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

Tytuł pracy dyplomowej

Jakub Młynarczyk
nr albumu 288226

promotor
dr inż. Łukasz Makowski

WARSZAWA 2019

Problem 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, dwóch rozdziałów (2-3) zawierających opis istniejących technologii, dostępnych rozwiązań oraz opisuje autorski system środowiska symulacyjnego. Rozdział treci przedstawia szczegóły implementacji środowiska symulacyjnego. Ostatni rozdział pracy to opis możliwości dalszego rozwoju projektu.

Słowa kluczowe: wsn, sieci czujnikowe, golang

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 presents the fundamentals of wireless sensor networks and technologies, currently available solutions, as well as authored new design proposal. The second chapter presents the authored simulation tool. I In the third chapter presents the implementation of the simulation environemtn in Go programming language. The fourth chapter contains results of tests which compare existing wireless sensor network protocols and proves simulator correctness. Finally some posibilities of further development of the invented algoriths are proposed.

Keywords: wsn, wireless sensor networks, golang

WARSZAWA, 1 lutego 2019

POLITECHNIKA WARSZAWSKA
WYDZIAŁ ELEKTRYCZNY

OŚWIADCZENIE

Świadom odpowiedzialności prawnej oświadczam, że niniejsza praca dyplomowa magisterska pt. Tytuł pracy dyplomowej:

- 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 teleinfromatycznych w zakresie bezprzewodowych sieci czujnikowych. Pierwsze implementacje bezprzewodowych czujników wykorzystano w celach zbrojeniowych. Aktualnie, możliwości oferowane przez bezprzewodowe czujniki znajdują zastosowanie w nieskończonej liczbie aplikacji, zarówno w przemyśle (np. medycynie), jak również w sektorze prywatnym (np. inteligentny dom).

1.1 Cel pracy

Celem pracy są badania problemów współbieżności w algorymatch 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 umożliwiające implementację oraz testowanie nowych algorytmów. Aplikacja umożliwia tworzenie symulacji porównujących efekty zastosowania różnych algorytmów transmisji danych, bez potrzeby wykorzystania fizycznych urządzeń.

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. Jak również metody i technologie niezbędne do wygenerowania danych wejściowych, technik serializacji danych oraz ich reprezentacji przy wykorzystaniu Protocol Buffers, GNUPlot, itd.

