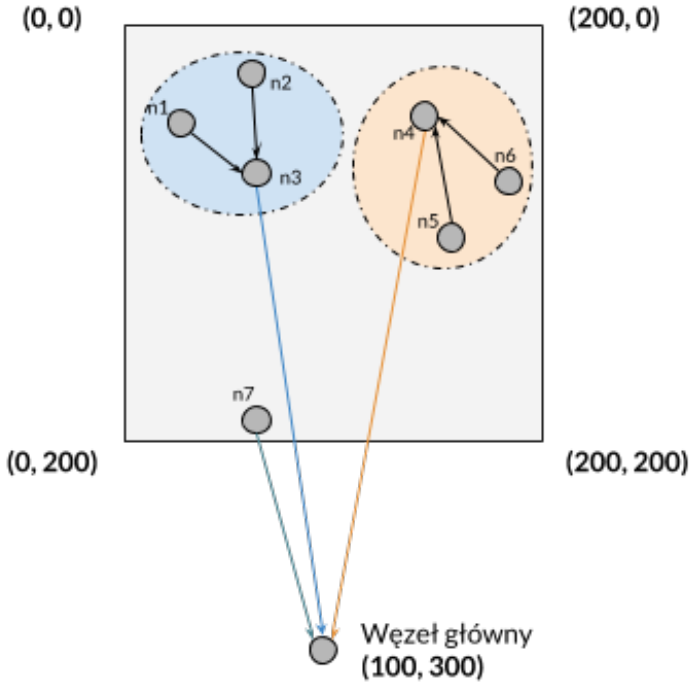
W momencie ustalenia roli przez węzły, każdy z węzłów informuje pozostałe węzły uczestniczące w sieci o swojej roli w rundzie.

Kolejnym etapem fazy konfiguracji jest wyznaczenie klastra do którego zostaną przyłączone węzły standardowe. Każdy z węzłów standardowych wybiera jeden węzeł typu *cluster head* (CH). 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 sieć 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 [21] [22].

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



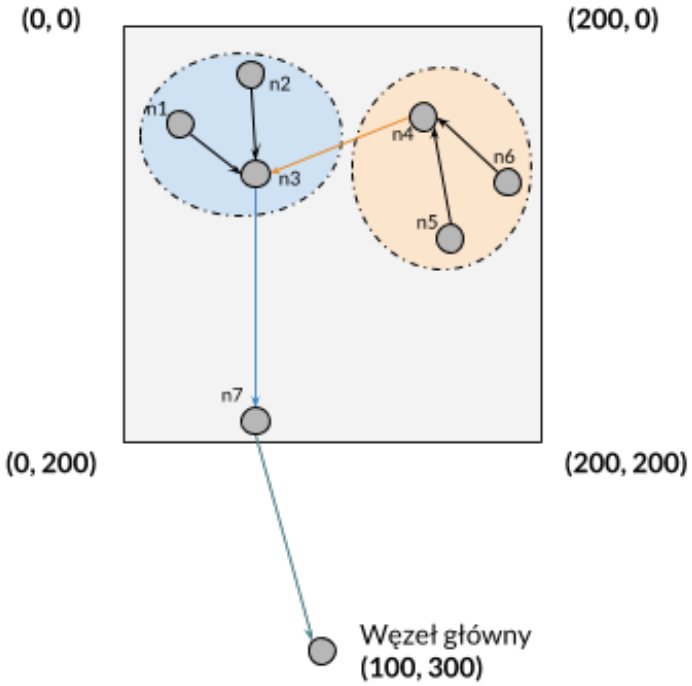
3.3.3 PEGASIS

PEGASIS (*Power-Efficient GAthering in Sensor Information Systems*) jest algorytmem wykorzystywanym w protokołach trasowania (*routing*) bezprzewodowej sieci czujnikowej [20]. 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* [32].

Efektem niekorzystnym w protokole PEGASIS jest wydłużenie czasu każdej z rund, wynikające z sekwencyjnej wymiany danych w łańcuchu [20]. 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.



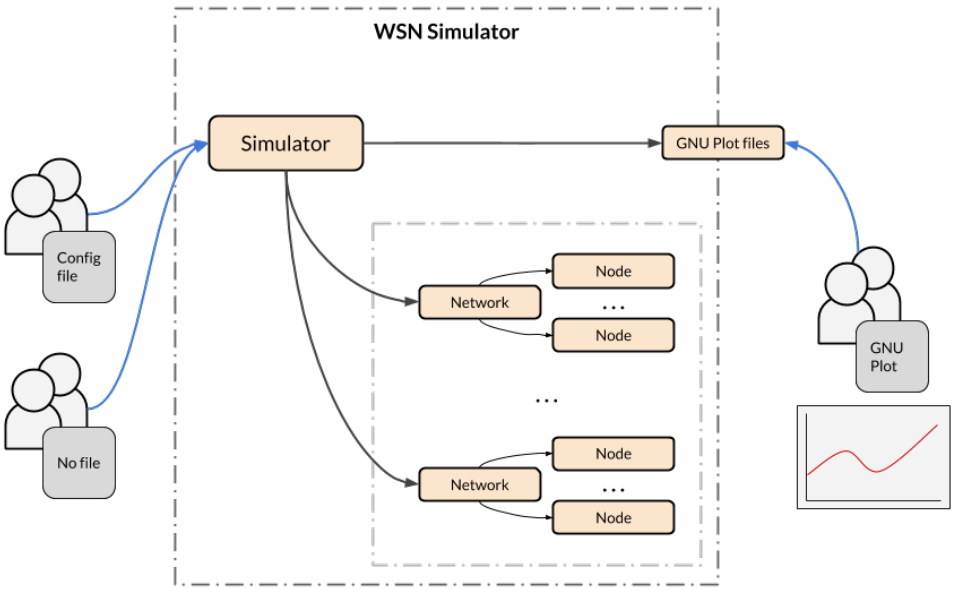
Rozdział 4

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 [m] x 200 [m]. 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 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*.

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.

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;
}
```

Fragment pliku konfiguracyjnego dla konfiguracji lokalizacji węzła:

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

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;
}
```

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;
}
```

Przykładowa konfiguracja

Poniższa lista przedstawia kompletny zbiór elementów losowego pliku konfiguracyjnego:

- Protokół: DIRECT, LEACH, PEGASIS
- Maksymalna liczba 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 [m]
- Lokalizacja węzła głównego na osi Y: 300 [m]
- Liczba węzłów: 200
- Energia węzłów: 1 [J]
- Lokalizacja węzłów na osi X: [0, 200] [m]
- Lokalizacja węzłów na osi Y: [0, 200] [m]

4.1.2 Moduł główny (Core)

Moduł główny posiada najważniejsze cechy 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.

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
```

Struktura węzła *Node* posiada 3 zmienne publiczne. Zmienna (*Conf*) przechowuje konfigurację węzła w formacie protocol buffer (szczegółowe informacje dostępne w podrozdziale *Konfiguracja węzła*). Zmienna (*Ready*) definiuje stan gotowości węzła do pracy. Wartość *true* oznacza, że węzeł jest gotowy do nadawania i odbierania informacji. (*Energy*) przechowuje aktualny stan energetyczny węzła. W przypadku wyczerpania energii (wartość mniejsza lub równa zero), zmienna *Ready* zostaje ustawiona na *false*.

Dodatkowo, struktura *Node* składa się ze zmiennych prywatnych przechowujących wiele istotnych stanów pracy węzła. Następny skok (*nextHop*) przechowuje adres pamięci do zmiennej węzła, będącego odbiorcą informacji nadawanych przez węzeł. Wartością domyślna w momencie rozpoczęcia symulacji jest adres głównego odbiornika danych (*sink*). Kolejka nadawania (*transmitQueue*) i kolejka odbioru (*receiveQueue*) przechowują informację o liczbie bajtów gotowych do przekazania do następnego węzła na końcu rundy oraz o liczbie bajtów odebranych przez węzeł na początku rundy. Dane nadane (*dataSent*) i dane odebrane (*dataReceived*) przechowuje sumę bajtów nadanych i odebranych przez węzeł.

Do struktury węzła przynależą również 3 funkcje publiczne. Funkcja *Transmit* pozwala na transmisję wiadomości do węzła docelowego, natomiast funk-

cja *Receive* pozwala na odbieranie wiadomości przez węzeł Funkcja *Info* generuje ciąg znaków w podstawowymi informacjami na temat węzła. Prywatne funkcje *distance* oraz *consume* odpowiadają za wyznaczanie odległości pomiędzy dwoma węzłami oraz obciążeniem węzła kosztem energetycznym za wykonaną pracę.

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.

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()
```

Struktura sieci *Network* posiada 12 zmiennych publicznych. Protokół (*Protocol*) przechowuje obiekt definiujący protokół komunikacji pomiędzy węzłami. Stacja bazowa (*BaseStation*) przechowuje adres pamięci do obiektu

węzła głównego (*sink*). Węzły (*Nodes*) 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*) przechowuje numer aktualnej rundy symulacji oraz maksymalna ilość rund (*MaxRounds*) definiuje maksymalną ilość rund symulacji. W przypadku osiągnięcia wartości *Round* równej *MaxRounds*, symulacja zostanie przerwana i wyniki zostają zwrócone do użytkownika. Długość wiadomości (*MsgLength*) przechowuje informację o całkowitej wielkości wiadomości (mierzonej w bajtach) jaka generowana i nadawana jest podczas rundy przez każdy z węzłów. W skład tej wartości wchodzą dane pomiarowe i dodatkowy nakład informacji powstały w wyniku enkapsulacji. Zmienne *GNUPlotNodes, GNUPlotTotalEnergy* przechowujące parametry rund wykorzystywane do generowania grafów przy użyciu narzędzia GNUPlot. Następnie *PlotTotalEnergy, PlotNodes, NodesAlivePoints, NodesEnergyPoints* przechowują parametry rund wykorzystywane do generowania grafów przy użyciu biblioteki plotter.

Dodatkowo, do struktura sieci *Network* przynależy 5 funkcji publicznych. Funkcja *AddNode* pozwala na dodanie węzła do obiektu sieci, oraz *CheckNodes* pozwala na sprawdzenie liczby węzłów w sieci. Funkcja *PopulateEnergyPoints* oraz *PopulateNodesAlivePoints* odpowiedzialna jest za sprawdzanie i dodawanie nowych danych pomiarowych dotyczących stanów energetycznych węzłów. Ostatnią i najważniejszą funkcją jest *Simulate*, która dokonuje rozpoczęcia symulacji w oparciu o wcześniej dodane węzły i protokół.

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:

```
1 const (
2   // Energy values measured in [J/byte].
3   E_ELEC = 40e-9
4   E_RX   = 4e-9
5   E_MP   = 0.0104e-12
6   E_FS   = 80e-12
7 )
```

Protokół (Protocol)

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

Fragment interfejsu protokołu oraz definicje funkcji przynależące do interfejsu:

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

Interfejs *Protocol* wymaga od struktury posiadania trzech funkcji o powyższych sygnaturach. Funkcja *Setup* odpowiedzialna jest za konfigurację węzłów w sieci zgodnie z zaimplementowanym protokołem. Dodatkowo, dwie funkcje pomocnicze *SetNodes* oraz *SetCluster*. Pierwsza z nich pozwalają na zdefiniowanie całkowitej liczby węzłów biorących udział w symulacji. Druga natomiast dokonuje przekonwertowania wartości parametru 'p' na maksymalną liczbę węzłów, które mogą pełnić rolę węzła typu *cluster head*.

W autorskim oprogramowaniu symulatora dostępne są trzy protokoły komunikacji. Komunikacja bezpośrednia (*Direct Communication*), w której wszystkie węzły w sieci komunikują się bezpośrednio z węzłem głównym (*sink*). Komunikacja z wykorzystaniem protokołu LEACH oraz PEGASIS, w których wszystkie węzły w sieci komunikują się zgodnie z topologią ustaloną w procesie konfiguracji LEACH, bądź 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 wyników symulacji. Dane wyjściowe mogą zostać podane dalszej analizie przy wykorzystaniu zewnętrznych narzędzi (np. gnuplot).

Struktura symulatora oraz definicje funkcji przynależące do struktury:

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

```

```
6   plotTotalEnergy *plot.Plot // An amount of total energy in the
        network per round.
7   plotNodes      *plot.Plot // A number of alive nodes in the
        network per round.
8 }
9
10 func Create() (*Simulator, error)
11
12 func (s *Simulator) AddScenario(name string, conf *config.Config)
       error
13 func (s *Simulator) Run() error
14 func (s *Simulator) ExportPlots(filepath string) error
15 func (s *Simulator) ExportGNUPlots(filepath string) error
16
17 func createAndPopulateFile(filepath string, data []string) error
18 func (s *Simulator) create(name string, conf *config.Config) error
19 func createPlot(title, x, y string) (*plot.Plot, error)
20 func (s *Simulator) plotter() error
```

Symulator posiada trzy zmienne prywatne typu dziennik. Przestrzeń nazw (*namespace*) przechowuje nazwy symulacji, które w jednoznaczny sposób identyfikują sieć i konfigurację. Przestrzeń konfiguracji (*config*) przechowuje obiekt konfiguracji (*Config*) dla każdej symulacji. Kluczem dziennika jest unikatowa nazwa symulacji. Przestrzeń sieci (*network*) przechowuje obiekt sieci (*Network*) dla każdej symulacji. Również w tym przypadku kluczem dziennika jest nazwa symulacji.

