НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

«До захисту допущено»

Завідувач кафедри

_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)

“___” червня 2019 р.

Дипломний проект

на здобуття ступеня бакалавра

з напряму підготовки **6.050102 «Комп'ютерна інженерія»**

на тему: «Система балансування навантаження в децентралізованих розподілених мережах».

Виконав: студент IV курсу, групи КВ-51

Колесник Олександр Сергійович _____
(підпис)

Керівник, доцент каф. СПіСКС, к.т.н. Замятін Д.С. _____
(підпис)

Консультант з нормоконтролю, доц.каф.СПіСКС, к.т.н. Клятченко Я.М. _____
(підпис)

Рецензент_____ _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ФАКУЛЬТЕТ ПРИКЛАДНОЇ МАТЕМАТИКИ

Кафедра системного програмування і спеціалізованих комп‘ютерних систем

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки 6.050102 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Тарасенко В.П.
(підпис) (ініціали, прізвище)
«___»червня 2019 р.

ЗАВДАННЯ
на дипломний проект студента
Колесника Олександра Сергійовича

1. Тема проекту: “Система балансування навантаження в децентралізованих розподілених мережах”.
Керівник проекту: доцент каф. СПіСКС, к.т.н. Замятін Д.С.,
затверджені наказом по університету від «22»травня 2019 р. №1330-С

2. Термін подання студентом проекту _____

3. Вихідні дані до проекту:див. технічне завдання.

4. Зміст пояснювальної записки: аналіз предметної області і постановка задачі, гіпотеза та побудова моделі, тестування системи, технічна реалізація моделі.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): алгоритм LEACH - setup phase, алгоритм LEACH - steady phase, алгоритм Дейкстри, схема функціонування вузла.

6. Консультанти розділів проекту*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Клятченко Я.М., доц. каф. СПіСКС, к.т.н.		

7. Дата видачі завдання_____

Календарний план

№ з/п	Назва етапів роботи та питань, які мають бути розроблені відповідно до завдання	Термін виконання
1.	Видача завдання на дипломне проектування	14.10.2018
2.	Розробка технічного завдання	24.10.2018
3.	Аналіз існуючих рішень	19.01.2019
4.	Вибір середовища розробки	19.02.2019
5.	Розробка програмного продукту	09.03.2019
6.	Відлагодження програмного продукту	09.04.2019
7.	Підготовка пояснювальної записки	29.04.2019
8.	Оформлення матеріалів проекту	09.05.2019
9.	Попередній огляд матеріалів диплому на кафедрі	19.05.2019

Студент _____ Колесник О.С.
(підпис) (ініціали, прізвище)

Керівник проекту _____ Замятін Д.С.
(підпис) (ініціали, прізвище)

АНОТАЦІЯ

Кваліфікаційна робота включає пояснювальну записку (50 с., 23 рис., 3 додатки).

Бакалаврський проект присвячено розробці програмного забезпечення для моделювання функціонування розподіленої сенсорної мережі малої потужності та обмеженим ресурсом споживання електроенергії з метою дослідження ефективності обраного алгоритму балансування навантаження.

Розроблене програмне забезпечення дозволяє будувати модель сенсорної мережі і оцінювати тривалість її життя, що зумовлюється обраними апаратними засобами та алгоритмом балансування навантаження.

В процесі розробки моделі було використано технологію Python та фреймворк Bokeh.

Мережа розроблена за допомогою комплекта MICA2, використано сенсорні мікросхеми MPR400CB MPR410CB.

В ході розробки:

- сформульовані вимоги до розподіленої сенсорної мережі малої потужності;
- описано структуру розташування сенсорів мережі, що задовольняє вимогам предметної галузі використання;
- обрано необхідні апаратні засоби для реалізації мережі;
- розроблено програмне забезпечення (веб додаток), що дозволяє моделювати роботу мережі.

Ключові слова:

СЕНСОРНА МЕРЕЖА, РОЗПОДІЛЕНІ МЕРЕЖІ, АЛГОРИТМ БАЛАНСУВАННЯ, WEB-ДОДАТОК, КОМПЛЕКТ MICA2, MPR400CB, MPR410CB, PYTHON, BOKEH.

ABSTRACT

The diploma project includes an explanatory note (50 p., 23 fig., 3 appendices).

The Bachelor's project is designed to develop a software for modelling a distributed low power sensor network with a limited battery charge resource. The aim of this software is an investigation of a chosen load balancing algorithm.

The developed software provides a building of a sensor network model and evaluation of its lifetime that is caused by a chosen network nodes hardware and a load balancing algorithm.

For software development a technology Python and a framework Bokeh were used.

The network was created using a MICA2 kit, MPR400CB and MPR410CB sensor boards.

In the development process were resolved:

- analysis of;
- formulation of the requirements for a distributed low power sensor network;
- description of a network nodes layout structure that satisfies the requirements of subject;
- consideration of a software needed for a network implementation;
- development of a software (web application) that provides a network functioning modelling

Key words:

SENSOR NETWORK, DISTRIBUTED NETWORK, LOAD BALANCING ALGORITHM, WEB APPLICATION, MICA2 KIT, MPR400CB, MPR410CB, PYTHON, BOKEH.

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.002 ТЗ	Система балансування	4		
			навантаження в			
			децентралізованих			
			розподілених мережах.			
			Технічне завдання			
	А4	ІАЛЦ.045492.003 ТП	Система балансування	2		
			навантаження в			
			децентралізованих			
			розподілених мережах.			
			Відомість технічного			
			проекту			
	А4	ІАЛЦ.045492.004 ПЗ	Система балансування	51		
			навантаження в			
			децентралізованих			
			розподілених мережах.			
			Пояснювальна записка			
	А4	ІАЛЦ.045492.005 Д1	Алгоритм LEACH - setup	1		
			phase.			
			Схема алгоритму.			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045492.001 ОА		
Розробив		Колесник А.С.			Система балансування навантаження в децентралізованих розподілених мережах **Опис альбому**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Замятін Д.С.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-51	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.006 Д2	Алгоритм LEACH - steady	1		
			phase.			
			Схема алгоритму.			
	А4	ІАЛЦ.045492.007 Д3	Алгоритм Дейкстри.	1		
			Схема алгоритму.			
	А4	ІАЛЦ.045492.008 Д4	Схема функціонування	1		
			вузла.			
			Схема алгоритму.			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

ЗМІСТ

1. НАЙМЕНУВАННЯ ТА ГАЛУЗЬ РОЗРОБКИ

Назва розробки: «Система балансування навантаження в децентралізованих розподілених мережах».

Галузь застосування: дослідження оточуючого середовища за допомогою сенсорів на відкритій місцевості.

2. ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання роботи першого (бакалаврського) рівня вищої освіти, затверджене кафедрою системного програмування і спеціалізованих комп'ютерних систем Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського».

3. МЕТА І ПРИЗНАЧЕННЯ РОБОТИ

Метою даного проекту є підвищення часу життя розподіленої мережі за рахунок зменшення загального енергоспоживання вузлів мережі.

4. ДЖЕРЕЛА РОЗРОБКИ

Джерелом інформації є науково-технічна література з теми побулови розподілених сенсорниз мереж, балансувальників навантаження, електронні статті у мережі Інтернет, що порушують ці питання, а також документація до програмних та апаратних засобів.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до програмного продукту, що розробляється

Програма повинна мати інтерфейс, за допомогою якого можна побудувати мережу, розташовуючи вузли за допомогою мишки та запускати моделювання, натискаючи кнопку.

Результатом роботи є зображення з конфігурацією мережі на початку роботи системи, а також вивід про кількість вузлів у мережі, час першого від’єднання вузла, а також загальний час життя мережі.

5.2. Вимоги до апаратного забезпечення

Комп'ютер на базі процесора сімейства AMD, A4-4020 та вище або сімейства Intel, G3900 та вище з оперативною пам'яттю 256 Мбайт і більше.

5.3. Вимоги до програмного та апаратного забезпечення користувача

1) Операційна система одного з сімейств:
   - Windows XP або вище;
   - Linux 3.0 або вище;
   - Mac OS X 10.7.3 (Lion) або вище.
2) Додаткові інсталяційні пакети:
   - Python 3;
   - Bokeh 1.2.0.

6. ЕТАПИ РОЗРОБКИ

№ з/п	Назва етапів виконання дипломного проекту	Термін виконання етапів
1.	Видача завдання на дипломне проектування	04.11.2018
2.	Розробка технічного завдання	16.11.2018
3.	Аналіз існуючих рішень	17.01.2019
4.	Вибір середовища розробки	09.02.2019
5.	Розробка програмного продукту	04.03.2019
6.	Відлагодження програмного продукту	04.04.2019
7.	Підготовка пояснювальної записки	29.04.2019
8.	Оформлення матеріалів проекту	18.05.2019
9.	Попередній огляд матеріалів диплому на кафедрі	27.05.2019

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки
	А4	ІАЛЦ.045492.004 ПЗ	Система балансування	51		
			навантаження в			
			децентралізованих			
			розподілених мережах.			
			Пояснювальна записка			
	А4	ІАЛЦ.045492.005 Д1	Алгоритм LEACH - setup	1		
			phase.			
			Схема алгоритму.			
	А4	ІАЛЦ.045492.006 Д2	Алгоритм LEACH - steady	1		
			phase.			
			Схема алгоритму.			
	А4	ІАЛЦ.045492.007 Д3	Алгоритм Дейкстри.	1		
			Схема алгоритму.			
	А4	ІАЛЦ.045492.008 Д4	Схема функціонування	1		
			вузла.			
			Схема алгоритму.			
		Диск CD-ROM	Текст пояснювальної	1		
			записки.			
			Графічний матеріал			

Змін.	Арк.	№ докум.	Підпис	Дата	ІАЛЦ.045492.003 ТП		
Розробив		Колесник А.С.			Система балансування навантаження в децентралізованих розподілених мережах **Відомість технічного проекту**	Літ.	Аркуш 1 / Аркушів 2
Перевірив		Замятін Д.С.					
Консульт.						КПІ ім. Ігоря Сікорського, ФПМ КВ-51	
Н. контроль		Клятченко Я.М.					
Зав. каф.		Тарасенко В.П.					

Поз.	Формат	ПОЗНАЧЕННЯ	НАЙМЕНУВАННЯ	Кількість аркушів	№ прим.	Примітки

Змін.	Арк.	№ докум.	Підпис	Дата	ДП.468300.003 ТП	Арк.
						2

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, ТЕРМІНІВ

CSS	–	Cascading StyleSheets;
DAL	–	Data access level;
LEACH	–	Low-energy adaptive clustering hierarchy;
MVC	–	Model View Controller;
SQL	–	Structured query language;
WSN	–	Wireless Sensor Network;
БМС	–	Базова мікросхема-інтерфейс;
БСМ	–	Безпровідна сенсорна мережа;
ГОСТ	–	«государственный стандарт» (міждержавний стандарт СНД);
ДСТУ	–	Державний Стандарт України;
НТУУ «КПІ»	–	Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського»;
ОКР	–	освітньо-кваліфікаційний рівень;
ПЗ	–	Програмне забезпечення;
ПК	–	Персональний комп`ютер.

ВСТУП

В останні роки значно зріс інтерес до розгортання розподілених мереж радіодатчиків (сенсорних мереж) для вирішення завдань розподіленого зондування, збору й обробки даних. На відміну від мереж зв'язку, які основані на протоколі ір та працюють з глобальною адресацією і маршрутизацією, у сенсорних вузлах, як правило, глобальні адреси відсутні. Крім того, оскільки обслуговування спеціалізованої мережі після розгортання, як правило, неможливе, існують обмеження за часом функціонування (через низький заряд батареї).