2.1 Język programowania Go

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

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

```
package main

import (
  "fmt"

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

type Label struct {
  Text string
```

```
  X, Y int
}

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

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

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

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

Porównanie Go z Python

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

- Python posiada większy bazę publiczych bibliotek.

Porównanie Go z C++

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

Unikatowe cechy Go

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

2.2 Serializacja danych Protocol Buffers

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

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

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

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

Struktura ta przechowywana jest w plikach o rozszerzeniu ‘.proto‘, które są następnie kompilowane do dowolnego z wspieranych języków oprogramowania (w przypadku Protocol Buffers w wersji 3, wspierana jest generacja kodu w Java, C++, Python, Java Lite, Ruby, JavaScript, Objective-C, C# oraz PHP). Następnie, zserializowane dane 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 Nardzędzia dodatkowe

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

2.3.1 System kontroli wersji git

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

2.3.2 GNUPlot

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

2.3.3 Docker

Docker to narzędzie stworzone jako wolne oprogramowanie (open source) napisane w języku Go przez firmę Docker, Inc. Narzędzie to pozwala na tworzenie kontenerów, które izolują aplikację na poziomie systmeu 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 rodziale przedstawiona zostanie najważniejsza część teoretyczna bezprzewodowych sieci sensorych, która jest niezbędna w celu zrozumienia problemu rozwiązywanego w pracy dyplomowej.

3.1 Wstęp do bezprzewodowych sieci sensorowych

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

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

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

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

3.2 Model transmisji

TODO * Koszt transmisji * Założenia dotyczące opóźnień i innych parametrów

3.3 Protokoły

3.3.1 Komunikacja bezpośrednia

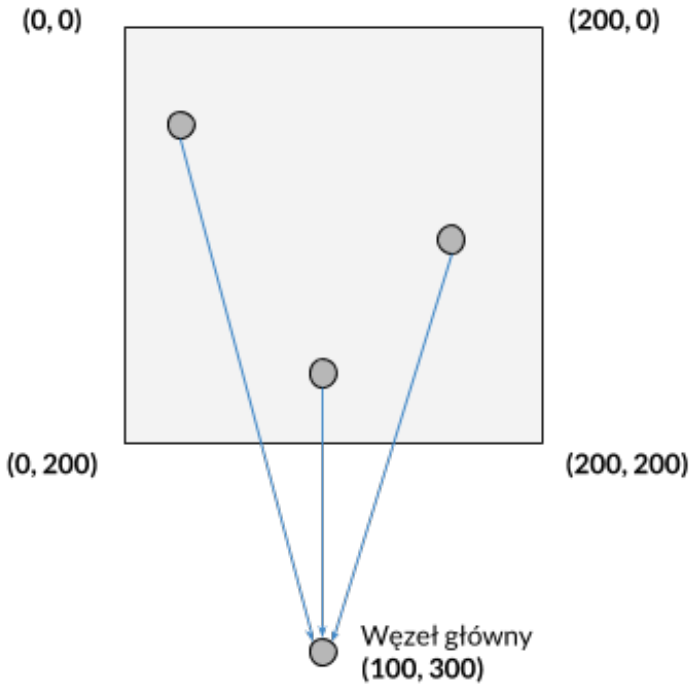
Komunikacja bezpośrednia jest rozwiązaniem najprostrzym z perspektywy implementacji, jednakże nieefektywnym z poziomu energetycznego. Węzły znajdujące się sieci nie są zaangażowane w podejmowanie decyzji dotyczących optymalizacji kosztów transmisji. Adres węzła głównego (sink) może być zaprogramowany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

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

3.3.2 LEACH

LEACH (Low-energy adaptive clustering hierarchy) jest jednym z wielu algortmemów wykorzystywanych w protokołach routingu bezprzewodowej sieci czujnikowej. W przeciwieństwie do komunikacji bezpośredniej, LEACH jest protokołem w którym występuje hierarchia. Węzły uczestniczące w sieci dzielą się na dwie kategorie:

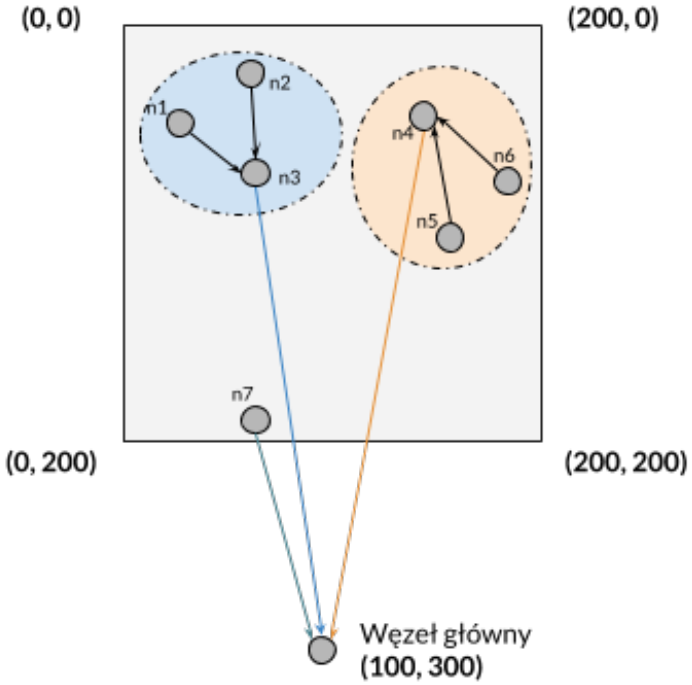
- węzły typu cluster head (CH)
- standardowe węzły (non-CH)



Standardowe węzły dokonują transmisji danych do najbliższego węzła typu cluster head, które następnie dokonują agregacji zebranch 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.

TODO * Przedstawienie algorytmu * Korzyści oraz wyzwania związane z implementacją

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



3.3.3 PEGASIS

TODO

Rozdział 4

Architektura systemu

System informatyczny stworzony na potrzeby pracy dyplomowej ma za zadanie dostarczenie wyników oraz wykresów niezbędnych do zbadania poniższych zależnożci 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.

4.1.1 Plik konfiguracyjny

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

- konfiguracja symulatora i sieci
- konfiguracja węzła

```
syntax = "proto3";

package config;