Dodatkowo struktura Symulatora (*Simulator*) posiada 4 funkcje publiczne. Funkcja *AddScenario* umożliwia dodanie scenariusza do obiektu symulatora. Wymaganym jest podanie nazwy symulacji oraz pliku konfiguracyjnego. Możliwym jest wykorzystanie tego samego pliku konfiguracyjnego w wielu scenariuszach, wymaganym jest jednak wygenerowanie unikatowej nazwy symulacji. W przeciwnym wypadku symulator zwróci błąd informujący o niedozwolonej duplikacji nazwy symulacji. W momencie umieszczenia przynajmniej jednej konfiguracji, użytkownik może wywołać funkcję *Run*, która odpowiedzialna jest za uruchomienie symulacji scenariuszy. Efektem pomyślnego zakończenia symulacji jest uzyskanie wyników pracy sieci w poszczególnych rundach. Funkcje *ExportPlots* oraz *ExportGNUPlots* pozwalają na przekonwertowanie danych do grafiki lub pliku w formacie kompatybilnym z gnuplot.

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 konfigu-

racyjny dla każdego zaimplementowanego protokołu.

Rysunek 4.1: Przykładowe rozstawienie węzłów w sieci wygenerowanej dynamicznie.

4.2.3 Uruchamianie scenariuszy

Oprogramowanie symulatora posiada możliwość wprowadzenia więcej niż jednego plik konfiguracyjnego równocześnie. Należy pamiętać, że chociaż górna granica ilości plików nie istnieje, czas symulacji ulegnie wydłużeniu i w przypadku zaawansowanych scenariuszy czas może wynosić nawet i kilkanaście sekund.

Przykład uruchamiania projektu z dwoma plikami konfiguracyjnymi, które uruchomione zostanę pięc razy:

```
$ /go/src/github.com/keadwen/msc_project/msc_project \
    --repeat_config=5
    --config_file=example1.proto,example2.proto
```

Przykład uruchamiania projektu bez flagi pliku konfiguracyjnego i flagi powtórzenia (która domyślnie wynosi 1):

```
$ /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 liczby 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. Dlatego w momencie poprawnego zakończenia symulacji, folder plotdata powinien posiadać zestaw plików w formacie GNUPlot. 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*. Przed przystąpieniem do implementacji nowego protokołu, zalecane jest wykorzystanie szablonu pliku komunikacji bezpośredniej, gdyż posiada on podstawowe zasady tworzenia nowego protokołu.

Po wykonaniu wszystkich 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 w nowej wersji oprogramowania.

Rozdział 5

Opracowanie wyników eksperymentów

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

5.1 Dane wejściowe

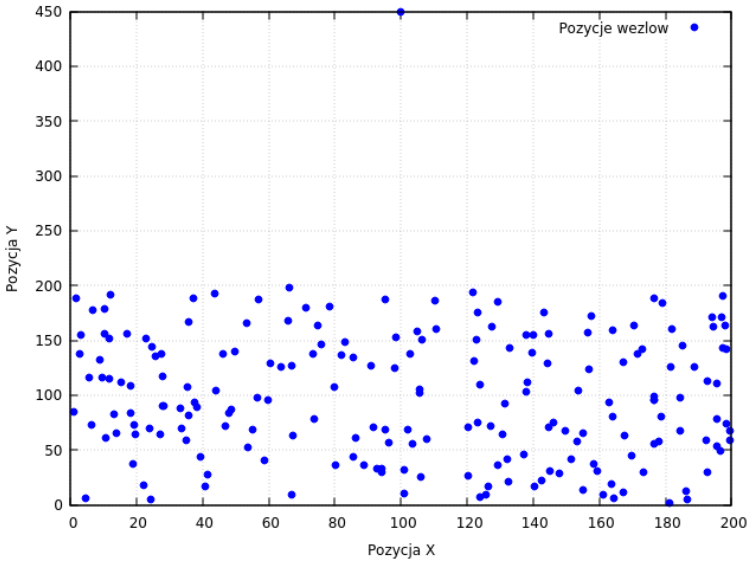
Oprogramowanie posiada implementacje trzech algorytmów: komunikację bezpośrednią, LEACH oraz PEGASIS (dokładniejszy opis działania protokołów został przedstawiony w poprzednich rozdziałach). W celu porównania efektywności energetycznej pomiędzy trzema rozwiązaniami, niezbędne jest precyzyjne zdefiniowanie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

- Wielkość wiadomości - parametr ten definiuje wielkość wiadomości wysyłanej pomiędzy węzłami (mierzone w Bajtach).
- Liczba 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 (mierzony w metrach).
- 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 (domyślnie 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] [m], Y: [0, 200] [m]. 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 o przestrzeni 200 [m] x 200 [m].

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.

Folder testdatatest_1 zawiera 12 plików konfiguracyjnych, w których parametry węzłów i sieci są takie same jak w konfiguracji domyślnej. W nazwie pliku występuje ciąg znaków (np. *m32*), który oznacza wielkość wiadomości wykorzystanej w scenariuszu.

```
~/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
```

Komenda uruchamiająca oprogramowanie symulatora dla testu z wykorzystaniem 12 plików konfiguracyjnych:

```
~/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,\
```

```
12 testdata/test_1/pegasis200_m128.pbtxt,\
13 testdata/test_1/pegasis200_m256.pbtxt
```

Poniższe tabele przedstawiają wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Protokół: DIRECT, Wielkość wiadomości: 32 [B]

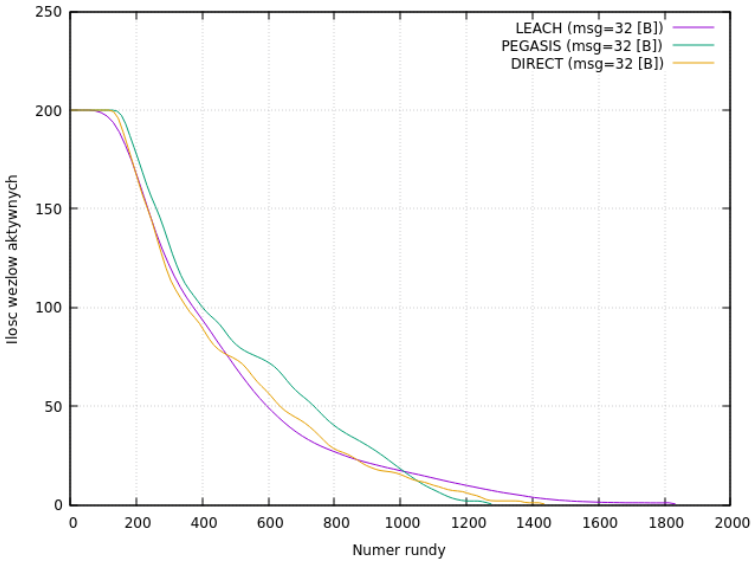
Kwantyl	-
Wartość	1434

Protokół: LEACH, Wielkość wiadomości: 32 [B]

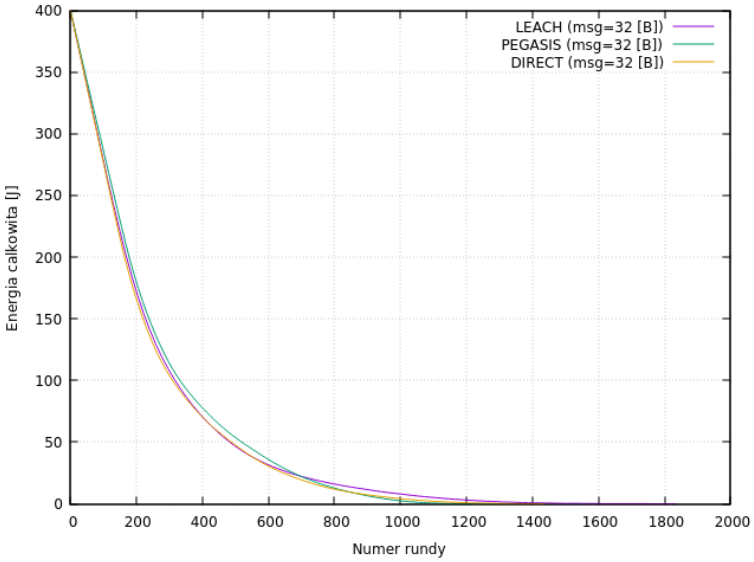
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	1431	1499	1518	1571	1635	1697	1765	1783	1805	1831

Protokół: PEGASIS, Wielkość wiadomości: 32 [B]

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	1136	1147	1152	1166	1185	1203	1224	1234	1262	1274



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



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

Maksymalna liczba rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 1434 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego *sink*). 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 poziomie -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, liczba 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.

Protokół: DIRECT, Wielkość wiadomości: 64 [B]

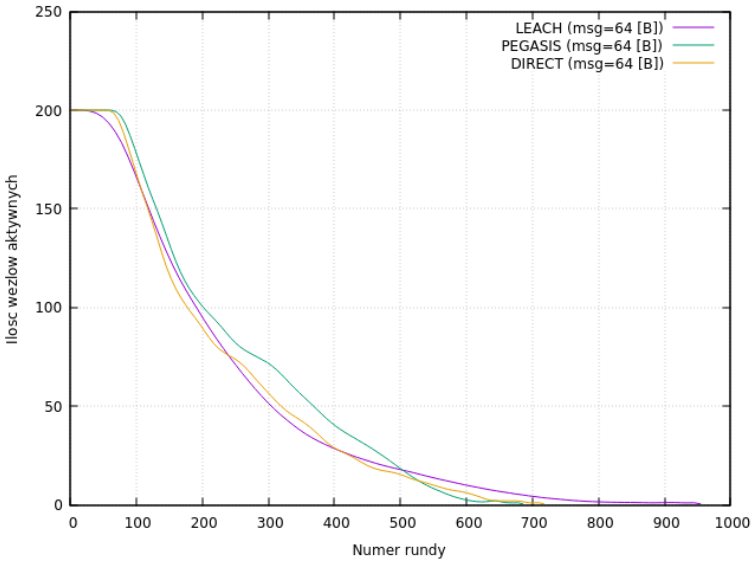
Kwantyl	-
Wartość	717

Protokół: LEACH, Wielkość wiadomości: 64 [B]

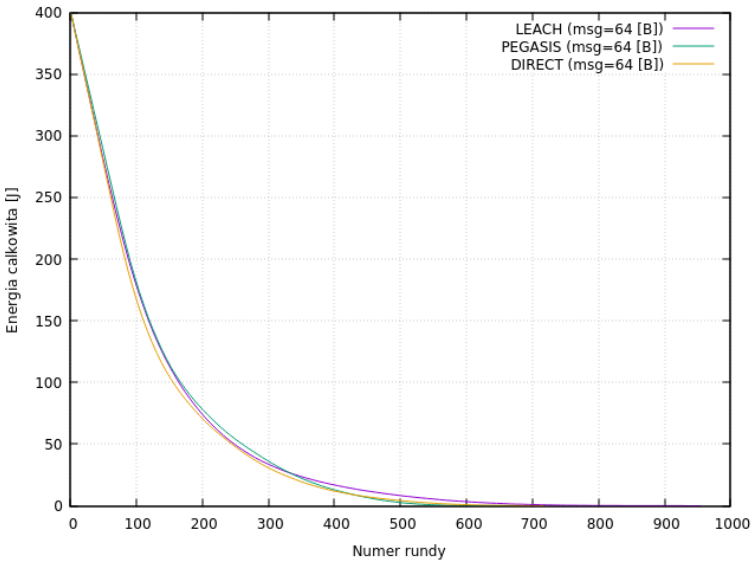
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	745	760	770	801	831	872	893	908	935	954

Protokół: PEGASIS, Wielkość wiadomości: 64 [B]

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	576	584	587	594	608	622	629	635	669	685



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



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

Maksymalna liczba rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 717 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego *sink*). 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 skró-

cenie 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, liczba 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.

Protokół: DIRECT, Wielkość wiadomości: 128 [B]

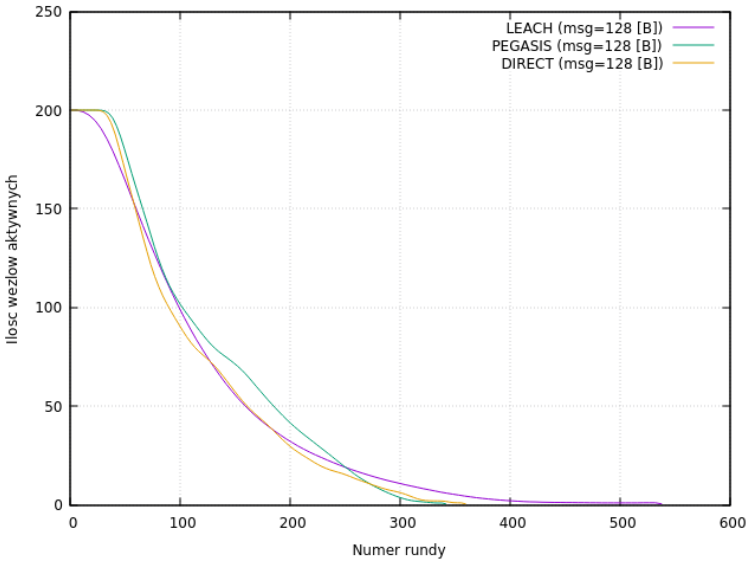
Kwantyl	-
Wartość	359

Protokół: LEACH, Wielkość wiadomości: 128 [B]

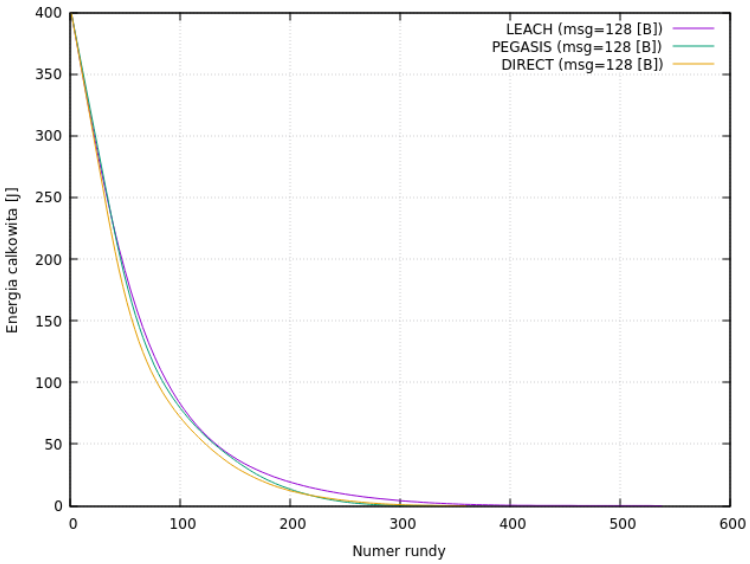
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	388	392	399	409	432	453	479	492	508	537

Protokół: PEGASIS, Wielkość wiadomości: 128 [B]

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	291	295	300	305	312	318	327	330	336	341



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



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

Maksymalna liczba 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), +20.3% dla mediany (432), +49.6% dla wartości maksymalnej (537). Dla PEGASIS uzyskano skró-

cenie 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, liczba 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.

Protokół: DIRECT, Wielkość wiadomości: 256 [B]

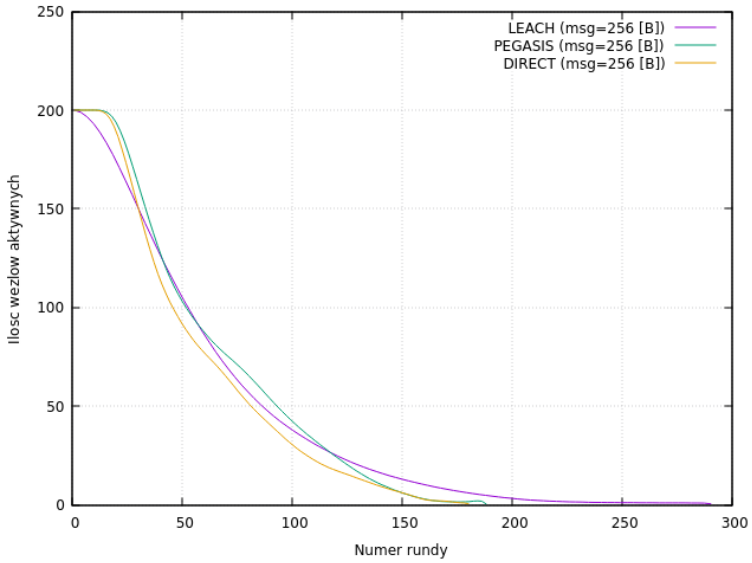
Kwantyl	-
Wartość	180

Protokół: LEACH, Wielkość wiadomości: 256 [B]

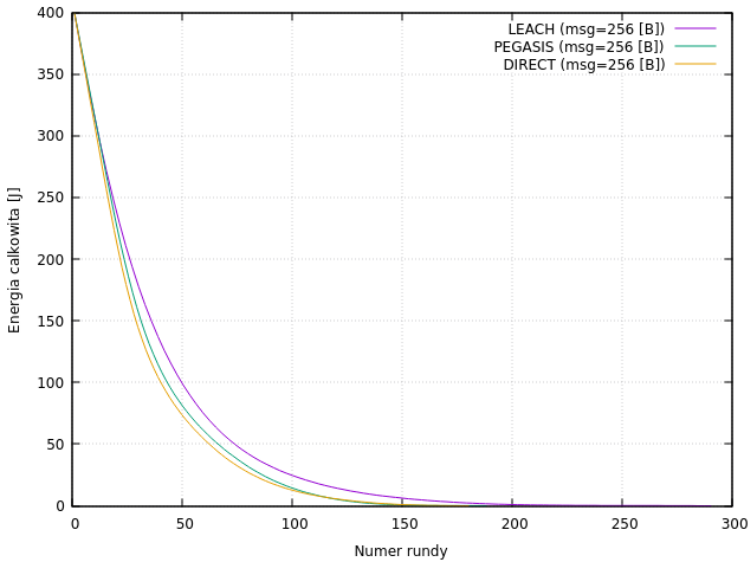
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	207	212	215	224	239	252	267	281	289	290

Protokół: PEGASIS, Wielkość wiadomości: 256 [B]

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	155	157	158	162	166	170	175	179	181	188



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



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

Maksymalna liczba rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 180 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego *sink*). 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 skró-

cenie 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, liczba 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 - Liczba węzłów w sieci

Test *Liczba węzłów w sieci* ma na celu przedstawienie zależności efektywności energetycznej dla trzech algorytmów względem liczby węzłów znajdujących się w sieci.

Folder testdatatest_2 zawiera 15 plików konfiguracyjnych, w których parametry węzłów i sieci są takie same jak w konfiguracji domyślnej. W nazwie pliku występuje liczba, która określa liczbę węzłów znajdujących się w sieci.

```
~/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
```

Komenda uruchamiająca oprogramowanie symulatora dla testu z wykorzystaniem 15 plików konfiguracyjnych:

```
~/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
```

Poniższe tabele przedstawiają wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Protokół: DIRECT, Liczba węzłów: 25

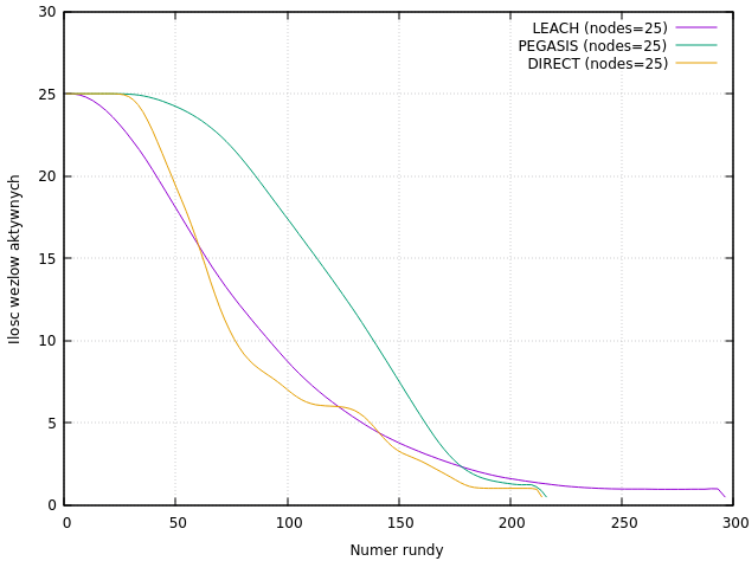
Kwantyl	-
Wartość	214

Protokół: LEACH, Liczba węzłów: 25

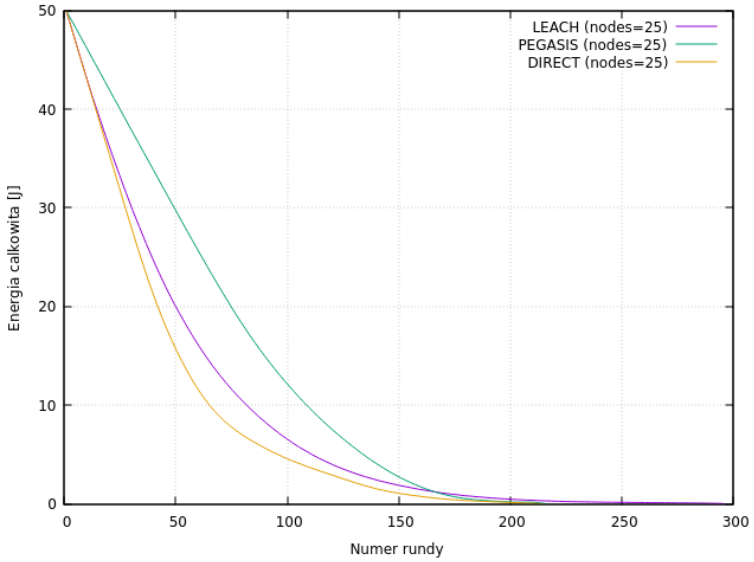
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	179	186	197	212	230	246	263	272	287	296

Protokół: PEGASIS, Liczba węzłów: 25

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	170	172	176	182	187	195	201	206	215	216



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



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

Maksymalna liczba 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 maksy-

malnej (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, liczba 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.

Protokół: DIRECT, Liczba węzłów: 50

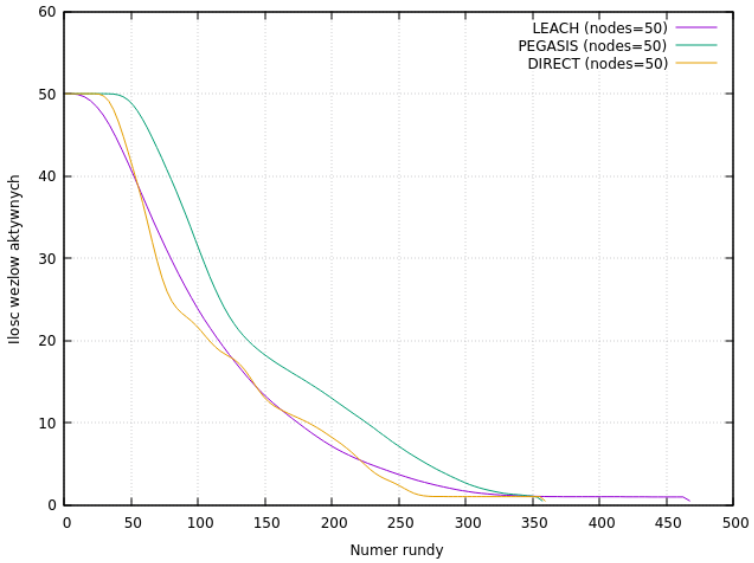
Kwantyl	-
Wartość	359

Protokół: LEACH, Liczba węzłów: 50

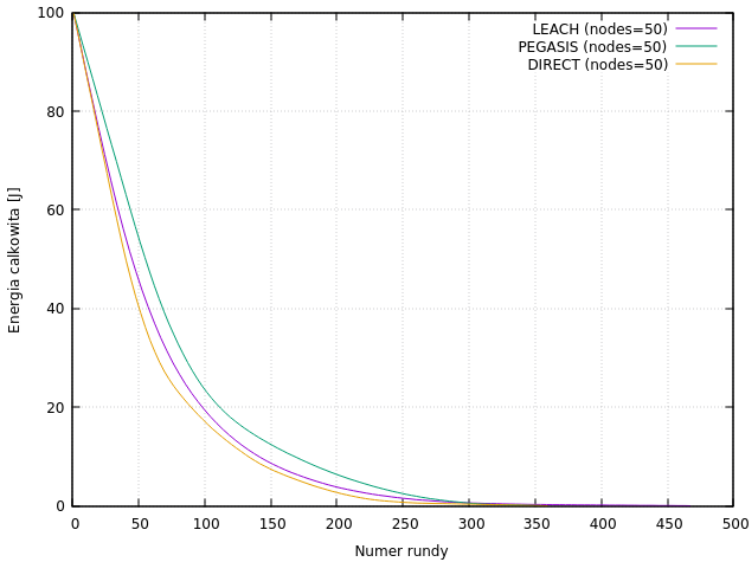
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	294	308	321	357	382	410	435	443	461	467

Protokół: PEGASIS, Liczba węzłów: 50

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	285	292	302	310	320	332	342	348	354	357



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



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

Maksymalna liczba 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 maksy-

malnej (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, liczba 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.

Protokół: DIRECT, Liczba węzłów: 100

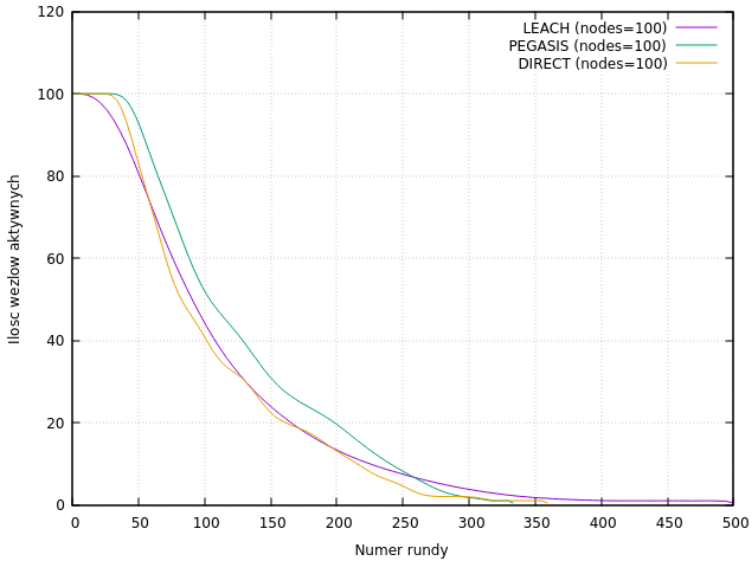
Kwantyl	-
Wartość	359

Protokół: LEACH, Liczba węzłów: 100

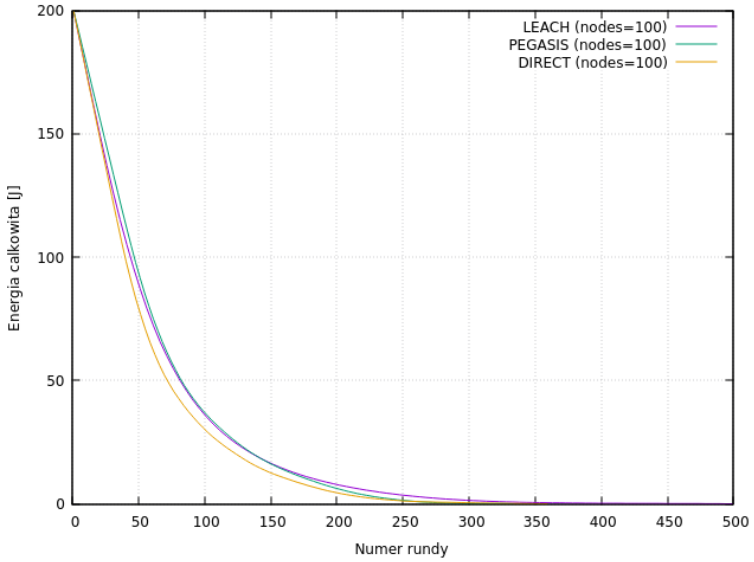
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	326	347	356	385	413	439	464	477	493	499

Protokół: PEGASIS, Liczba węzłów: 100

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	269	274	278	285	291	301	311	314	319	333



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



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

Maksymalna liczba 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, liczba 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.

Protokół: DIRECT, Liczba węzłów: 150

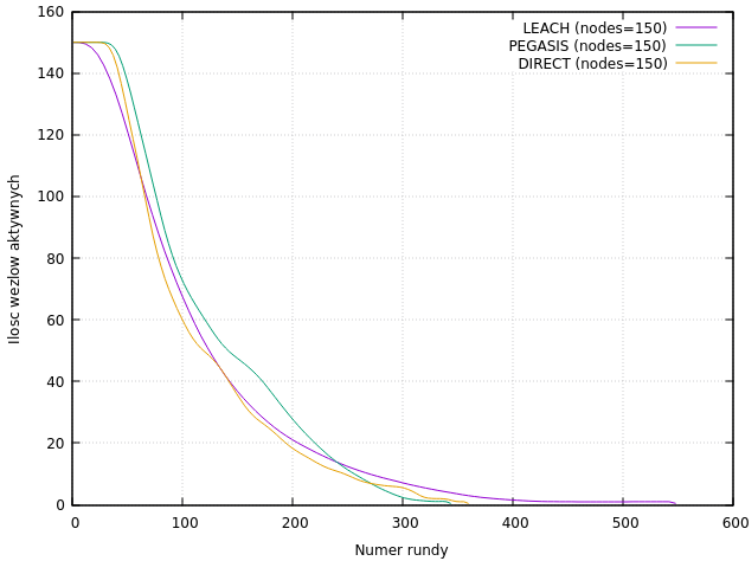
Kwantyl	-
Wartość	359

Protokół: LEACH, Liczba węzłów: 150

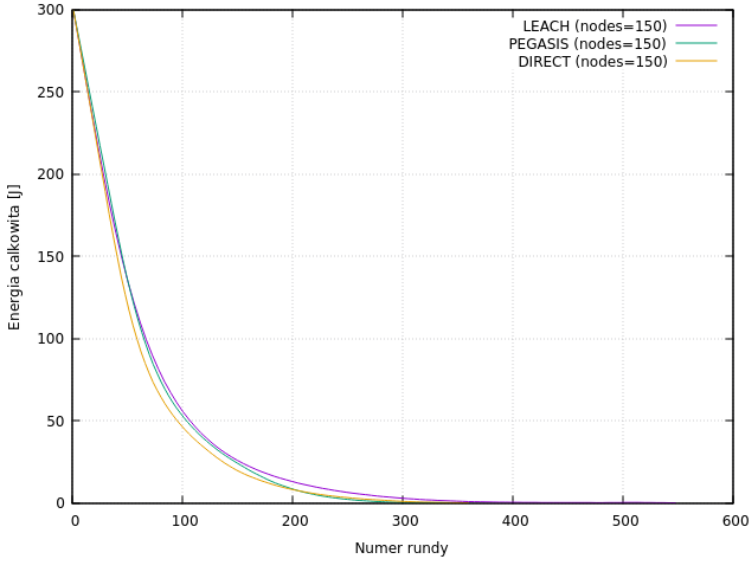
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	368	378	384	399	425	447	468	484	499	547

Protokół: PEGASIS, Liczba węzłów: 150

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	287	293	297	304	312	322	330	333	339	343



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



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

Maksymalna liczba 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 skró-

cenie 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, liczba 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.

Protokół: DIRECT, Liczba węzłów: 200

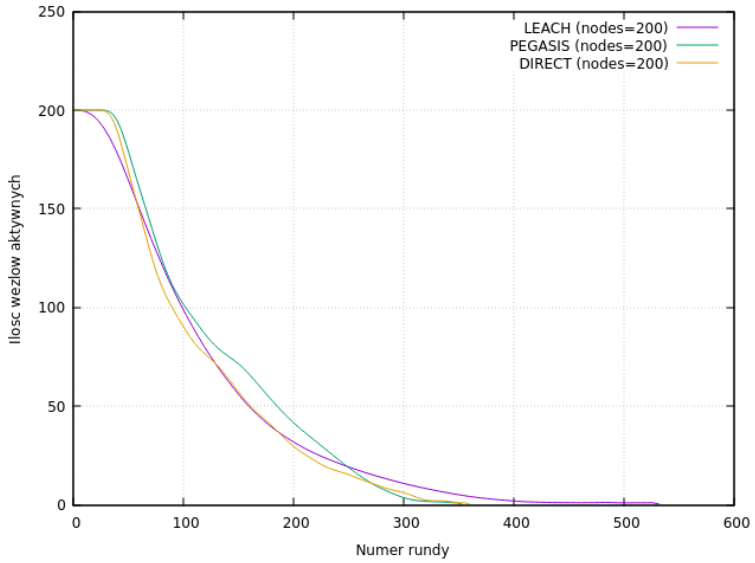
Kwantyl	-
Wartość	359

Protokół: LEACH, Liczba węzłów: 200

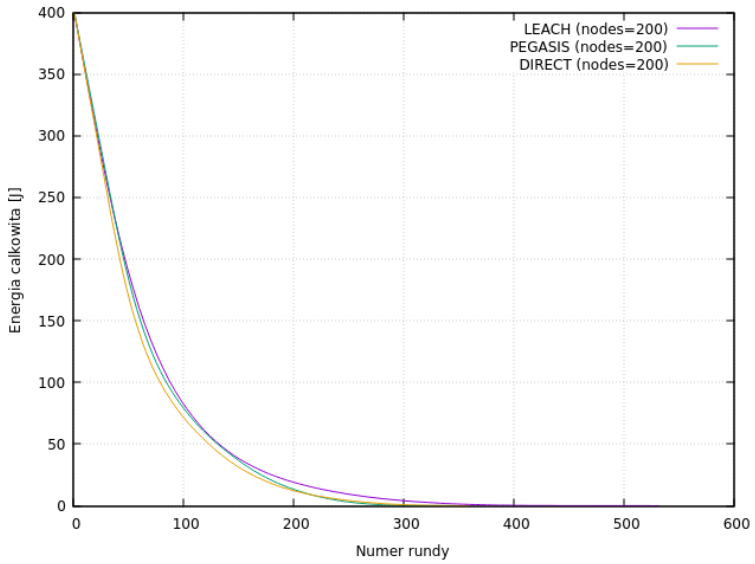
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	382	398	405	415	430	449	467	475	499	531

Protokół: PEGASIS, Liczba węzłów: 200

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	297	299	302	307	313	321	329	335	349	352



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



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

Maksymalna liczba 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 skró-

cenie 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, liczba 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.

Folder testdatatest_3 zawiera 15 plików konfiguracyjnych, w których parametry węzłów i sieci są takie same jak w konfiguracji domyślnej. W nazwie pliku występuje ciąg znaków (np. *xy100*), który oznacza wielkość sieci wykorzystanej w scenariuszu.