У даній роботі мається опис архітектури мережі, орієнтованої на спеціалізовані сенсорні мережі датчиків, апаратні засоби, які використовуються для реалізації цієї системи а також реалізація алгоритму балансування навантаження. Запропонована архітектура працює для типового сценарію застосування: зондування поверхні водойми або сільськогосподарських угідь малого розміру (до 800 м завдовжки).

Мережа, призначена для такого використання, повинна мати наступні властивості:

- Бути фінансово доступною для невеликих підприємств та окремих фермерів
- Стабільно працювати в зазначеному радіусі розгортання
- Бути легкою у налаштуванні та розгортанні

З урахуванням цих особливостей сенсорні мережі потребують особливої уваги стосовно мінімізації енергоспоживання на більшості рівнів стека протоколів. Для вирішення цього завдання більшість досліджень зосереджуються на продовженні часу життя мережі, забезпеченні масштабованості великої кількості сенсорних вузлів, підвищенні відмовостійкості (наприклад, стійкості до помилок сенсорів

або розряду джерела). Способом досягнення поставленої мети в даній роботі є впровадження балансування навантаження на систему, враховуючи рівень електроенергії вузлу, його розташування, роль, яку він виконує у мережі, а також типове для даної задачі розташування вузлів на поверхні. Для цього було поєднано алгоритми LEACH розподілення вузлів на кластери та алгоритм Дейкстри для встановлення найкоротшого зв’язку між базовою станцією та головами кластерів. Основною відмінністю підходу, основаного на принципах мультиагентного управління, є покращення надійності та спрощення стандартного для цієї задачі підходу балансування.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАДАЧІ

1.1. Мета роботи

Метою нашої роботи є підвищення часу життя розподіленої мережі за рахунок зменшення загального енергоспоживання вузлів мережі.

1.2. Задачі роботи

Задачами роботи є дослідження предметної області, аналіз аналогічних систем, вибір апаратних та програмних технологій розробки мережі, вибір програмної технології побудови моделі мережі, імплементація алгоритму балансування навантаження, прогнозування поведінки мережі та часу її життя, визначення дослідним шляхом конфігурацій мережі, що дають найдовший час життя системи.

1.3. Огляд існуючих сенсорних систем

Сенсорні мережі можуть бути використані в багатьох прикладних областях. Бездротові сенсорні мережі — це нова перспективна технологія, і всі пов'язані з нею проекти в основному перебувають у стадії розробки. Надамо основні області застосування даної технології:

- системи оборони й забезпечення безпеки;
- контроль навколишнього середовища;
- моніторинг промислового встаткування;
- охоронні системи;
- моніторинг стану сільськогосподарських угідь;
- керування енергопостачанням;
- контроль систем вентиляції, кондиціювання й освітлення;

- пожежна сигналізація;
- складський облік;
- спостереження за транспортуванням вантажів;

Реальний приклад використання WSN - бездротова система моніторингу стану будівельних конструкцій, розроблена російською компанією methlogic в 2010 році. Система забезпечує збір, реєстрацію й відображення показань від безлічі датчиків, установлених на різних елементах конструкцій для контролю їх напружено-деформованого стану й структурної цілісності.

Основним призначенням сучасних ТКС є забезпечення різних інформаційних систем: Internet, банківських, сенсорних тощо. Головний напрямок розвитку сучасних телекомунікаційних систем у світі засновано на створенні мереж наступного покоління (NGN, Next Generation Network), які розглядаються як конвергентні, гібридні широкосмугові мережі, які інтегрують різноманітні мережні архітектури і платформи. Користувач мережі повинен обслуговуватися найбільш відповідним на даний момент підключенням, який задовольняє вимогам до якості обслуговування для використовуваного застосування. Такі особливості NGN засновані на підтримці декількох мереж доступу. Отже, мережі NGN повинні бути досить гнучкими і забезпечувати всі послуги, які надаються на даний час у фіксованих мережах з різноманітними технологіями без будь-яких обмежень, використовуючи оптимальний метод підключення. Актуальною стає проблема не тільки охоплення мережі, забезпечення якості зв’язку, але й розширення переліку додаткових послуг, що надаються. Серед сучасних технологій на рівні транспорту і особливо доступу бездротові технології передачі інформації є найбільш швидко прогресуючими на телекомунікаційному ринку. Особливе місце серед телекомунікаційних мереж різного призначення займають сенсорні мережі, які спрямовані на

забезпечення моніторингу як зосереджених, так і розподілених об’єктів. Сенсорні мережі являють собою специфічну структуру, що забезпечує вирішення задач моніторингу, збирання, зберігання та обробки інформації. У сенсорних мережах використовуються як проводові, так і бездротові технології. Сучасна телекомунікаційна сенсорна мережа (WSN) – бездротова сенсорна мережа, яка складається з просторово розподілених автономних пристроїв з використанням датчиків, які забезпечують загальний контроль фізичних, екологічних чи інших параметрів: температури, звуку, вібрації, тиску, руху, дії забруднення та ін. Сенсорні мережі знайшли широке застосування в промисловості, сільському господарстві, правоохоронних, контролюючих і охоронних структурах. Передбачається стійка тенденція щодо розширення сфери їх застосування, за якістю моніторингу, стійкості, адаптованості. Одне з актуальних застосувань сенсорних систем – це забезпечення моніторингу житлових, господарських та адміністративних приміщень. Це так звана проблема створення «Інтелектуального будинку». В такій системі встановлюються датчики для різних служб «Інтелектуального будинку»: управління світлом, кліматом, домашньою електронікою. Такий приклад мережі наведений на рисунку 1.1.

Рисунок 1.1 – Структурна схема «Інтелектуального будинку»

У сфері телемедицини, медицини катастроф та інших завдань охорони здоров’я сенсорні мережі застосовуються під час проведення дистанційного забезпечення медико-біологічних досліджень, моніторингу основних показників медичної телеметрії (пульс, тиск, температура), контролюванні приймання ліків і т.д., що дозволяє виконувати збір даних і зберігання інформації в деяких просторово-розподілених лабораторіях з наданням дослідникам видаленого доступу до накопичених даних.

Моніторинг місцевості включає в себе систему охорони й контролю доступу, контроль стаціонарних рубежів охорони, контроль маршрутів переміщення людей, раннє виявлення аварій, протипожежну систему. Прикладом побудови таких мереж можуть бути сенсорні мережі, призначені спеціально для охорони нафтогонів, які являють собою сукупність автономних сенсорних модулів, що монтуються над трубопроводом, що охороняється на глибині 50-80 см. Більшість існуючих на даний час сенсорних систем: системи, які забезпечують протипожежний моніторинг, контроль цілісності приміщень і об’єктів,

охоронні системи використовують, як правило, передачу інформаційних сигналів по виділених або незалежних каналах, що виділяються. Загальна структура таких систем (охоронної чи пожежної сигналізації) представлена на рисунку 1.2.

При такій структурі на одному кінці системи встановлюється датчик, який функціонує за системою 1/0, на другому кінці – пульт сигналізації, на якому відображається наявність сигналу 0 чи 1.

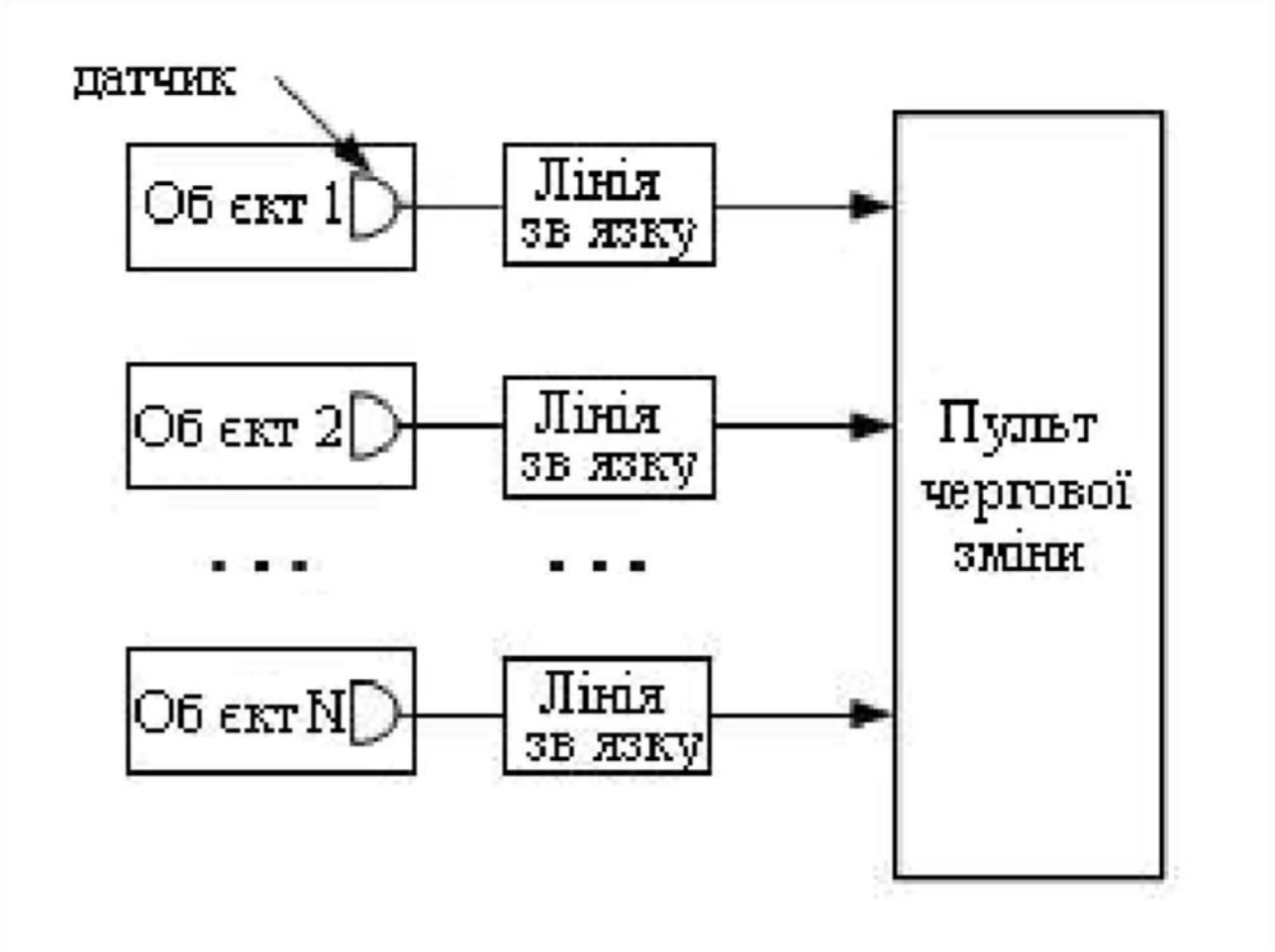


Рисунок 1.2 – Структура системи охоронного і пожежного оповіщення

Як видно, структура пожежної сигналізації проста і дозволяє реагувати черговій зміні на зміну параметрів на об’єктах. Лінія зв’язку виконується шляхом прокладання чи використання існуючого 2-жильного кабелю. Живлення датчика виконується автономним джерелом, яке

знаходиться на об’єкті чи на пульті сигналізації, чи на тому й іншому. Перевагою такої структури є досить висока оперативність, оскільки інформація про зміну 15 параметрів дозволяє швидко реагувати на зміни, які з’являються. До недоліків цієї структури належать: – обмежений обсяг інформації, що передається, часто це 0 або 1, що не дає змогу передавати іншу доречну інформацію (голос, відео, дані), яка дозволяє більш точно оцінити стан об’єкту моніторингу і прийняти обґрунтоване рішення; – необхідність використовувати автономні виділені канали, що здорожує всю систему, а при передачі сигналів по проводах міської телефонної мережі виключає передачу мовних й інших повідомлень під час рішення сенсорних завдань; – низька надійність ліній зв’язку і велика кількість помилкових тривог, які пов’язані зі зміною параметрів лінії, зривом, зниженням ємності і відсутністю живлення; – низька оперативність розгортання і нарощування можливостей та ін. Сенсорні системи і мережі, які зараз знаходяться в експлуатації, будуються, як правило, на основі виділених ліній зв'язку. На одному кінці такої лінії встановлюється датчик, що функціонує за системою 1/0, на другому – пульт сигналізації, де відображається наявність сигналу 0 або 1 (рис. 1.2). За таким принципом побудовані мережі охоронної сигналізації, мережі пожежного сповіщення та ін. Така структура ліній забезпечує мінімальний обсяг інформації, що передається.