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;
}

// 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;
}

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

// 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;
}

// 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;
}
```

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 kluster 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. Długosc 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 poszeczególnych węzłów przedstawiony został w kolejnym porozdziale 'Konfiguracja węzła'.

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 umieszenie węzła w dwuwymiarowej przestrzenii. 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 posiada ogórnie zdefiniowane koszty energetyczne (model przedstawiony w sekcji TODO). 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.

4.1.2 Moduł główny (Core)

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

Węzeł (Node)

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

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

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

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

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

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

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

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

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

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

Sieć (Network)

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

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

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

  Round    int64
  MaxRounds int64
  MsgLength int64

  GNUPlotNodes     []string
  GNUPlotTotalEnergy []string

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

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

Poniższa lista pzedstawia oraz opisuje znaczenie poszeczególnych zmiennych struktury sieci (Network):

- protokół (Protocol) - zmienna publiczna przechowuje obiekt definiujący protokół komunikacji pomiędzy węzłami.
- stacja bazowa (BaseStation) - zmienna publiczna przechowuje obiekt węzła głównego (sink).
- węzły (Nodes) - zmienna publiczna przechowuje obiekty węzłów w sieci. Implementacja przy wykorzystaniu dziennika (hashmap), który umoż-

liwia operacje zapisu i odczytu w procesach równoległych. Kluczem dziennika jest unikatowy identyfikator węzła.

- runda (Round) - zmienna publiczna przechowuje numer aktualnej rundy symulacji.
- maksymalna ilość rund (MaxRounds) - zmienna publiczna przechowuje maksymalną ilość rund symulacji. W przypadku osiągnęcia wartości Round równej MaxRounds, symulacja zostanie przerwana.
- długość wiadomości (MsgLength) - zmienna publiczna przechowuje informację o całkowitej wielkości wiadomości (mierzonej w bajtach) jaka generowana jest podczas rundy przez każdy z węzłów. W skład tej wartości wchodzią dane pomiarowe i dodatkowy nakład informacji powstały w wyniku enkapsulacji.
- GNUPlotNodes, GNUPlotTotalEnergy - zmienne publiczne przechowujące parametry rund wykorzystywane do generowania grafów przy użyciu narzędzia GNUPlot.
- PlotTotalEnergy, PlotNodes, NodesAlivePoints, NodesEnergyPoints - zmienne publiczne przechowujące parametry rund wykorzystywane do generowania grafiów przy użyciu biblioteki plotter.

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

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

Protokół (Protocol)

Protokół (Protocol) reprezentuje model protokołu komunikacji między wezłami.

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

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

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

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

Poniższa lista przedstawia zaimplementowane protokoły:

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

4.1.3 Moduł symulatora (Simulator)

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

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

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

func Create() (*Simulator, error)

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

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

Poniższa lista pzedstawia oraz opisuje znaczenie poszeczególnych zmiennych struktury symulatora (Simulator):

- przestrzeń nazw (namespace) - zmienna prywatna przechowuje nazwy symulacji, które w jednoznaczy sposób identyfikują sięć i konfigurację.
- przestrzeń konfiguracji (config) - zmienna prywatna przechowuje obiekt konfiguracji (Config) dla każdej symulacji. Kluczem dziennika jest nazwa symulacji.
- przestrzeń sieci (network) - zmienna prywatna przechowuje obiekt sieci (Network) dla każdej symulacji. Kluczem dziennika jest nazwa symulacji.

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

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

4.2 Obsługa systemu

4.2.1 Konfiguracja środowiska i kompilacja

Poprawna konfiguracja środowiska wymaga instalacji niezbędnych bibliotek i pakietów.

1. Instalacja kompilatora i bilbiotek 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
! Brak   bdw   powinien utworzy plik o nazwie msc_project.
```

4.2.2 Generowanie konfiguracji

Wcześniej opisany plik konfiguracyjny jest elementem niezbędnym do uruchomienia symulacji.

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

```
protocol: 1
nodes: <
  id: 0
```

```
  initial_energy: 1.0e4
  location: <
    X: 0
    Y: 0
  >
  energy_cost <
  >
  time_delay: <
  >
>
nodes: <
  id: 1
  initial_energy: 10.0e-6
  location: <
    X: 300.0
    Y: 0
  >
  energy_cost <
  >
  time_delay: <
  >
>
nodes <
  id: 2
  initial_energy: 20.0e-6
  location: <
    X: 500.0
    Y: 0
  >
  energy_cost: <
  >
  time_delay: <
  >
>
```

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

- Protokół: Bezpośrednia komunikacja, LEACH, PEGASIS

- Maksymalna ilość rund: 25000
- Proporcja węzłów typu kluster do wszystkich węzłów: 0.15
- Liczba węzłów głównych (sink): 1
- Enegia węzła głównego: 100 [J]
- Lokalizacja węzła głównego na osi X: 100
- Lokalizacja węzła głównego na osi Y: 300
- Liczba węzłów: 200
- Enegia węzłów: 1 [J]
- Lokalizacja węzłów na osi X: [0, 200]
- Lokalizacja węzłów na osi Y: [0, 200]

Poniższa ilustracja przedstawia rozstawienie węzłów w sieci wygenerowanych dynamicznie:

4.2.3 Uruchamianie scenariuszy

Przykład uruchamiania projektu z dwoma plikami konfiguracyjnymi:

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

Przykład uruchamiania projektu bez pliku konfiguracyjnego:

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

4.2.4 Generowanie wykresów

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

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

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

W momencie poprawnego zakończenia symulacji, folder TODO powinien posiadać zestaw plików:

Pliki te należy następnie przekierować do GNUPlot w następujący sposób:

4.2.5 Dodawanie nowych protokołów

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

1. Rozszerzenie definicji enum E_Protocol w proto/config.proto.
2. Rozszerzenie mapProtocol w simulator/simulator.go

3. Stworzenie nowego pliku .go w folderze core. Struktura reprezentująca nowy protokołu musi posiadać zestaw funkcji zgodny z interfejsem Protocol.

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

Rozdział 5

Opracowanie wyników eksperymentów

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

5.1 TODO

Bibliografia

[1] W. R. Stevens, G. R. Wright, „Biblia TCP/IP tom 1”, RM, 1998.

Opinia

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

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

Treść opinii

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

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

(data, podpis)

Recenzja

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

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

Treść recenzji

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

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

(data, podpis)