```
~/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
```

Komenda uruchamiająca oprogramowanie symulatora dla testu z wykorzystaniem 15 plików konfiguracyjnych:

```
~/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
```

Poniższe tabele przedstawiają wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Protokół: DIRECT, Skala sieci: 0.25

Kwantyl	-
Wartość	42976

Protokół: LEACH, Skala sieci: 0.25

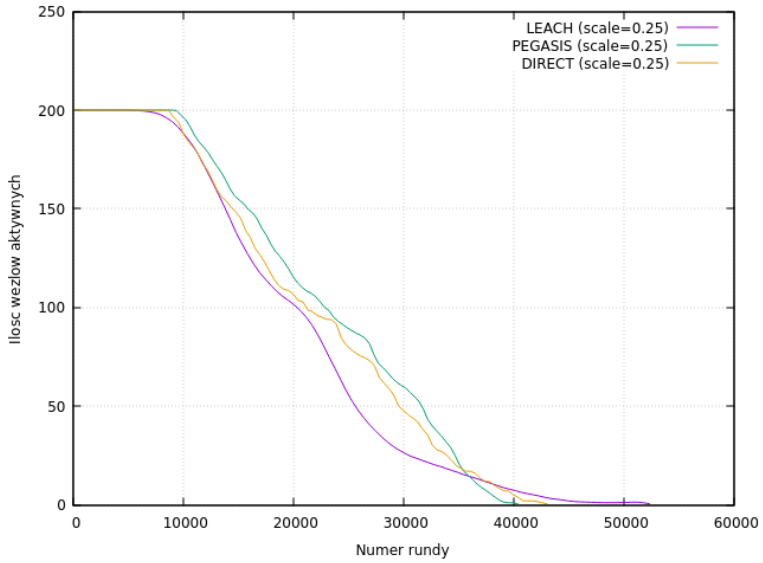
Kwantyl	1	5	10	25	50	75
Wartość	43417	44197	45130	45790	46910	48151

Kwantyl	90	95	99	100
Wartość	49310	50097	51909	52278

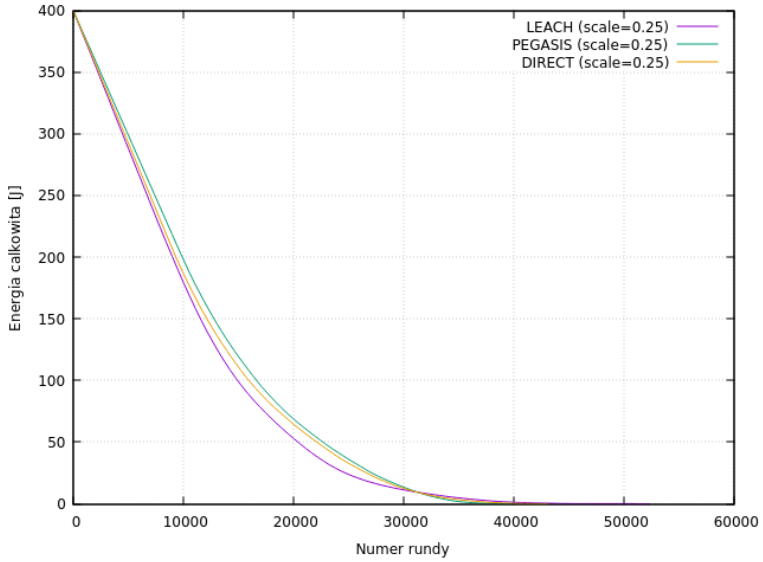
Protokół: PEGASIS, Skala sieci: 0.25

Kwantyl	1	5	10	25	50	75	90
Wartość	38521	38724	38847	39050	39279	39544	39756

Kwantyl	95	99	100
Wartość	39828	40038	40316



Rysunek 5.20: Funkcja liczby węzłów aktywnych w rundzie.



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

Maksymalna liczba 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: DIRECT, Skala sieci: 0.50

Kwantyl	-
Wartość	5657

Protokół: LEACH, Skala sieci: 0.50

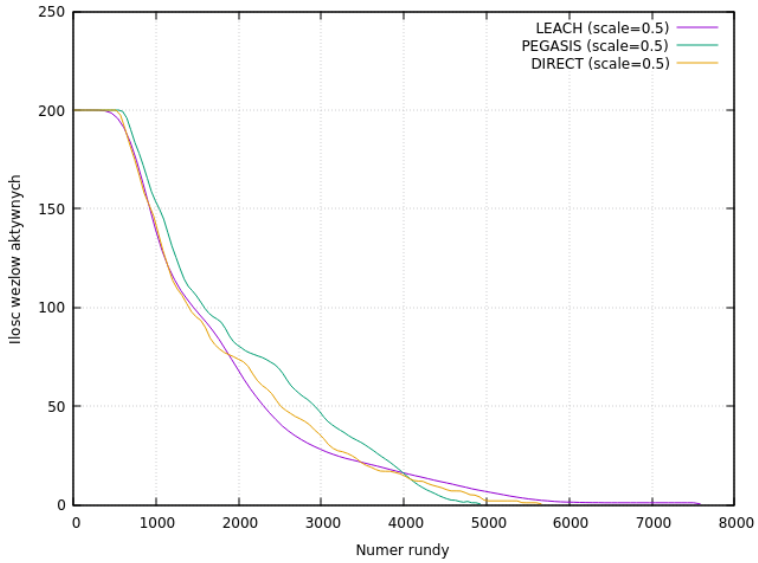
Kwantyl	1	5	10	25	50	75	90
Wartość	5598	5744	5872	6035	6293	6564	6780

Kwantyl	95	99	100
Wartość	6920	7214	7580

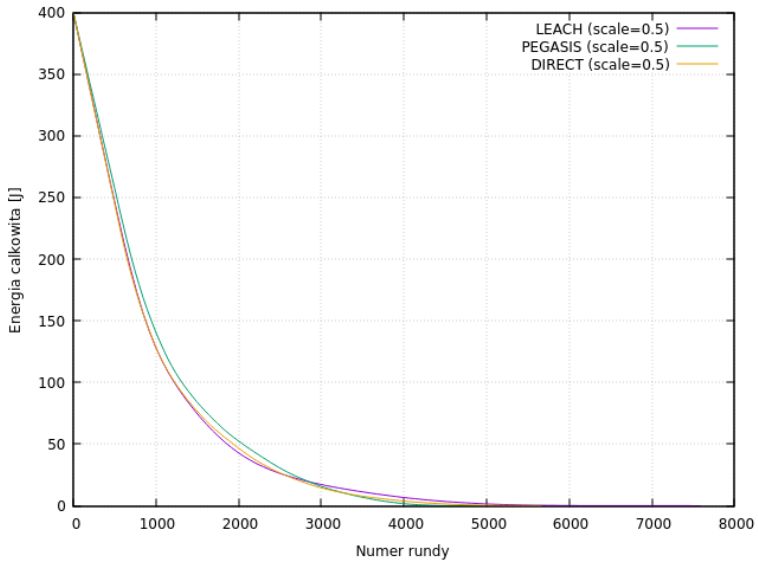
Protokół: PEGASIS, Skala sieci: 0.50

Kwantyl	1	5	10	25	50	75	90
Wartość	4407	4455	4473	4516	4577	4627	4691

Kwantyl	95	99	100
Wartość	4708	4776	4920



Rysunek 5.22: Funkcja liczby węzłów aktywnych w rundzie.



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

Maksymalna liczba 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.

Protokół: DIRECT, Skala sieci: 2.0

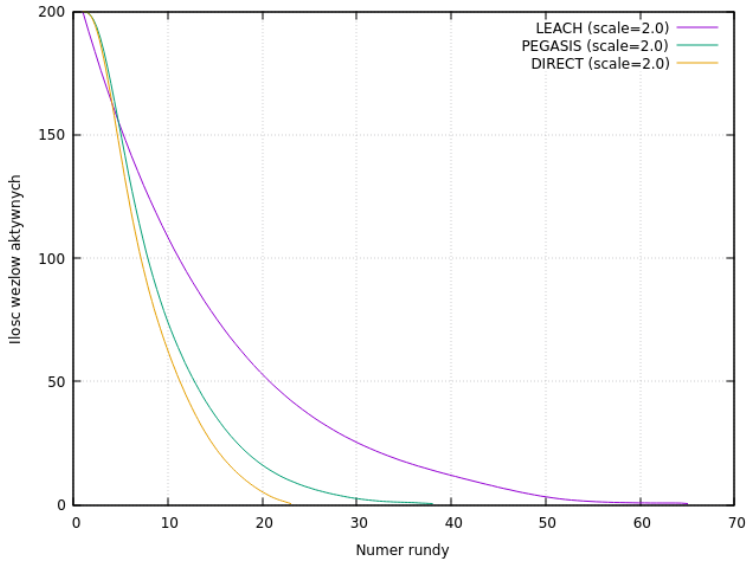
Kwantyl	-
Wartość	23

Protokół: LEACH, Skala sieci: 2.0

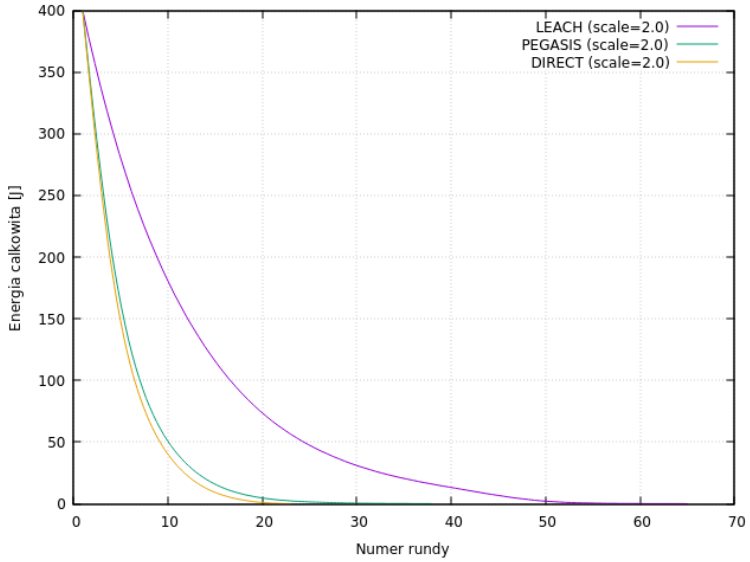
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	49	51	51	53	55	58	60	61	63	65

Protokół: PEGASIS, Skala sieci: 2.0

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	27	29	29	30	32	33	34	36	37	38



Rysunek 5.24: Funkcja liczby węzłów aktywnych w rundzie.



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

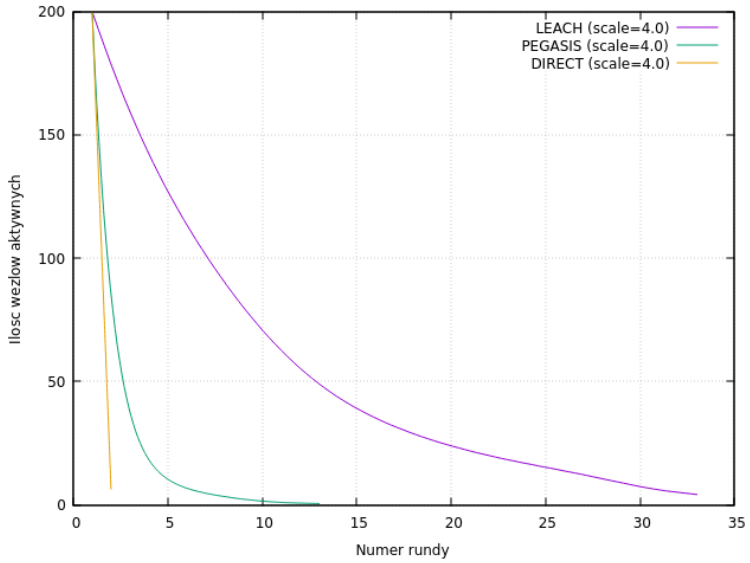
Maksymalna liczba rund dla komunikacji bezpośredniej (DIRECT) jest parametrem stałym i wynosi 23 (uzyskana przez węzeł znajdujący się najbliższej węzła głównego *sink*). Dla LEACH uzyskano wydłużenie czasu pracy na poziomie +213.0% dla wartości minimalnej (49), +239.1% dla mediany (55), +282.6% dla wartości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

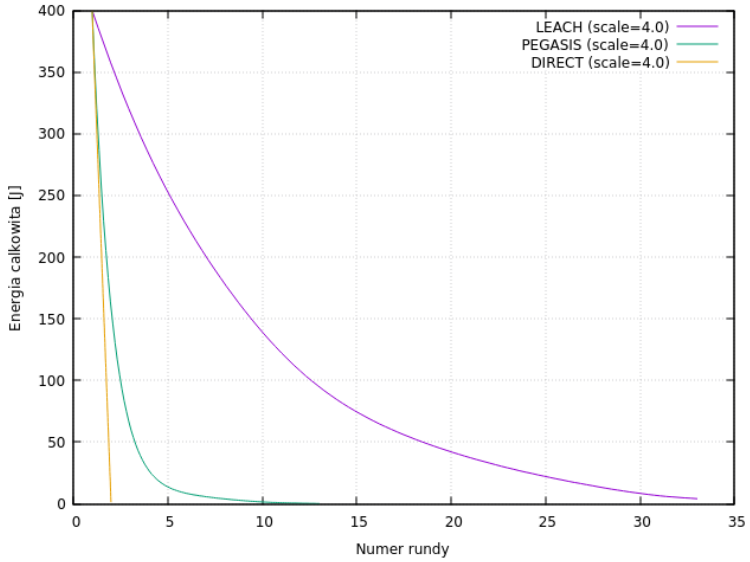
Protokół: DIRECT, Skala sieci: 4.0	
Kwantyl	-
Wartość	2

Protokół: LEACH, Skala sieci: 4.0										
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	27	28	29	29	30	31	32	32	33	33

Protokół: PEGASIS, Skala sieci: 4.0										
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	7	8	9	9	10	11	11	12	13	13



Rysunek 5.26: Funkcja liczby węzłów aktywnych w rundzie.



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

Maksymalna liczba 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.

Folder testdatatest_4 zawiera 13 plików konfiguracyjnych, w których parametry węzłów i sieci są takie same jak w konfiguracji domyślnej. W nazwie pliku występuje ciąg znaków (np. *p010*), który oznacza wartość paramatru ‘p‘ wykorzystanego w scenariuszu.