Огляд алгоритмів балансування навантаження

Питання про планування навантаження слід вирішувати ще на ранній стадії розвитку будь-якого веб-проекту. «Падіння» сервера (а воно завжди відбувається несподівано, в самий невідповідний момент) загрожує дуже серйозними наслідками - як моральними, так і матеріальними. Спочатку проблеми недостатньої продуктивності сервера в зв'язку зростанням навантажень можна вирішувати шляхом нарощування

потужності сервера, або ж оптимізацією використовуваних алгоритмів, програмних кодів і так далі. Але рано чи пізно настає момент, коли і ці заходи виявляються недостатніми.

Доводиться вдаватися до кластеризації: кілька серверів об'єднуються в кластер; навантаження між ними розподіляється за допомогою комплексу спеціальних методів, які називаються балансуванням. Крім вирішення проблеми високих навантажень кластеризація допомагає також забезпечити резервування серверів один на одного.

Ефективність кластеризації безпосередньо залежить від того, як розподіляється (балансується) навантаження між елементами кластера.

Балансування навантаження може здійснюватися за допомогою як апаратних, так і програмних інструментів. Про основні методи і алгоритми і балансування ми б хотіли розповісти в цій статті.

Процедура балансування здійснюється за допомогою цілого комплексу алгоритмів і методів, відповідним такими рівнями моделі OSI:

- мережевому;
- транспортному;
- прикладного.

Розглянемо ці рівні більш детально.

Балансування на мережевому рівні передбачає вирішення наступного завдання: потрібно зробити так, щоб за один конкретний IP-адреса сервера відповідали різні фізичні машини. Таке балансування може здійснюватися за допомогою безлічі різноманітних способів.

- *DNS-балансування*. На одне доменне ім'я виділяється кілька IP-адрес. Сервер, на який буде спрямований клієнтський запит, зазвичай визначається за допомогою алгоритму Round Robin

(про методи і алгоритми балансування буде детально розказано нижче).

- *Побудова NLB-кластера*. При використанні цього способу сервери об'єднуються в кластер, що складається з вхідних і обчислювальних вузлів. Розподіл навантаження здійснюється за допомогою спеціального алгоритму. Використовується в рішеннях від компанії Microsoft.
- *Балансування по IP з використанням додаткового маршрутизатора.*
- *Балансування за територіальною ознакою* здійснюється шляхом розміщення однакових сервісів з однаковими адресами в територіально різних регіонах Інтернету (так працює технологія Anycast DNS, про яку вже згадувалось раніше). Балансування за територіальною ознакою також використовується в багатьох CDN.

Балансування на транспортному рівні є найпростішим: клієнт звертається до балансувальника, той перенаправляє запит одного з серверів, який і буде його обробляти. Вибір сервера, на якому буде оброблятися запит, може здійснюватися відповідно до самими різними алгоритмами: шляхом простого кругового перебору, шляхом вибору найменш завантаженого сервера з пулу і т.п.

Іноді балансування на транспортному рівні складно відрізнити від балансування на мережевому рівні. Розглянемо наступне правило для мережевого фільтра pf в BSD-системах: так, наприклад, формально тут йдеться про балансування трафіку на конкретному порту TCP (приклад для мережевого фільтра pf в BSD-системах):

На мережевому рівні балансувальник просто вирішує, на який сервер передавати пакети. Сесію з клієнтом здійснює сервер.

На транспортному рівні спілкування з клієнтом замикається на балансувальнику, який працює як проксі. Він взаємодіє з серверами від свого імені, передаючи інформацію про клієнта в додаткових даних і заголовках. Таким чином працює, наприклад, популярний програмний балансувальник HAProxy.

При **балансуванні на прикладному рівні** балансувальник працює в режимі «розумного проксі». Він аналізує клієнтські запити і перенаправляє їх на різні сервери в залежності від характеру запитуваної контенту. Так працює, наприклад, веб-сервер Nginx, розподіляючи запити між фронтенд і бекенд. За балансування в Nginx відповідає модуль Upstream. Більш докладно про особливості балансування Nginx на основі різних алгоритмів можна прочитати, наприклад, тут .

В якості ще одного прикладу інструменту балансування на прикладному рівні можна привести pgpool - проміжний шар між клієнтом і сервером СУБД PostgreSQL. З його допомогою можна розподіляти запити оп серверів баз даних в залежності від їх змісту, наприклад, запити на читання будуть передаватися на один сервер, а запити на запис - на інший.

Існує багато різних алгоритмів і методів балансування навантаження. Вибираючи конкретний алгоритм, потрібно виходити, по-перше, з специфіки конкретного проекту, а по-друге - з цілей. які ми плануємо досягти.

У числі цілей, для досягнення яких використовується балансування, потрібно виділити наступні:

- справедливість : потрібно гарантувати, щоб на обробку кожного запиту виділялися системні ресурси і не допустити виникнення ситуацій, коли один запит обробляється, а всі інші чекають своєї черги;
- ефективність : всі сервери, які обробляють запити, повинні бути зайняті на 100%; бажано не допускати ситуації, коли один з серверів простоює в очікуванні запитів на обробку (відразу ж обмовимося, що в реальній практиці ця мета досягається далеко не завжди);
- скорочення часу виконання запиту : потрібно забезпечити мінімальний час між початком обробки запиту (або його постановкою в чергу на обробку) і його завершення;
- скорочення часу відгуку : потрібно мінімізувати час відповіді на запит користувача.

Рекомендовано, щоб алгоритм балансування мав також такі властивості:

- передбачуваність : потрібно чітко розуміти, в яких ситуаціях і при яких навантаженнях алгоритм буде ефективним для вирішення поставлених завдань;
- рівномірна завантаження ресурсів системи ;
- масштабованість : алгоритм повинен зберігати працездатність при збільшенні навантаження.

Round Robin, або алгоритм кругового обслуговування, являє собою перебір по круговому циклу: перший запит передається одного сервера, потім наступний запит передається іншому і так до досягнення останнього сервера, а потім все починається спочатку.

Найпоширенішою імплементацією цього алгоритму є, звичайно ж, метод балансування Round Robin DNS. Як відомо, будь-який DNS-сервер зберігає пару «ім'я хоста - IP-адреса» для кожної машини в певному домені.

DNS-сервер проходить по всіх записів таблиці і віддає на кожен новий запит наступний IP-адреса: наприклад, на перший запит - xxx.xxx.xxx.2, на другий - xxx.xxx.xxx.3, і так далі. В результаті всі сервери в кластері отримують однакову кількість запитів.

У числі безперечних плюсів цього алгоритму слід назвати, по-перше, незалежність від протоколу високого рівня. Для роботи за алгоритмом Round Robin використовується будь-який протокол, в якому звернення до сервера йде по імені.

Балансування на основі алгоритму Round Robin ніяк не залежить від навантаження на сервер: кешуючі DNS-сервери допоможуть впоратися з будь-яким напливом клієнтів.

Використання алгоритму Round Robin не вимагає зв'язку між серверами, тому він може використовуватися як для локального, так і для глобального балансування.

Нарешті, рішення на базі алгоритму Round Robin відрізняються низькою вартістю: щоб вони почали працювати, досить просто додати кілька записів в DNS.

Алгоритм Round Robin має і цілий ряд істотних недоліків. Щоб розподіл навантаження за цим алгоритмом відповідало згаданим вище критеріями справедливості і ефективності, потрібно, щоб у кожного сервера був в наявності однаковий набір ресурсів. При виконанні всіх операцій також має бути задіяно однакову кількість ресурсів. У реальній практиці ці умови в більшості випадків виявляються нездійсненними.

Також при балансуванні за алгоритмом Round Robin абсолютно не враховується завантаженість того чи іншого сервера в складі кластера. Уявімо собі таку гіпотетичну ситуацію: один з вузлів завантажений на 100%, в той час як інші - всього на 10 - 15%. Алгоритм Round Robin можливості виникнення такої ситуації не враховує в принципі, тому перевантажений вузол все одно буде отримувати запити. Ні про яку справедливість, ефективності та передбачуваності в такому випадку не може бути й мови.

В силу описаних вище обставин сфера застосування алгоритму Round Robin значно обмежена.

Weighted Round Robin – це вдосконалена версія алгоритму Round Robin. Суть удосконалень полягає в наступному: кожного серверу присвоюється ваговий коефіцієнт відповідно до його продуктивністю і потужністю. Це допомагає розподіляти навантаження більш гнучко: сервери з великою вагою обробляють більше запитів. Однак усіх проблем з відказостійкістью це аж ніяк не вирішує. Більш ефективну балансування забезпечують інші методи, в яких при плануванні і розподілі навантаження враховується більшу кількість параметрів.

Назвемо ще один алгоритм, **Least Connections** – в ньому абсолютно не враховується кількість активних на даний момент підключений.

Розглянемо практичний приклад. Є два сервера - позначимо їх умовно як А і Б. До сервера А підключено менше користувачів, ніж до сервера Б. При цьому сервер А виявляється більш перевантаженим. Це можливо завдяки тому, що підключення до сервера А підтримуються протягом довшого часу в порівнянні з підключеннями до сервера Б.

Описану проблему можна вирішити за допомогою алгоритму, відомого під назвою least connections (скорочено - leastconn). Він враховує

кількість підключень, підтримуваних серверами в поточний момент часу. Кожен наступний питання передається сервера з найменшою кількістю активних підключень.

Існує вдосконалений варіант цього алгоритму, призначений в першу чергу для використання в кластерах, що складаються з серверів з різними технічними характеристиками і різною продуктивністю. Він називається *Weighted Least Connections* і враховує при розподілі навантаження не тільки кількість активних підключень, а й ваговий коефіцієнт серверів.

У числі інших вдосконалених варіантів алгоритму Least Connections слід перш за все виділити *Locality-Based Least Connection Scheduling* і *Locality-Based Least Connection Scheduling with Replication Scheduling*.

Перший метод був створений спеціально для кешуючих проксі-серверів. Його суть полягає в наступному: найбільша кількість запитів передається серверам з найменшою кількістю активних підключень. За кожним з клієнтських серверів закріплюється група клієнтських IP. Запити з цих IP направляються на «рідний» сервер, якщо він не завантажений повністю. В іншому випадку запит буде перенаправлено на інший сервер (він повинен бути завантажений менш ніж наполовину).

В алгоритмі *Locality-Based Least Connection Scheduling with Replication Scheduling* кожен IP-адреса або група IP-адрес закріплюється не за окремим сервером, а за цілою групою серверів. Запит передається найменш завантаженому сервера з групи. Якщо ж все сервери з «рідної» групи перевантажені, то буде зарезервований новий сервер. Цей новий сервер буде додано до групи, яка обслуговує IP, з якого був відправлений запит. У свою чергу найбільш завантажений сервер з цієї групи буде знищено - це дозволяє уникнути надмірної реплікації.

Алгоритм **Destination Hash Scheduling** був створений для роботи з кластером кешуючих проксі-серверів, але він часто використовується і в інших випадках. У цьому алгоритмі сервер, який обробляє запит, вибирається з статичної таблиці по IP-адресою одержувача.

Алгоритм **Source Hash Scheduling** базується на тих же самих принципах, що і попередній, тільки сервер, який буде обробляти запит, вибирається з таблиці за IP-адресою відправника.

Sticky Sessions - алгоритм розподілу вхідних запитів, при якому з'єднання передаються на один і той же сервер групи. Він використовується, наприклад, в веб-сервері Nginx. Сесії користувача можуть бути закріплені за конкретним сервером за допомогою методу IP hash (детальну інформацію про нього див. В офіційній документації). За допомогою цього методу запити розподіляються по серверам на основі IP-Адреса клієнта. Як зазначено в документації (див. Посилання вище), «метод гарантує, що запити одного і того ж клієнта буде передаватися на один і той же сервер». Якщо закріплений за конкретною адресою сервер недоступний, запит буде перенаправлено на інший сервер.

Застосування цього методу пов'язане з деякими проблемами. Проблеми з прив'язкою сесій можуть виникнути, якщо клієнт використовує динамічний IP. У ситуації, коли велика кількість запитів проходить через один проксі-сервер, балансування навряд чи можна назвати ефективною і справедливою. Описані проблеми, однак, можна вирішити, використовуючи cookies. У комерційній версії Nginx є спеціальний модуль sticky, який якраз використовує cookies для балансування. Є у нього і безкоштовні аналоги - наприклад, nginx-sticky-module .

1.4. Висновки до розділу

В даному розділі проаналізовано практичні приклади використання безпровідних сенсорних мереж та систем: розглянуті особливості будови деяких систем, проаналізовані проблеми, що виникають при реалізації даних систем. Розглянуто технології та рекомендації щодо ефективного впровадження безпровідних сенсорних мереж у різні сфери життєдіяльності.

Проведено аналіз та порівняння існуючих алгоритмів балансування навантаження. Розглянуто функціонал, який вони надають, проаналізовано їх недоліки та переваги, що дозволяє знайти шляхи для вдосконалення цих систем.

Розроблено технічне завдання з прописаними вимогами, що дозволяє почати проектування та реалізацію ПЗ для моделювання мережі.

2. ГІПОТЕЗА ТА ПОБУДОВА МОДЕЛІ

2.1. Аналіз предметної області

Предметною областю даної роботи є поверхні водойм або невеликих сільськогосподарських угідь малого розміру. Розберемо окремо приклади застосування мережі сенсорів у даних сферах.

За допомогою запропонованої мережі сенсорів можливо проводити сканування водойми на якість води.

Якість води — характеристика, що визначає придатність води для певного способу її використання. Залежності від мети використання, вимоги, що висуваються до якості води можуть бути різними і базуються, насамперед, на якісному та кількісному складі речовин, що містяться у воді. Існують нормативні документи, за якими оцінюється придатність води для різних цілей:

- централізованого комунально-питного водопостачання;
- технічного водопостачання;
- рекреації;
- рибного господарства;
- зрошення;
- тощо.

Для визначення приналежності води певному класу якості використовуються сенсори, що визначають наступні параметри:

- концентрацію біогенних елементів;
- насиченість киснем;
- температуру;
- тощо.

Для отримання повних даних необхідно зробити перевірку на різних глибинах.

Для проведення цього дослідження на поверхні водойми розташовують датчики, для збору різних її параметрів на різних глибинах. Ці датчики об`єднуються у мережу, що надсилають дані на базову станцію. Для такої мережі властиві переважно рівномірне розподілення датчиків на поверхні та незмінність положення у просторі (статичність). Тобто, датчики один за одним розташовуються на поверхні водойми, поступово включаючись у мережу. Вони під'єднуються до сусідніх вузлів, враховуючи алгоритм балансування навантаження. Протягом роботи мережі її структура може змінюватися, як того вимагає алгоритм балансування, для досягнення максимального часу її роботи. Після завершення роботи датчики, що ще мають запас електроенергії вимикаються та забираються з поверхні водойми.

2.2. Опис структури мережі

Як було сказано вище, мережа датчиків розрахована на водойму до 800 метрів завдовжки.

Використовується 3 типи вузлів:

- слабкий вузол (надалі СВ)
- потужний вузол (надалі ПВ)
- базова станція (БС)

Сильні вузли використовуються як концентратори трафіку. Вони є краще за апаратними характеристиками. Основні з них, що враховуються для балансування трафіку в даній мережі: запас електроенергії та кількість каналів радіозв`язку.

Площа водойми розподіляється на квадрати (кола певного радіусу, що перетинаються). Вузли в цьому колі утворять кластер. Якщо вузол не може бути доданий до кластера (з причини нестачі каналів радіозв`язку, недосяжності вузлів через технічні збої, тощо), він приєднається до іншого кластера, або напряму до базової станції. Вузли в загальному випадку можна розташовувати довільно. Але для даної задачі необхідно В кожний квадрат поміщується один чи декілька сильних вузлів та декілька слабких вузлів.

2.3. Опис апаратної реалізації вузлів мережі

Для реалізації вузлів мережі використовуються мікросхеми MPR400CB (рис. 2.1), MPR410CB і комплект розробки сенсорних мереж MICA2 (рис. 2.2).

Загальна характеристика комплекту:

- платформа для комерційної розробки;
- мікросхеми процесорних пристроїв та радіопередавачів-приймальників з надійним мережевим зв’язком;
- мультифункійональні плати, зручний зовнішній сенсорний інтерфейс, послідовні порти та виходи Ethernet;
- постачається з Windows додатком для створення топології мережі, доступу до сенсорних даних, аналізу поточних даних та історії спостереження;
- інструменти для налагодження та розробки, включаючи модуль радіозв’язку та всі його компоненти.

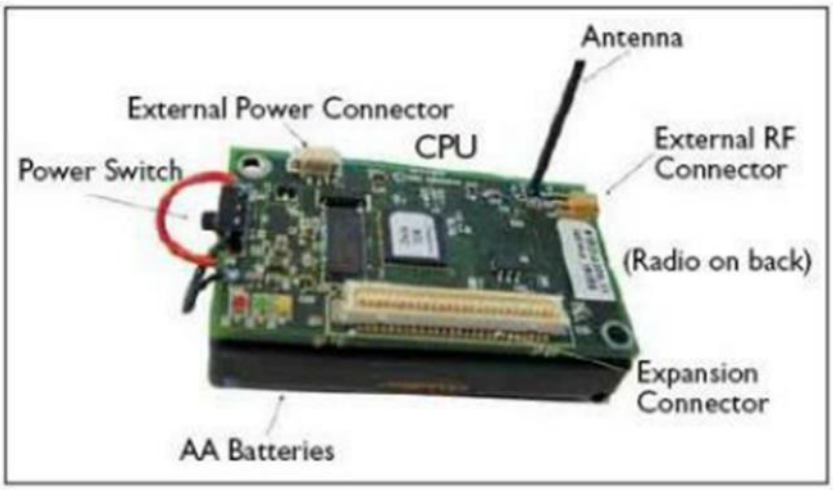


Рисунок 2.1 – Мікросхема MPR400CB

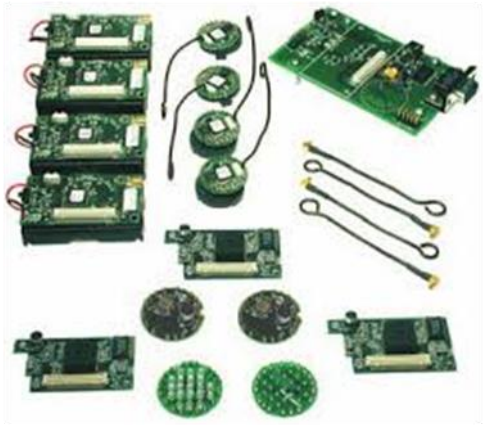


Рисунок 2.2 – Комплект MICA2

Апаратні засоби даної лінійки виявили свої найкращі якості за останні десять років, що також стало причиною вибору даного комплекту для реалізації даної мережі. За допомогою наборів MICA було побудовано безліч сенсорних мереж, що знайшли застосування в наступних галузях:

- виробництво;
- безпека;
- здоров'я;
- фізика;
- дослідження місцевості;
- інші.

Базова станція дозволяє агрегування даних сенсорної мережі на ПК чи іншу обчислювальну машину. Будь-який вузол, побудований на платформі MICA2 (MPR400CB, MPR410CB) може виконувати функції базової станції. Щоб цього досягти, необхідно з'єднати необхідну кількість відповідних мікросхем за допомогою базового мікросхеми-інтерфейсу (надалі БМС) MIB500CA. У БМК передбачено послідовний інтерфейс RS-232, а також паралельні порти для програмування мотів.

Моут MICA2 (рис 2.3) э представником модулів третього покоління, що використовуються для побудови невимогливих до електроенергії бездротових сенсорних мереж. Моут MICA2 має переваги над звичайним MICA моут. Завдяки наступні якостям він краще підходить для комерційної розробки:

- 868/916MHz мультиканальний приймальник-передавач;
- TinyOS (TOS) Distributed Software Operating System v1.0 з покращеним мережевим стеком та інструментами налагодження;
- підтримка бездротового віддаленого перепрограмування;
- широкий спектр мікросхем сенсорів та додаткових чипів;

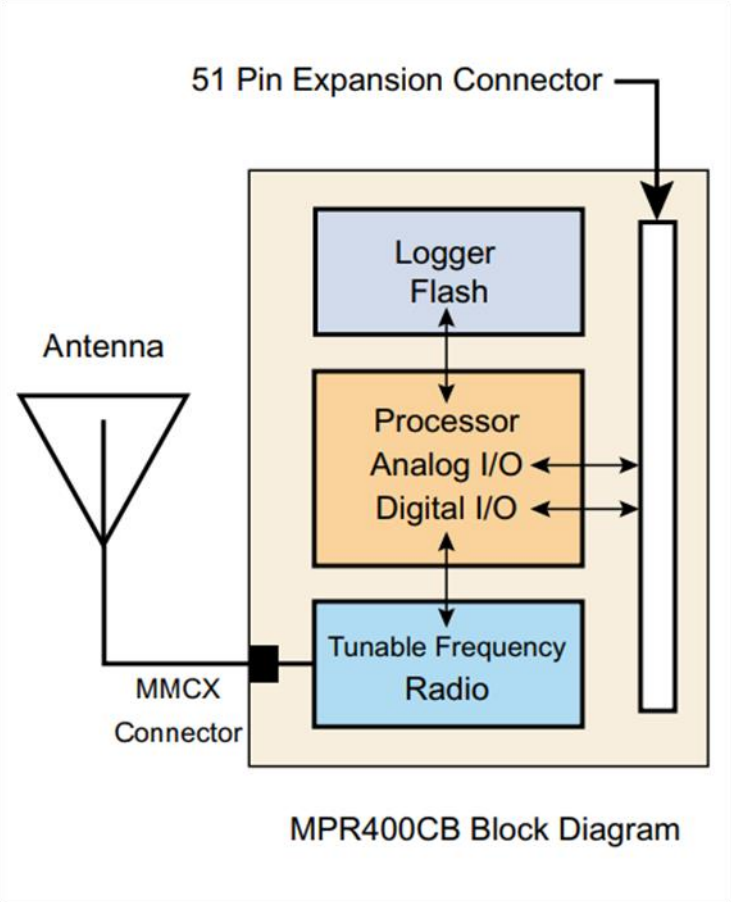


Рисунок 2.3 – Блокова діаграма моута MPR400CB

Моут працює під операційною системою Tiny OS.

TinyOS 1.0 — це легка, енергоефективна операційна система з відкритим вихідним кодом, що розробляється UC Berkeley та має вбудовану підтримку інструментів для побудови маштабованих самоналаштовуваних сенсорних мереж.