```
~/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
```

Komenda uruchamiająca oprogramowanie symulatora dla testu z wykorzystaniem 13 plików konfiguracyjnych:

```
~/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
```

Poniższe tabele przedstawiają wyniki uzyskane w ramach symulacji, powtórzonej 100 razy na każdym z plików konfiguracyjnych.

Protokół: DIRECT, Parametr ‘p‘: 0.05

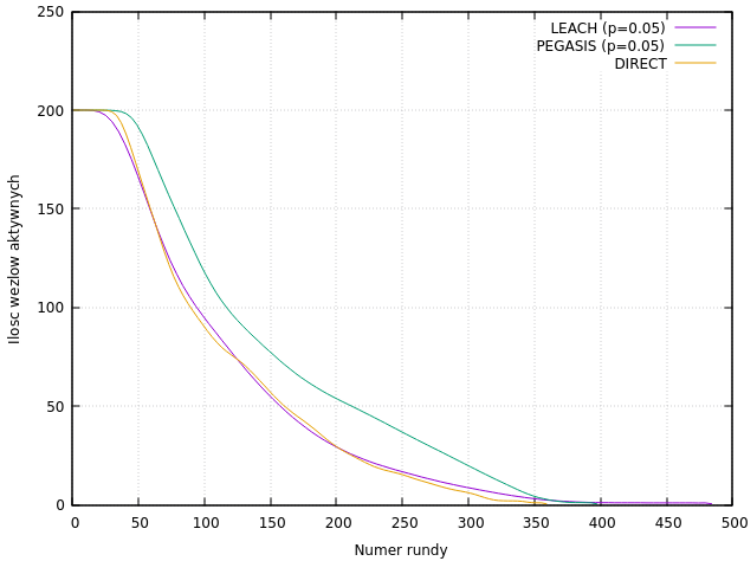
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.05

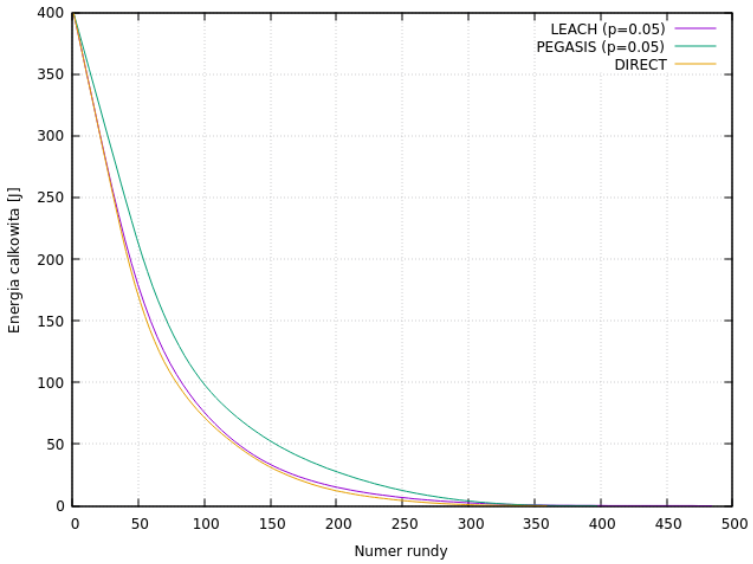
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	356	363	365	383	396	417	435	455	476	484

Protokół: PEGASIS, Parametr ‘p‘: 0.05

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	345	351	352	356	365	372	377	380	387	389



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



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

Maksymalna liczba 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.

Protokół: DIRECT, Parametr ‘p‘: 0.10

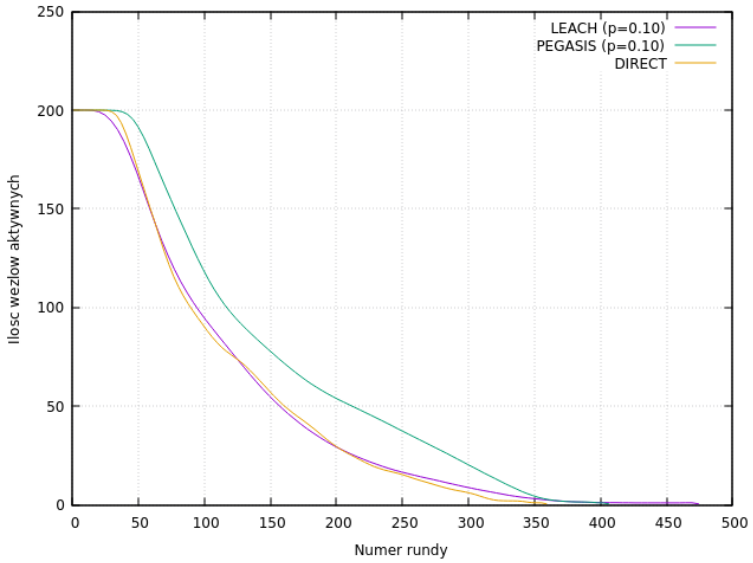
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.10

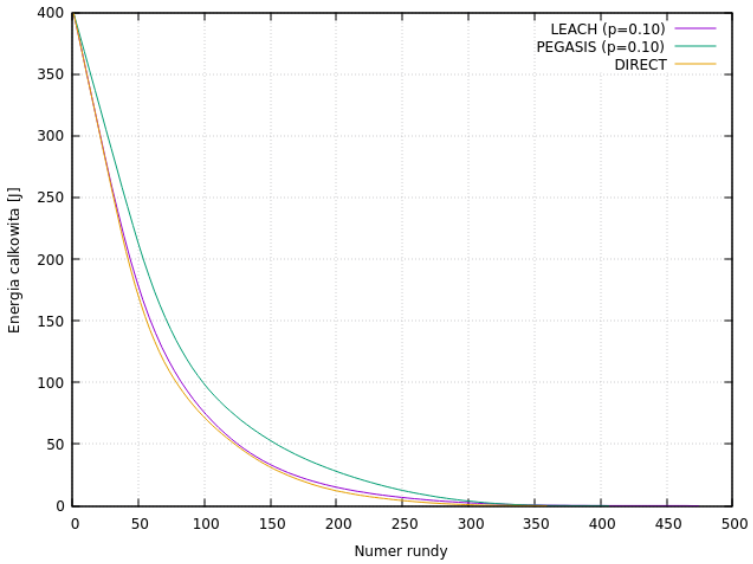
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	364	366	373	386	400	413	430	436	455	474

Protokół: PEGASIS, Parametr ‘p‘: 0.10

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	341	347	351	359	364	372	377	380	388	395



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



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

Maksymalna liczba 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 skró-

cenie 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).

Protokół: DIRECT, Parametr ‘p‘: 0.15

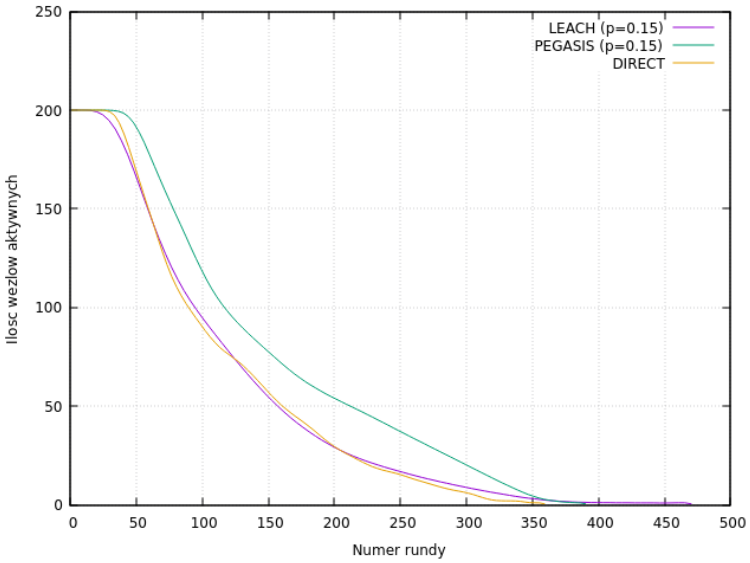
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.15

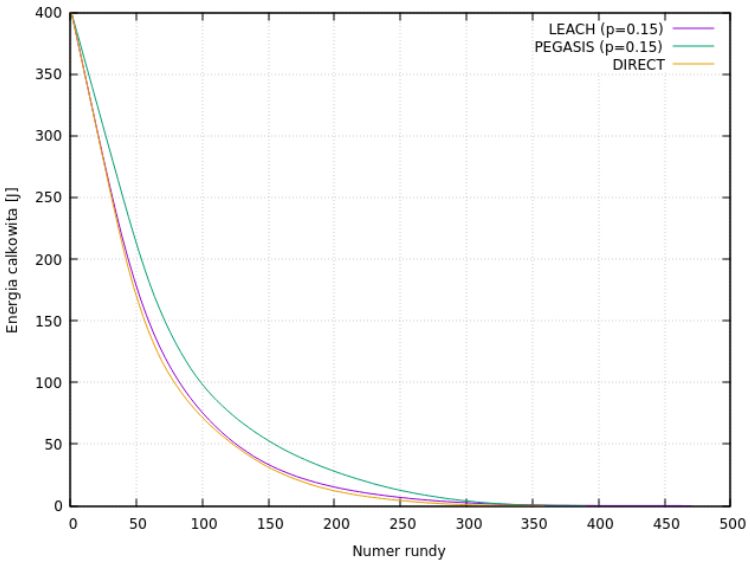
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	357	367	371	386	399	419	434	442	452	470

Protokół: PEGASIS, Parametr ‘p‘: 0.15

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	340	346	350	355	365	371	382	386	402	411



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



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

Maksymalna liczba 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.

Protokół: DIRECT, Parametr ‘p‘: 0.20

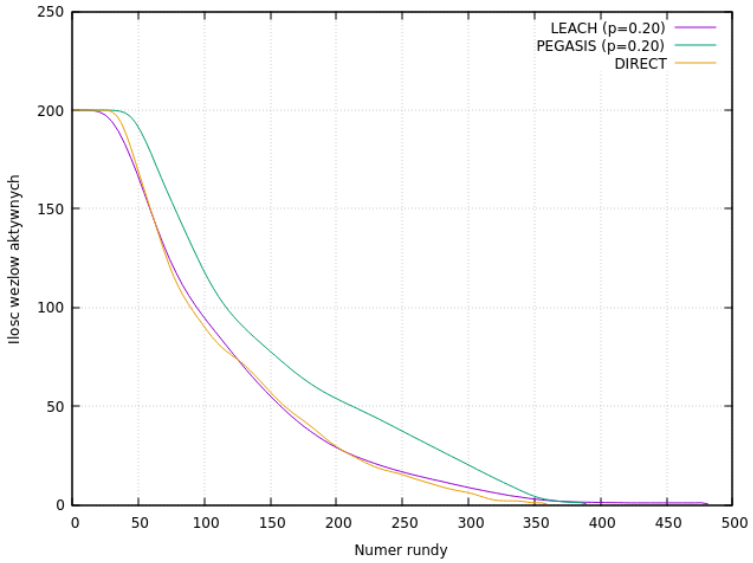
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.20

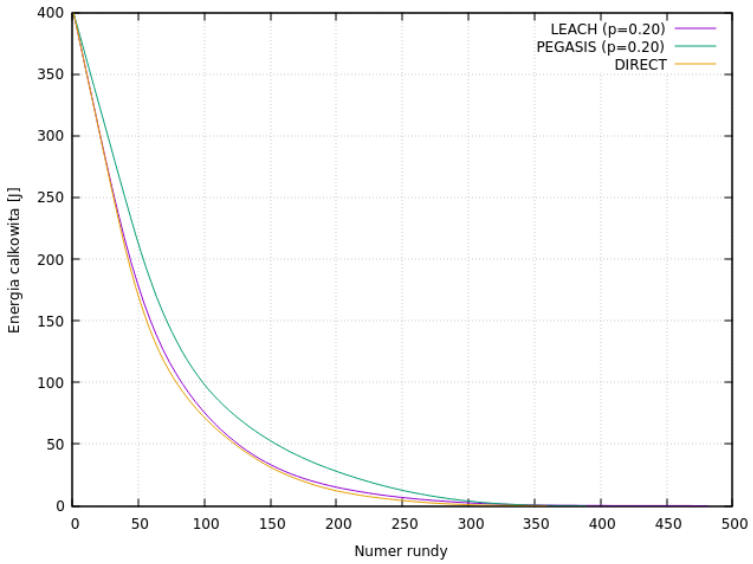
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	362	366	372	386	403	423	442	451	471	481

Protokół: PEGASIS, Parametr ‘p‘: 0.20

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	347	349	352	358	367	373	378	382	388	397



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



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

Maksymalna liczba 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 skró-

cenie 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.

Protokół: DIRECT, Parametr ‘p‘: 0.25

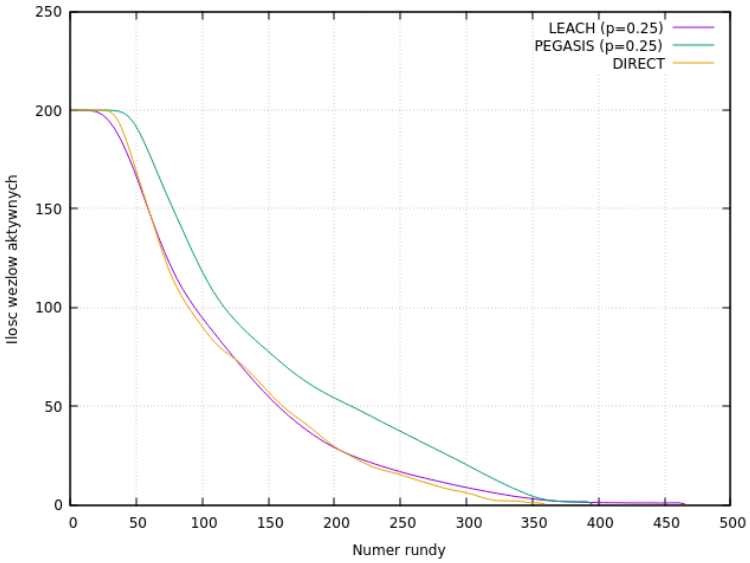
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.25

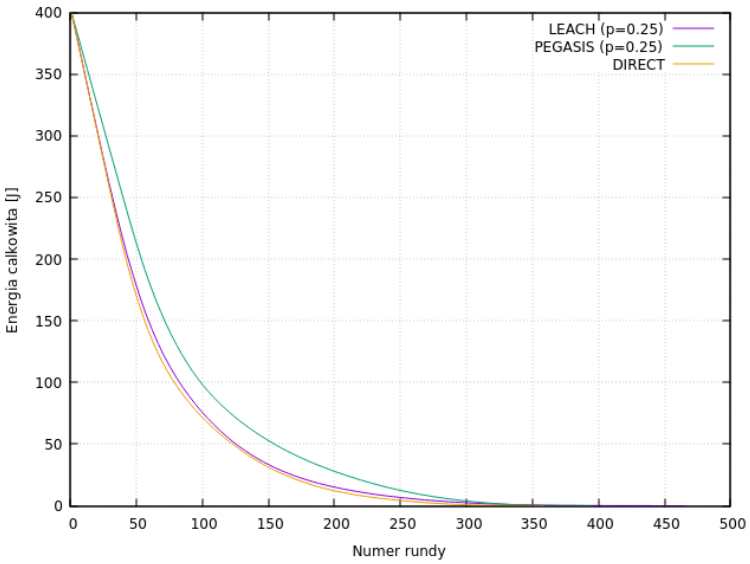
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	359	363	371	384	404	429	439	450	454	465

Protokół: PEGASIS, Parametr ‘p‘: 0.25

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	339	347	350	356	362	370	381	385	397	406



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



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

Maksymalna liczba 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.

Protokół: DIRECT, Parametr ‘p‘: 0.30

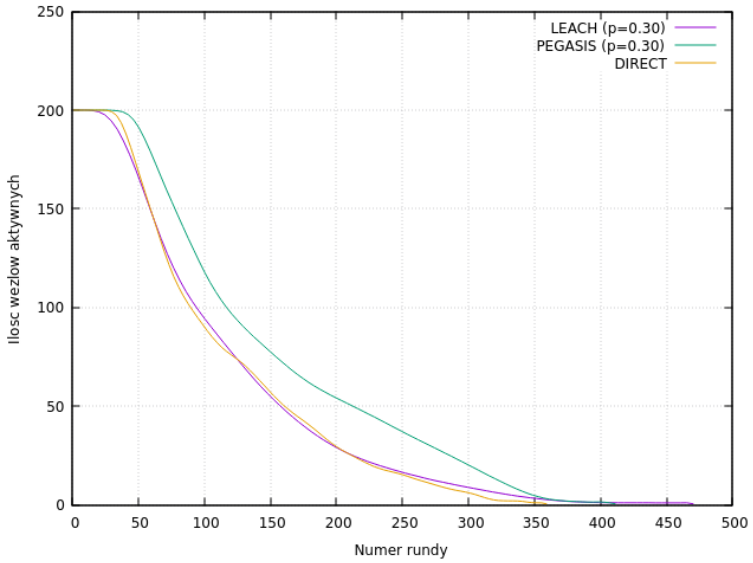
Kwantyl	-
Wartość	359

Protokół: LEACH, Parametr ‘p‘: 0.30

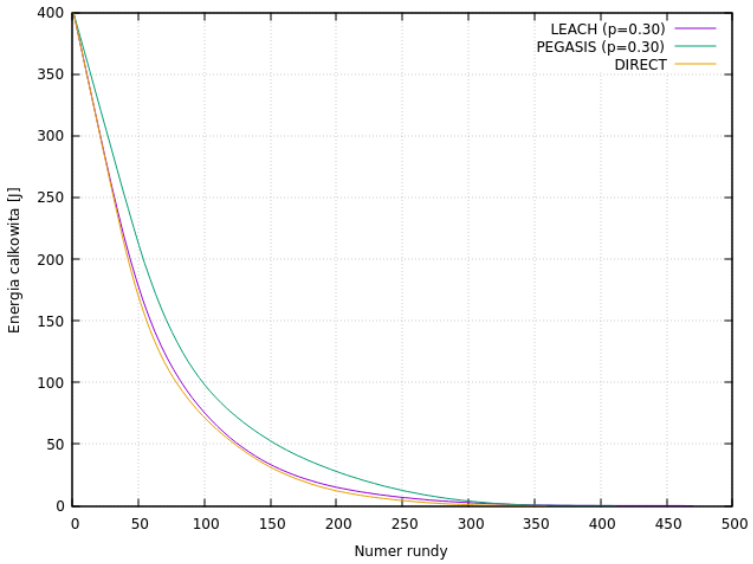
Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	352	369	373	385	399	418	431	444	456	470

Protokół: PEGASIS, Parametr ‘p‘: 0.30

Kwantyl	1	5	10	25	50	75	90	95	99	100
Wartość	348	350	354	359	365	372	377	384	389	390



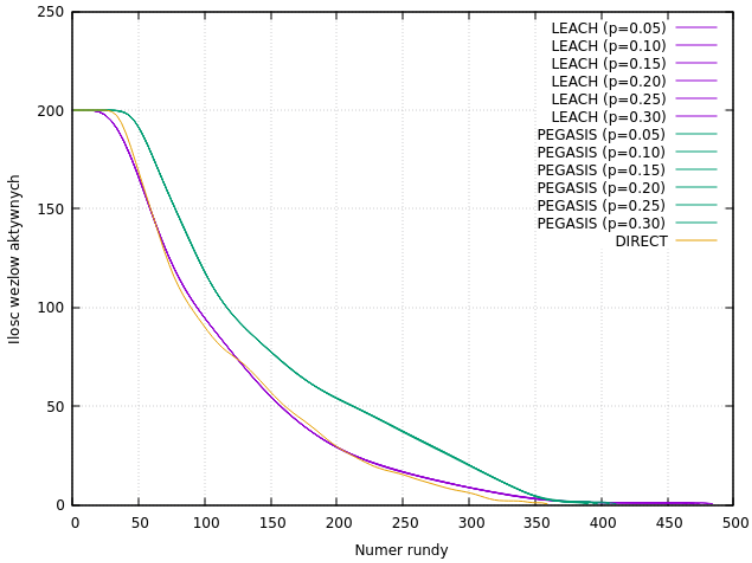
Rysunek 5.38: Funkcja liczby węzłów aktywnych w rundzie (p=0.30).



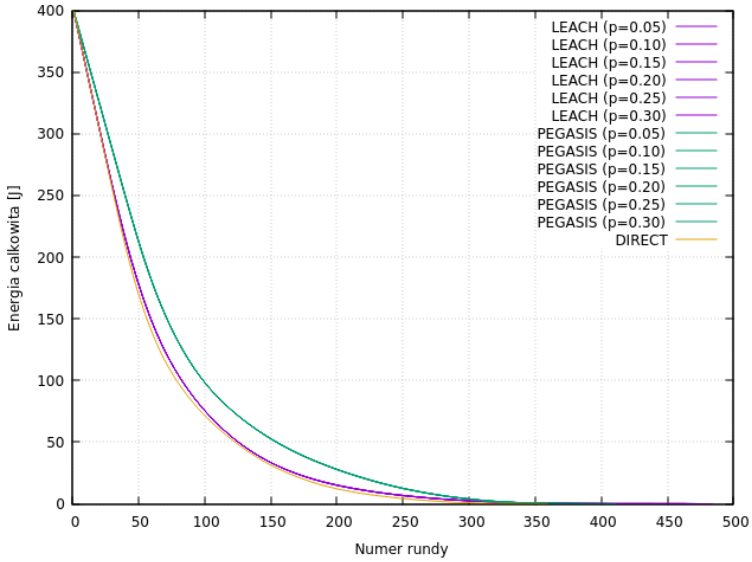
Rysunek 5.39: Funkcja całkowitej ilości energii wszystkich węzłów w rundzie (p=0.30).

Maksymalna liczba 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 liczby 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