2.4. Опис алгоритму балансування навантаження

Для балансування навантаження в даній мережі використовується поєднання двох алгоритмів: LEACH (для об’єднання вузлів в кластери та визначення лідера в кожному з них) та алгоритму Дейкстри пошуку найкоротшого шляху. Найкоротший шлях будується від базової станції до всіх лідерів кластерів. Це пояснюється тим, що споживання енергії (в максимальному випадку) для обробки даних (до 20 %) значно менше аніж пересилання даних (до 60 %).

LEACH є алгоритмом ієрархічної маршрутизації за кластерною схемою. Ієрархічні протоколи створені для зменшення споживання енергії засобами агрегації даних для зменшення трафіку даних між вузлами та базовою станцією. Робота протоколу відбувається в декілька раундів з двома фазами на кожному раунді. LEACH використовує раунд як одиницю планування роботи. Кожен раунд складається з двох стадій:

- налаштування (set-up phase);
- стабільна (steady phase).

Протягом **фази налаштування** головною задачею є створити кластер та вибрати лідера для кожного кластера, вибираючи вузол з максимальним запасом енергії. Для даної мережі зроблена модифікація: вибирається вузол з максимальним запасом енергії серед ПВ. Ця фаза має три фундаментальні кроки:

1) Обирання лідера;
2) Налаштування кластера;
3) Створення таблиці поділу часу.

Протягом першого кроку вузли вирішують чи будуть вони головами кластерів за допомогою спеціальної формули (рис. 2.4), де T(n) — порогове значення. Вузол стає лідером на поточний раунд коли число менше за порогове. Коли вузол обраний лідером, він не може стати лідером знову доки всі потужні вузли не побудуть у ролі лідера один раз. Це покращує споживання енергії.

$$T(n) = \frac{P}{1 - P \times (r \bmod P^{-1})} \qquad \forall n \in G$$

$$T(n) = 0 \qquad \forall n \notin G$$

Where n is a random number between 0 and 1
P is the cluster-head probability and
G is the set of nodes that weren't cluster-heads the previous rounds

Рисунок 2.4 – Формула визначення голів кластерів

На другому кроці, вузли, що не є головами кластера отримують запит на приєднання до лідера та відправляють відповідь про підтвердження приєднання. Всі вузли кластера економлять велику частину енергії через те, що вони тримають передавачі вимкненими та вмикають їх тільки тоді, коли мають що передати голові кластеру.

На третьому етапі кожен лідер створює таблицю розподілу часу. TDMA (Time division multiple access) таблиця створюється відповідно до кількості вузлів у кластері. Кожен вузол передає дані у відведений для нього час.

Протягом **стабільної фази** вузол відправляє дані тому, до кого він приєднаний.

Алгоритм Дейкстри (англ. Dijkstra's algorithm) - алгоритм на графах, винайдений нідерландським вченим Е. Дейкстрой в 1959 році. Знаходить найкоротшу відстань від однієї з вершин графа до всіх інших. Алгоритм

працює тільки для графів без ребер негативного ваги. Алгоритм широко застосовується в програмуванні і технологіях.

В орієнтованому зваженому графі G = (V, E), вага ребер якого невід'ємна і визначається ваговою функцією w: E> R, алгоритм Дейкстри знаходить довжини найкоротших шляхів із заданої вершини s до всіх інших.

Кожній вершині з V зіставимо мітку - мінімальне відоме відстань від цієї вершини до заданої, позначимо її a. Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки. Робота алгоритму завершується, коли всі вершини відвідані.

Мітка самої вершини a покладається рівною 0, мітки інших вершин - нескінченності. Це відображає те, що відстані від a до інших вершин поки невідомі. Всі вершини графа позначаються що не відвідані.

Графічна інтерпретація алгоритму представлена у вигляді граф-схеми в додатку.

2.5. Теоретичні обґрунтування роботи моделі

Специфікація комплекту MICA-2 вказує, що на модуль прийому-передачі припадає до 60% відсотків споживаної енергії, на CPU — до 20%, 20% розподіляються між іншими модулями. Також мікросхеми дозволяють програмно встановлювати потужність випромінення сигналу.

Ємність батареї MPR400CB складає 3500 mAh, MPR410CB — 7000 mAh.

За допомогою наступної формули (Рис. 2.5), що запропонована в дослідженні "The Power of Sensitivity Improving range with receiver sensitivity", David Steed, Jr., MaxStream вираховується потужність,

необхідна для передачі даних на задану відстань між вузлами, і, відповідно, споживаний заряд батареї.

$$R = \sqrt[N]{\frac{P_T G_T \lambda^2}{P_R F_M 16\pi^2}}$$

R = *Maximum range for communication link*
N = *Propagation Law (N=2 for line-of-sight, N=4 for urban environments)*
$\mathbf{P_T}$ = *Transmit power*
$\mathbf{G_T}$ = *Total antenna gain*
λ = *Wavelength*
$\mathbf{P_R}$ = *Receiver sensitivity*
$\mathbf{F_M}$ = *Fading margin*

Рисунок 2.5 – Формула визначення потужності, необхідної для передачі даних на задану відстань між вузлами

Тести проводитимемо на для мережі з наступними параметрами.

- Ділянка - 100x200.
- Очікуваний час роботи — 12 годин.
- Дистанція між датчиками — 10 метрів.

Задача полягає у правильному встановленні ПВ, що збалансують трафік у мережі.

2.6. Висновки до розділу

В даному розділі було проаналізовано предметну галузь та обрано алгоритм балансування навантаження. При підборі алгоритму навантаження враховувалися характеристики апаратних засобів, на яких працює мережа, а також особливості предметної галузі.

3. ТЕСТУВАННЯ СИСТЕМИ

3.1. Опис випадків тестування

В результаті першого тесту з вхідними даними проілюстрованими на рисунку 3.1 було отримано графічний результат, що подано на рисунку 3.2 та наступний текстовий вивід в консолі операційної системи:

- Кількість вузлів: 89;
- Перше від’єднання вузла: 3 год. 16хв;
- Час життя мережі: 4 год. 13хв..

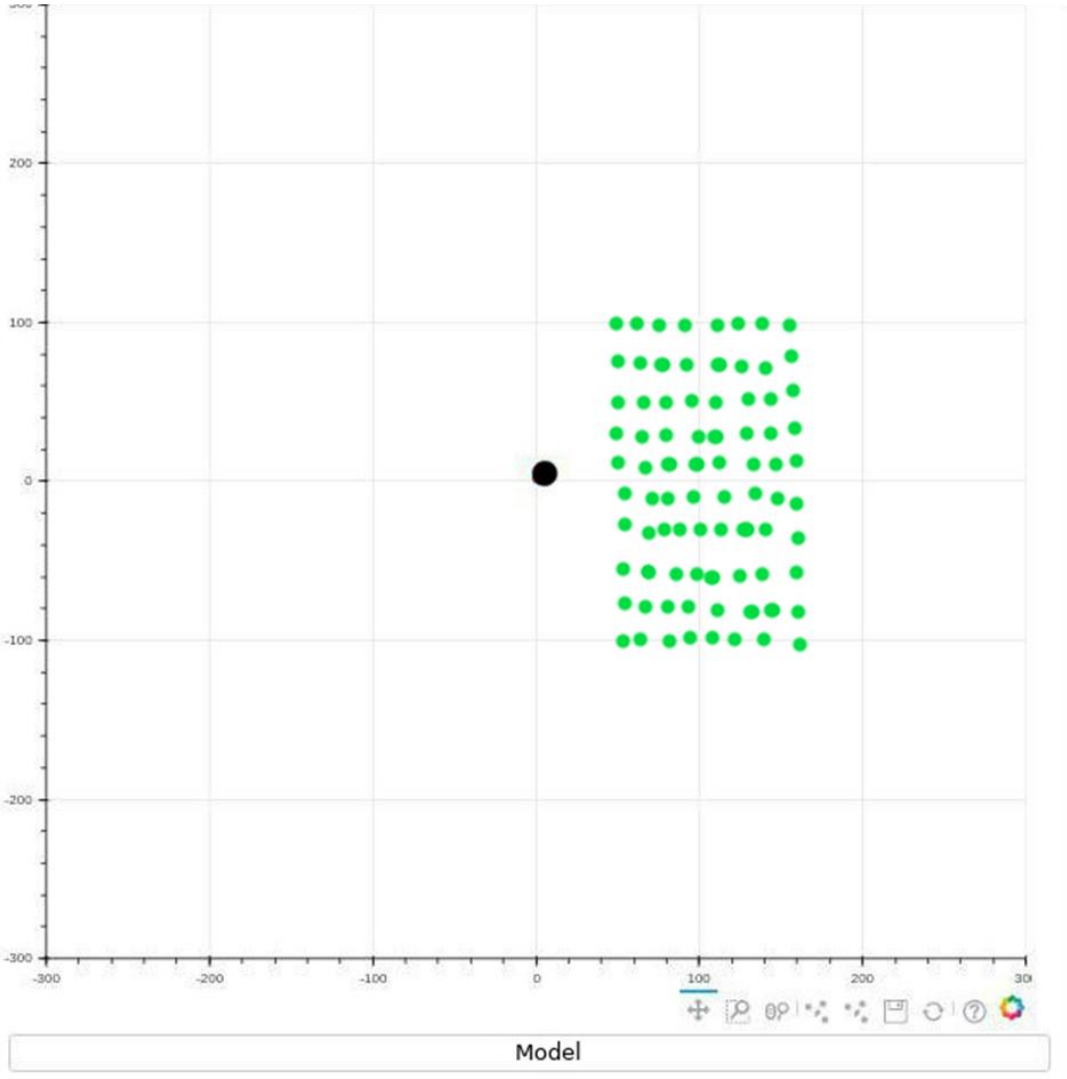


Рисунок 3.1 – Вхідні дані першого тесту

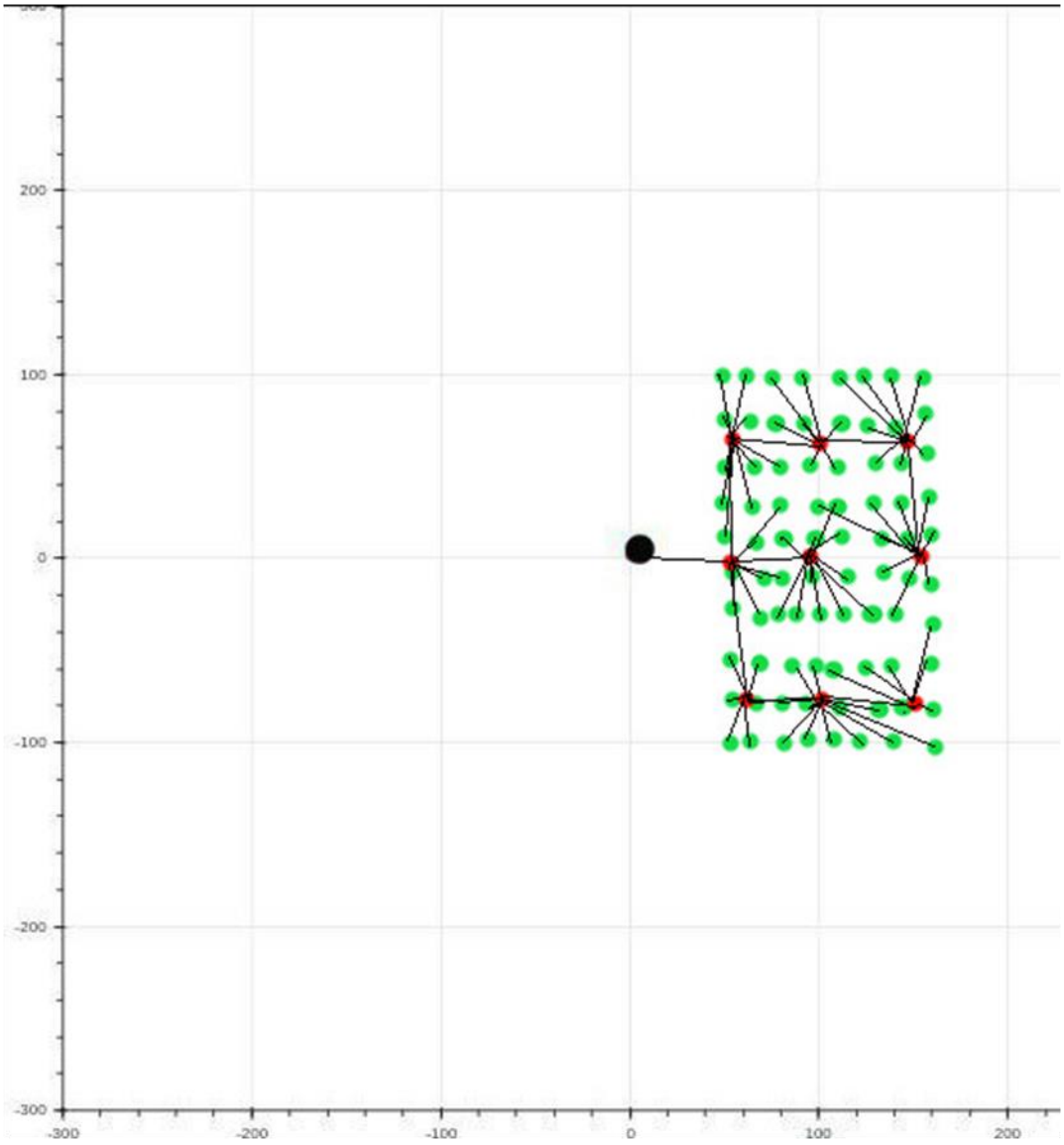


Рисунок 3.2 – Результат першого тесту

В результаті другого тесту з вхідними даними проілюстрованими на рисунку 3.3 було отримано графічний результат, що подано на рисунку 3.4 та наступний текстовий вивід в консолі операційної системи:

- Кількість вузлів: 98;
- Перше від’єднання вузла: 3 год. 21хв;
- Час життя мережі: 7 год. 10хв.

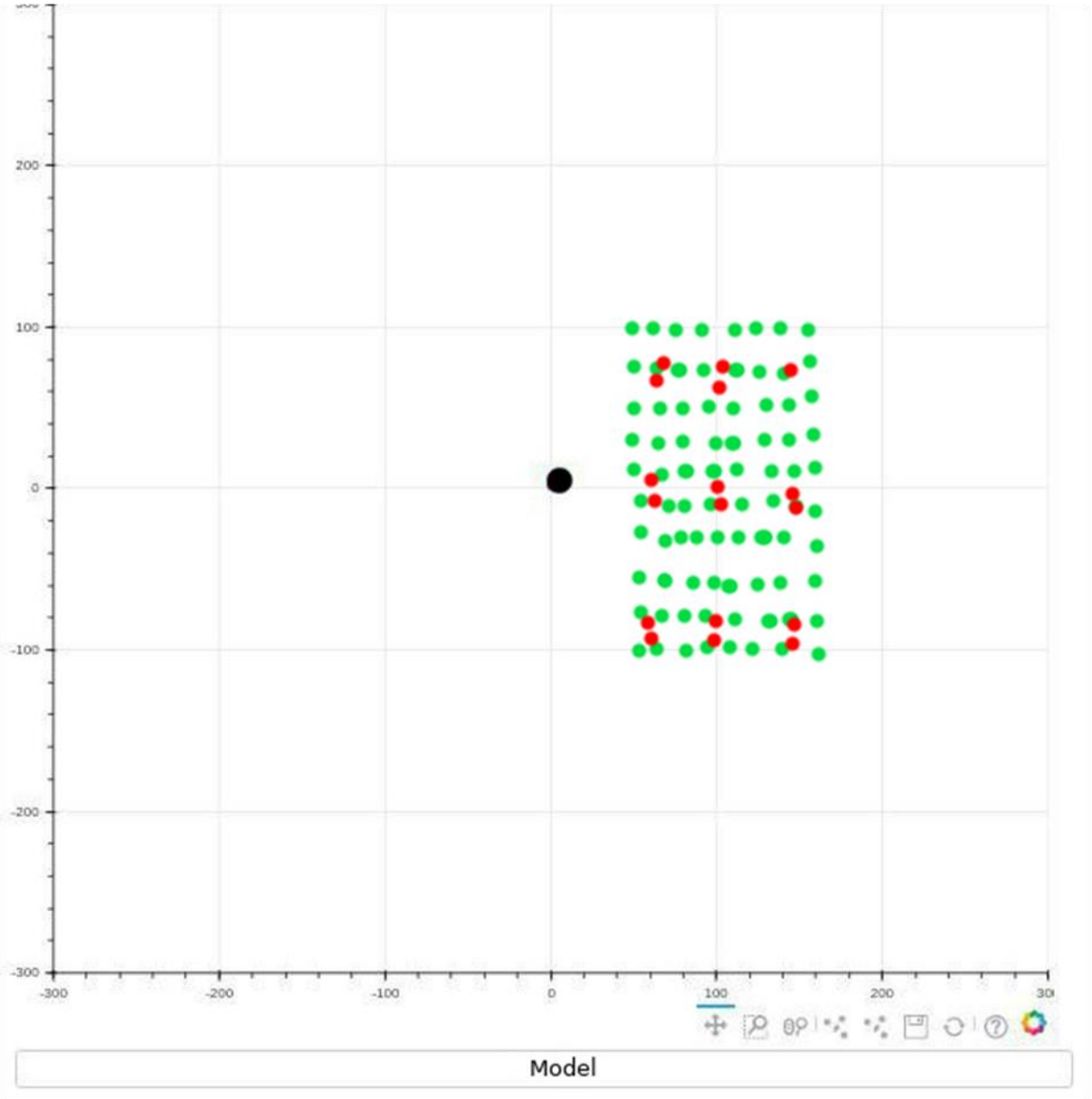


Рисунок 3.3 – Вхідні дані другого тесту

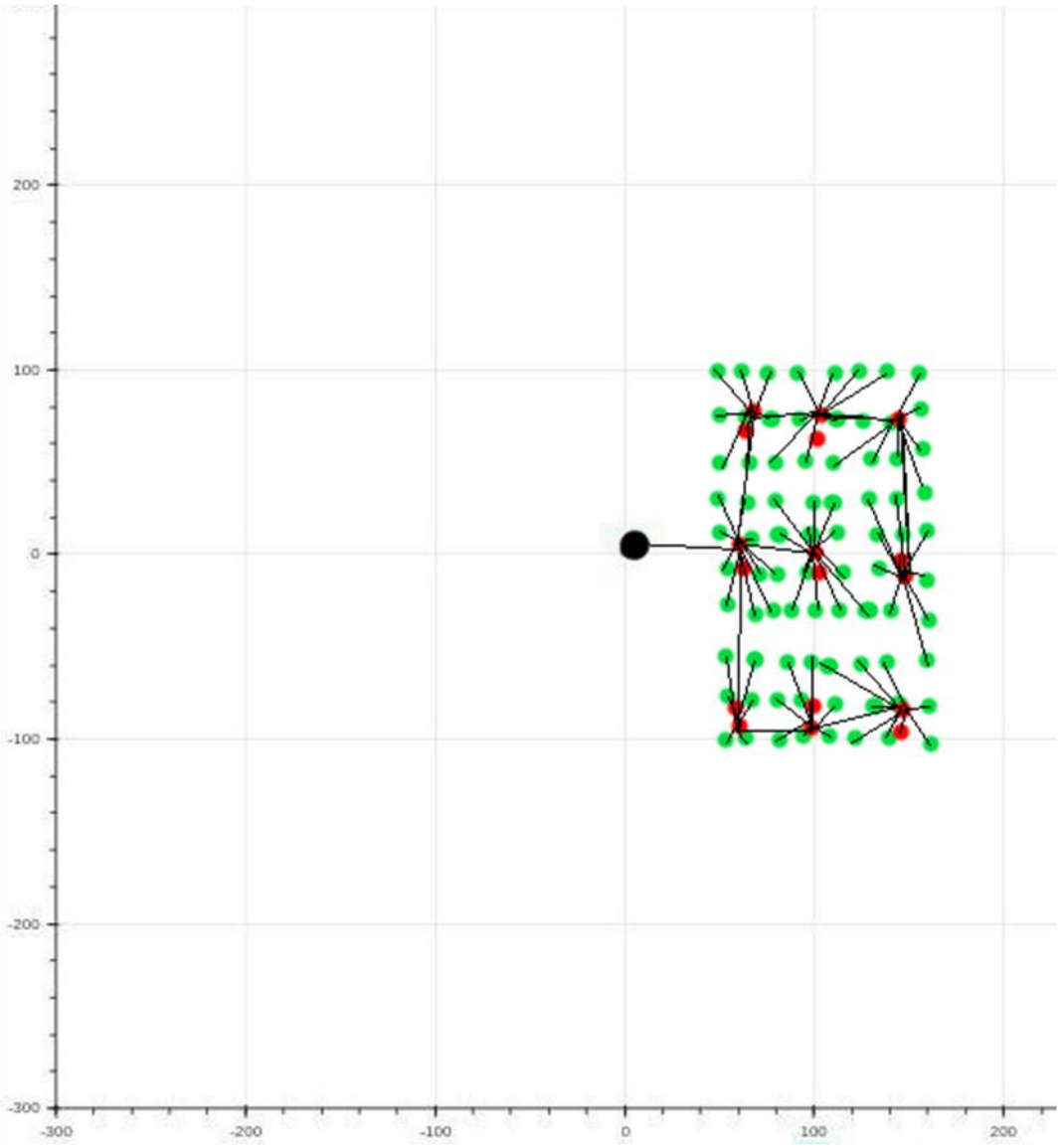


Рисунок 3.4 – Результат другого тесту

В результаті третього тесту з вхідними даними проілюстрованими на рисунку 3.5 було отримано графічний результат, що подано на рисунку 3.6 та наступний текстовий вивід в консолі операційної системи:

- Кількість вузлів: 116;
- Перше від’єднання вузла: 3 год. 2хв;
- Час життя мережі: 8 год. 31хв.