W niniejszej pracy przedstawiony został autorski projekt oprogramowania do symulacji bezprzewodowych sieci czujnikowych. Pierwszym etapem pracy było określenie wymagań oraz cech systemu końcowego. Proces ten pozwolił mi na zebranie szczegółowych wymagań projektowych. W końcowym etapie zbierania wymagań, zdecydowałem się na stworzeniem prototypu systemu symulującego. Decyzja ta pozwoliła na wczesne przetestowanie zebranych założeń i pomysłów dotyczących dekompozycji problemu, których efekt zaowocował uproszczeniem całkowitej architektury systemu [28]. Projekt zrealizowałem w języku Go, przy wykorzystaniu rozszerzonego systemu kontroli wersji git. Kod źródłowy finalnej wersji systemu symulacyjnego jest dostępny publicznie na portalu GitHub (github.com/keadwen/msc_project), co pozwala na weryfikację oprogramowania, samodzielne przetestowanie jego działania oraz zaproponowanie ulepszeń.

Zbieranie wyników generowanych przy pomocy systemu symulacyjny podzieliłem na cztery główne testy. W każdym z testów wybrałem dokładnie jeden parametr konfiguracji, który podlegał wielokrotnym zmianom mającym na celu zaobserwowanie zależności czasu i efektywności pracy sieci. Najistotniejszym wnioskiem zebranym zarówno w momencie pisania pracy, jak i procesie uzyskiwania wyników jest zależność między wartościami 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. Wynika to z pseudolosowego wybierania węzłów pełniących rolę węzła typu *cluster head*.

Każdy z czterech głównych testów definiuje kilka dodatkowych scenariuszy, w których wartości parametru modyfikowanego podlega zmianie. Domyślna wielkość wiadomości wynosi 128 [B]. Test numer 1 składa się z czterech scenariuszy o różnych wartościach wielkości wiadomości. Wielkość wia-

domości wysyłanej pomiędzy węzłami podlega modyfikacji, a wielkości te wynoszą 32, 64, 128 oraz 256 [B]. Pozostałe parametry węzłów (t.j. lokalizacji węzłów, parametr 'p', energia początkowa) i sieci (t.j. pozycja węzła głównego typu *sink*) są takie same jak w konfiguracji domyślnej. Stworzenie wielu scenariuszy dla każdego z testów dało mi możliwość zaobserwowania w jaki sposób poszczególne wartości wypływają na pracę sieci i efektywność protokołów.

Wyniki testu numer 1 pozwalają zaobserwować, że LEACH niezależnie od wielkości wiadomości wybranych w scenariuszach uzyskuje lepsze rezultaty dla każdego z powtórzeń symulacji. Zaskakującym wynikiem jest efektywność PEGASIS, który nie uzyskuje lepszych czasów pracy maksymalnej, pomimo widocznie lepszej sprawności energetycznej w początkowym etapie symulacji.

W teście numer 2 zaobserwowałem jak istotnym jest położenie węzłów. W scenariuszu występują sytuacje gdzie węzłami typu *cluster head* staje się węzły o małej energii bądź sporej odległości do węzła głównego. Pomimo tego, wykresy funkcji całkowitej liczby węzłów aktywnych w sposób precyzyjny pokazują korzyści płynące z implementacji protokołu PEGASIS. Występują rundy w których ilość węzłów aktywnych jest nawet dwukrotnie większa, niż w przypadku komunikacji bezpośredniej czy LEACH.

Najlepsze korzyści płynące z implementacji protokołów LEACH oraz PEGASIS uzyskano dla testu numer 3, w którym dystanse pomiędzy węzłami uległy powiększeniu. Dla sieci powiększonej 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 zaobserwowano w najgorszym przypadku 7 rund, w najlepszym 13. Natomiast sprawność protokołu LEACH zaowocowała utrzymaniem pracy węzłów w 27 rundach dla najgorszego przypadku, oraz 33 rundach dla przypadku najlepszego. W pozostałych testach również można zauważyć korzyści płynące z wykorzystania LEACH oraz PEGASIS, należy jednak zwrócić szczególną uwagę, że nie są to zyski na tak dużą skalę jak w przypadku trzeciego testu.

Test numer 4 w którym parametr 'p' ulegał zmianom w zakresie [0.05, 0.10, 0.15, 0.20, 0.25, 0.30] uwidoczniła sporą przewagę protokołu PEGASIS. Pomimo obciążenia energetycznego wymaganego przez stworzenie łańcucha po którym agregowane dane *sink*, PEGASIS posiada większą sprawność energetyczną przez dłuższy czas symulacji. Funkcja liczy aktywnych węzłów nie jest aż tak stroma, jak w przypadku LEACH czy DIRECT.

W początkowym etapie pracy oczekiwałem, że algorytm PEGASIS będący usprawnieniem protokołu LEACH, przez co umożliwi to uzyskanie dłuższego czasu maksymalnego pracy sieci i poszczególnych węzłów. Na pierwszy

rzut oka, wyniki tego nie odzwierciedlają. Dystrybucja maksymalnego czas pracy w LEACH jest dłuższa. Jednakże, wykresy pozwalają zauważyć, że liczba 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 liczbę danych z większej puli węzłów.

W scenariuszach testu trzeciego 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).

W procesie przeprowadzonych prac nauczyłem się przede wszystkim jak istotnym elementem prac informatycznych jest etap zbierania wymagań oraz tworzenia prototypu. Nabyte doświadczenie pozwoliło mi zbudować zdecydowanie lepsze rozwiązanie końcowe. Dodatkowo, ugruntowałem swoją wiedzę teoretyczną z zakresu bezprzewodowych sieci czujnikowych, algorytmów i protokołów wykorzystywanych w sieciach w sposób praktyczny.