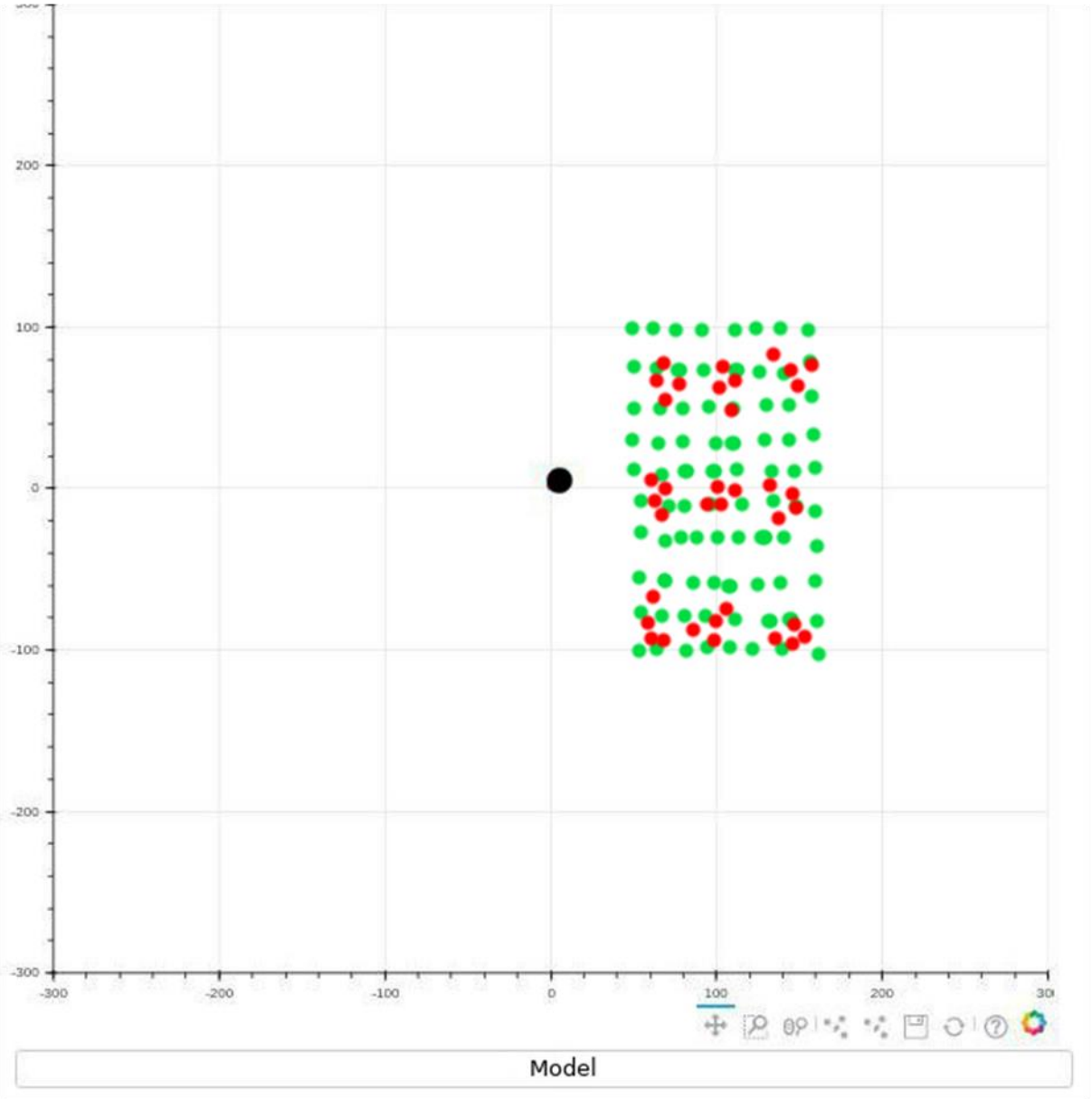


Рисунок 3.5 – Вхідні дані третього тесту

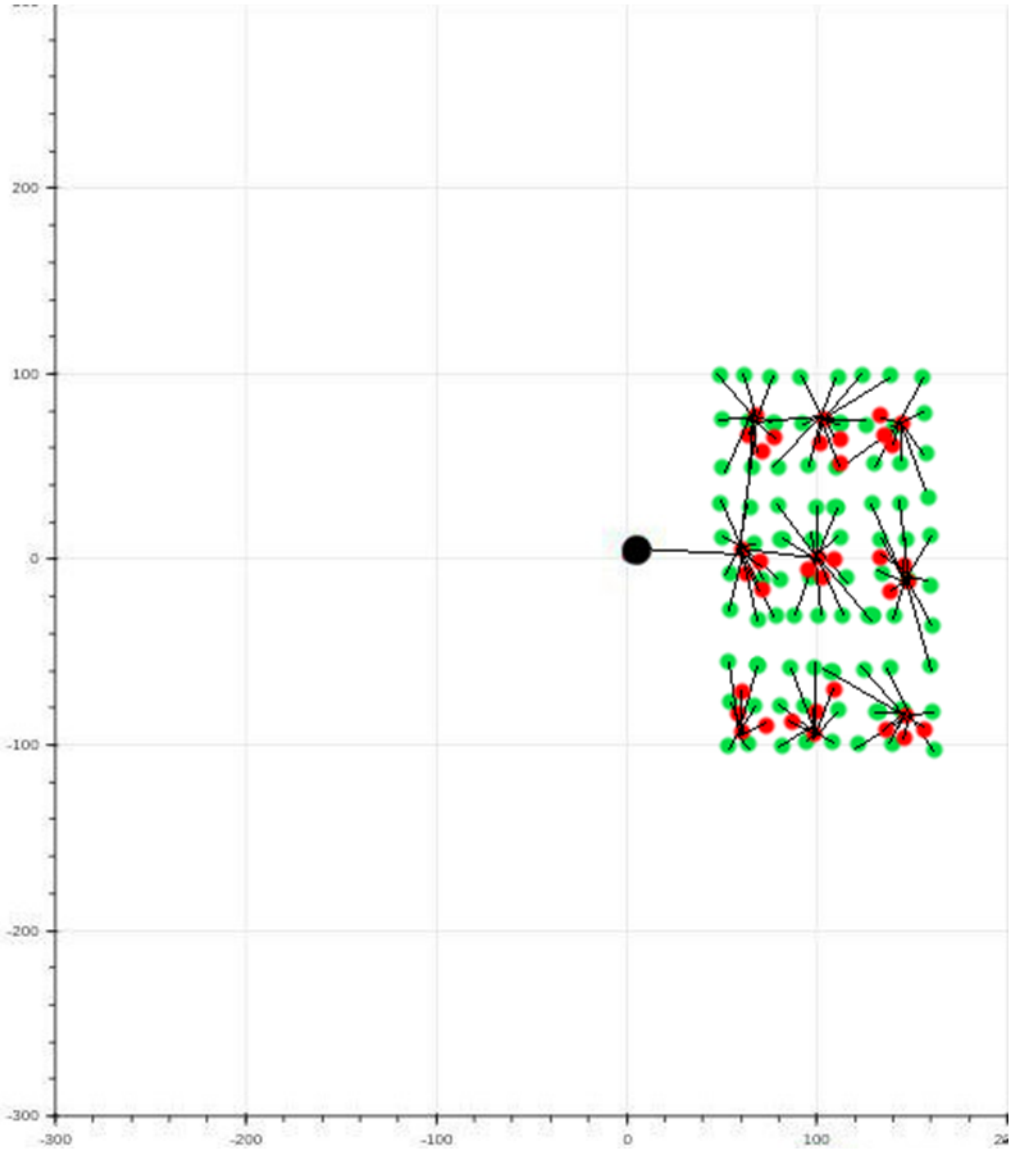


Рисунок 3.6 – Результат третього тесту

В результаті четвертого тесту з вхідними даними проілюстрованими на рисунку 3.7 було отримано графічний результат, що подано на рисунку 3.8 та наступний текстовий вивід в консолі операційної системи:

- Кількість вузлів: 125;
- Перше від’єднання вузла: 2 год. 57хв;
- Час життя мережі: 8 год. 36хв.