Istnieje wiele elementów pracy które warte byłyby przeprowadzenia dalszych badań. Począwszy od implementacji dodatkowych protokołów, nie tylko z puli protokołów hierarchicznych (t.j. PEGASIS, LEACH) [23], ale również protokołów t.j. SAR, SPEED. Protokoły w których przesyłane dane posiadają różne priorytety (*Quality of Service*) Interesującym byłoby również zaimplementowanie protokołów wykorzystujących wielotorowość (M-MPR *Mesh MultiPath Routing*) [13] [18] [17]. Korzystnym z perspektywy istniejących protokołów byłoby zaimplementowanie metody agregacji i kompresji zebranych danych przed ich przesłaniem do węzła docelowego. Zdefiniowanie w plikach konfiguracyjnych kosztów energetycznych pracy węzła, wynikającego z czasu pracy procesora, kosztów strat wynikających z odbierania i przesyłania danych. Wersja oprogramowani dostępna na GitHub umożliwia definiowanie dodatkowych kosztów energetycznych *EnergyCost* [14] czy opóźnień czasowych *TimeDelay* w pliku konfiguracyjnym. Finalnie, przydatnym byłoby napisanie większej bazy testów jednostkowych, umożliwiających weryfikację aktualnego oprogramowania.

Bibliografia

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

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

[3] G. Amato, A. Caruso, S. Chessa, V. Masi, A. Urpi, „State of the art and future directions in wireless sensor network's data management.", ISTI, 2004.

[4] S. Carn, A. Krueger, S. Arellano, N. Krotkov, K. Yang, „Daily monitoring of Ecuadorian volcanic degassing from space", Journal of Volcanology and Geothermal Research, Vol. 176, s. 141-150, 2008.

[5] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. Peh, D. Rubenstein, „Energy-efficient computing for wildlife tracking: design tradeoffs and early experiences with ZebraNet.", ACM SIGOPS Operating Systems Review, Vol. 36(5), s. 96–107, 2002.

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

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

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

[9] J. Al-karaki , A. Kamal, „Routing Techniques in Wireless Sensor Networks: A Survey", IEEE Wireless Communications, 2004.

[10] M. Younis, M. Youssef, K. Arisha, „Energy-Aware Routing in Cluster-Based Sensor Networks", In Proceedings of the 10th IEEE/ACM, 2002.

[11] E. Petriu, N. Georganas, D. Petriu, D. Makrakis, V. Groza, „Sensor-based information appliances.", Instrumentation & Measurement Magazine, Vol. 3(4), s. 31–35, 2000.

[12] C. Shen, C. Srisathapornphat, C. Jaikaeo, „Sensor information networking architecture and applications.", IEEE Personal Communications, Vol. 8(4), s. 52–59, 2001.

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

[14] S. Slijepcevic, M. Potkonjak, „Power efficient organization of wireless sensor networks", IEEE International Conference on Communications ICC'01, 2001.

[15] V. Rodoplu, T. Meng, „Minimum energy mobile wireless networks", IEEE Journal of Selected Areas in Communications, Vol. 17(8), s. 1333-1344, 1999.

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

[17] J. L. Gao, „Energy Efficient Routing for Wireless Sensor Networks", Department of Electrical Engineering, UCLA, 2000.

[18] 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.

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

[20] S. Lindsey, C. Raghavendra, „PEGASIS:Powerefficient gathering in sensor information systems", IEEE Aerospace Conference Proceedings, s. 1125-1130, 2002.

[21] R. Rajagopalan, P. Varshney, „Data aggregation techniques in sensor networks: a survey.", IEEE Communication Surveys & Tutorials, Vol. 8, s. 48–63, 2006.

[22] C. M. Sadler, M. Martonosi, „Data compression algorithms for energy-constrained devices in delay tolerant networks.", In Proceedings of ACM SenSys'06, 2006.

[23] A. Manjeshwar and D. Agrawal „TEEN: a protocol for enhanced efficiency in wireless sensor networks.”, In Proceedings of the 1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing, 2001.

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

[25] 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.

[26] 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.

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

[28] J. Ousterhout, „A Philosophy of Software Design”, ISBN 9781732102200, 2018.

[29] K. Matthias, S. Kane, „Docker: Up & Running: Shipping Reliable Containers in Production, O'Reilly Media, ISBN 9781492036722, 2015.

[30] J. Loeliger, M. McCullough, „Version control with Git”, O'Reilly Media, ISBN 9781449316389, 2012.

[31] L. Phillips, „gnuplot Cookbook”, Packt Publishing, ISBN 9781849517249, 2012.

[32] C. Leiserson, C. Stein, R. Riverst, T. Cormen, „Introduction to Algorithms, 3rd Edition”, The MIT Press, ISBN 9780262033848, 2009.