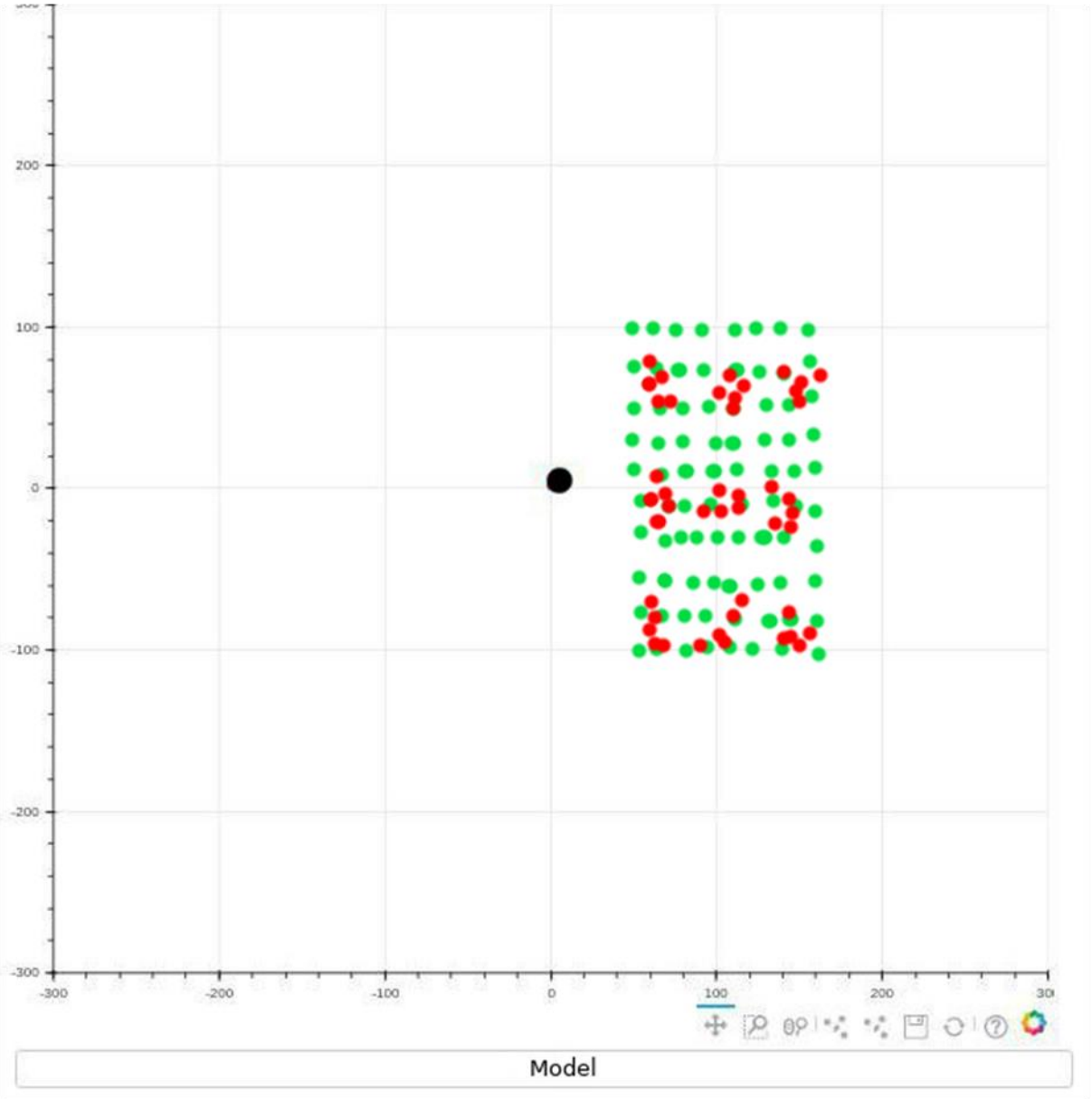


Рисунок 3.7 – Вхідні дані четвертого тесту

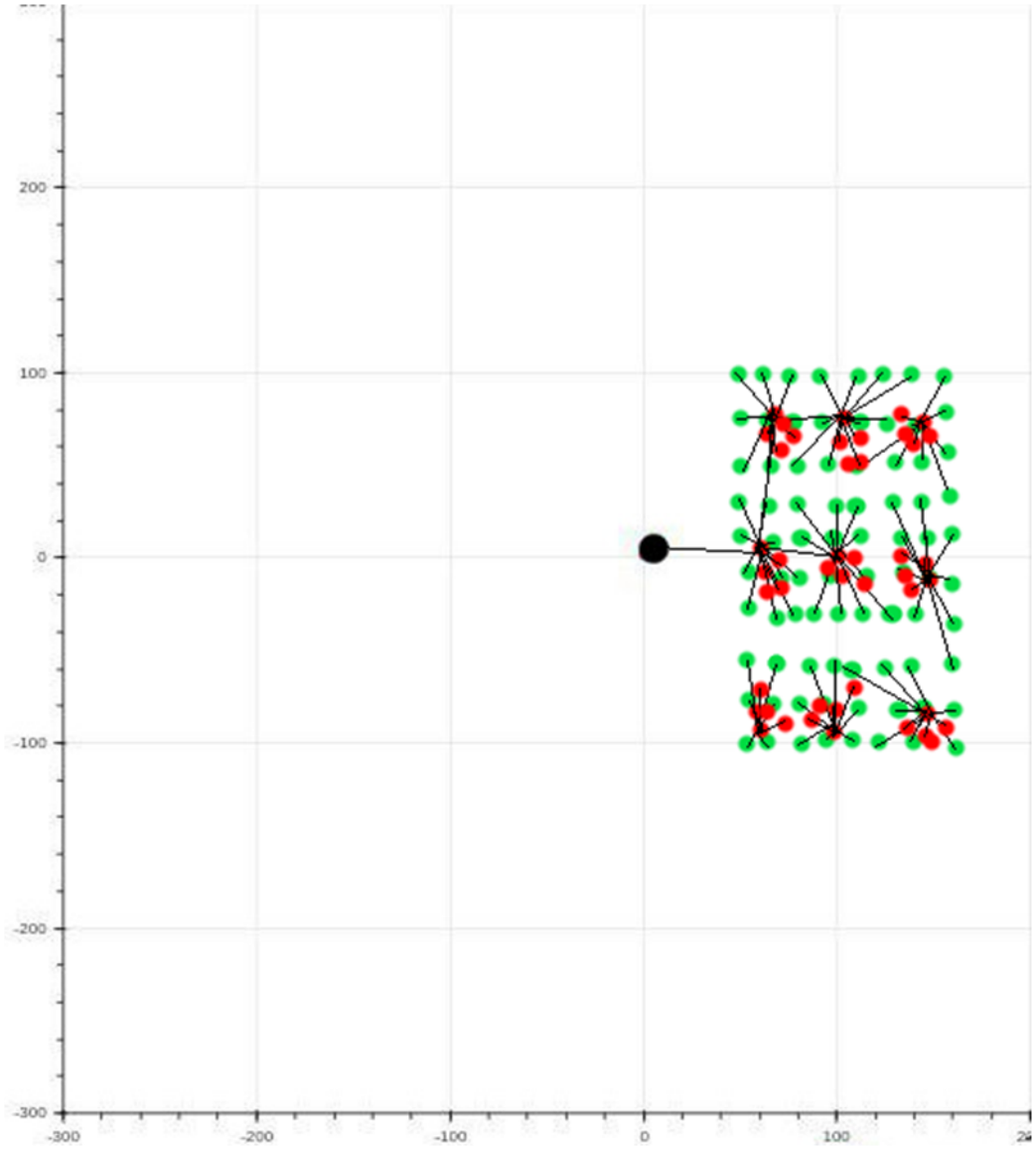


Рисунок 3.8 – Результат четвертого тесту

В результаті п’ятого тесту з вхідними даними проілюстрованими на рисунку 3.9 було отримано графічний результат, що подано на рисунку 3.10 та наступний текстовий вивід в консолі операційної системи:

- Кількість вузлів: 102;
- Перше від’єднання вузла: 3 год. 4хв;
- Час життя мережі: 12 год. 15хв.

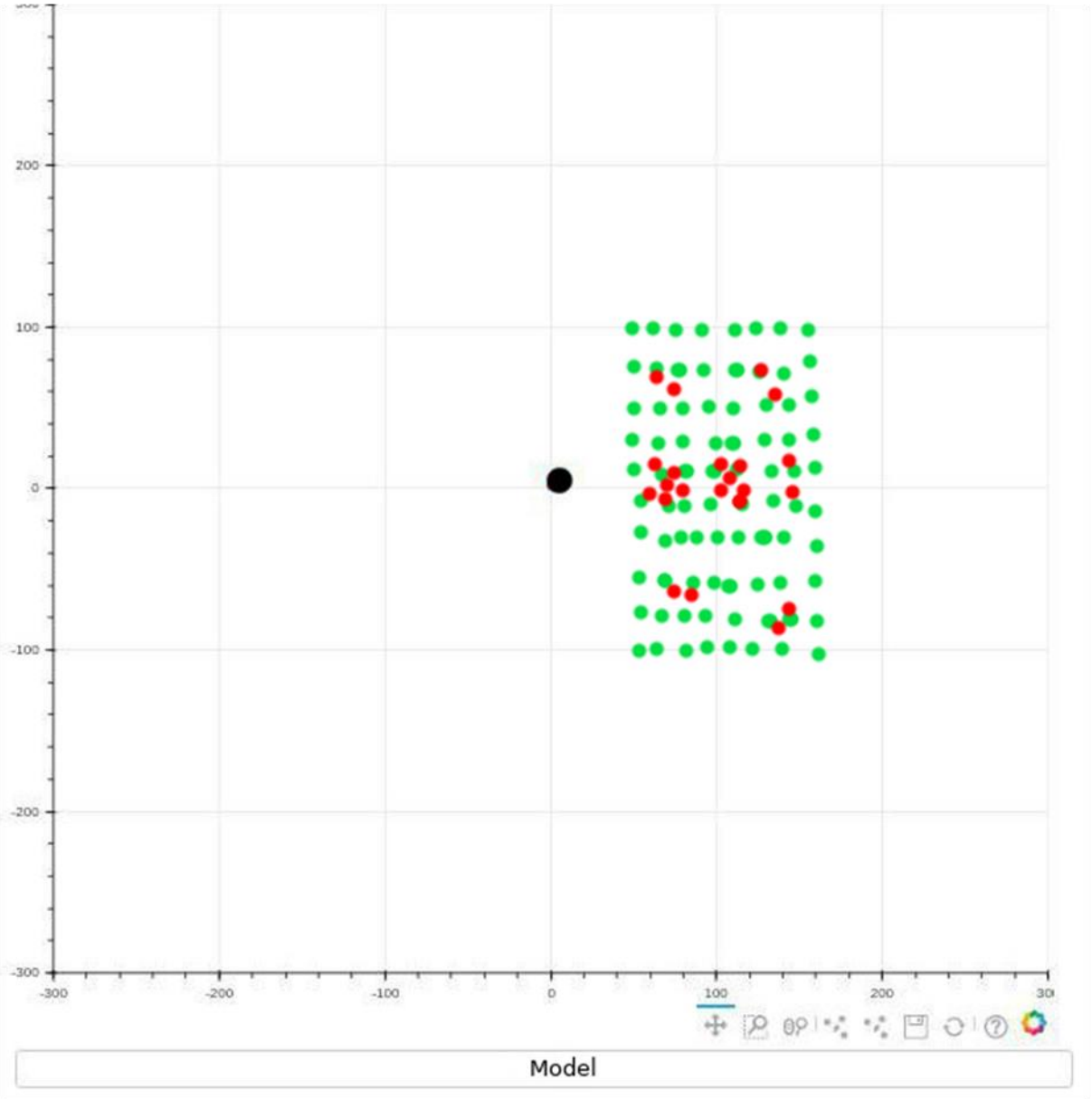


Рисунок 3.9 – Вхідні дані п’ятого тесту

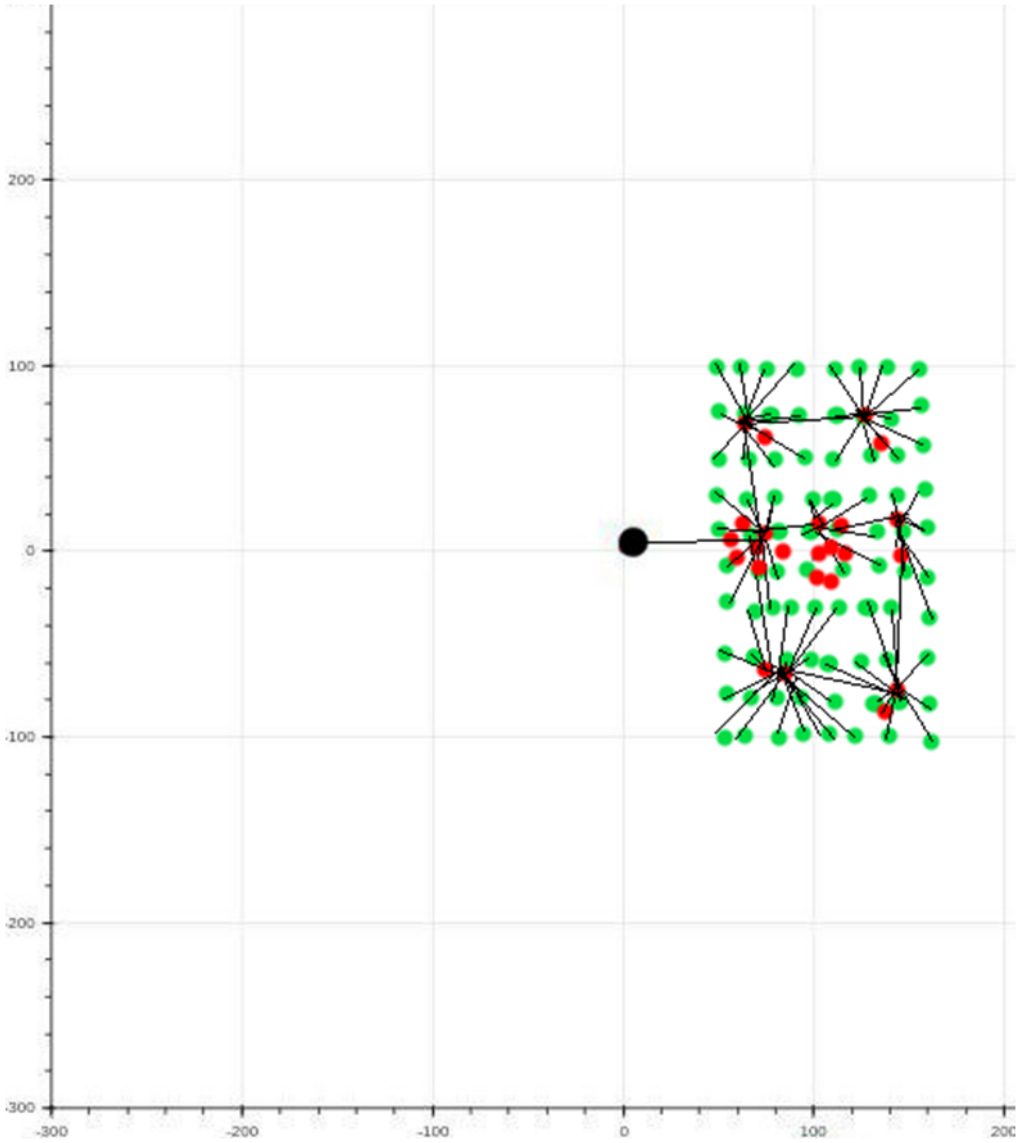


Рисунок 3.10 – Результат п’ятого тесту

3.2. Аналіз результатів

Як видно з тестів 1 та 2 при додаванні додаткового ПВ у кожний кластер збільшує час життя мережі у 1.7 раз.

Порівнюючи тести 2 та 3, можна дійти висновку про те, що додавання ще двох ПВ у кожний кластер збільшує загальний час життя мережі.

Порівнюючи тести 3 та 4 можна дійти висновку про те, що додавання ще одного ПВ у кожний кластер не впливає суттєво на загальний час життя мережі.

Дослідним шляхом було встановлена конфігурація, що є оптимальною для даної мережі. В ній мається 2 кластера з шістьома ПВ, що стоять в найбільш навантаженій частині мережі та 7 кластерів з 2 ПВ.

3.3. Висновки до розділу

В даному розділі було описано хід проведення експериментів, наведно їх аналіз та обрана оптимальна конфігурація для даної мережі.

4. ТЕХНІЧНА РЕАЛІЗАЦІЯ МОДЕЛІ

4.1. Інтерфейс додатка

На рисунку 4.1 показано інтерфейс програми. Він складається з поля для побудови мережі, кнопки “Model” та наступних компонентів:

1) переміщення поля;
2) масштабування;
3) виставляння слабких вузлів;
4) виставляння сильних вузлів;
5) збереження зображення;
6) справка.

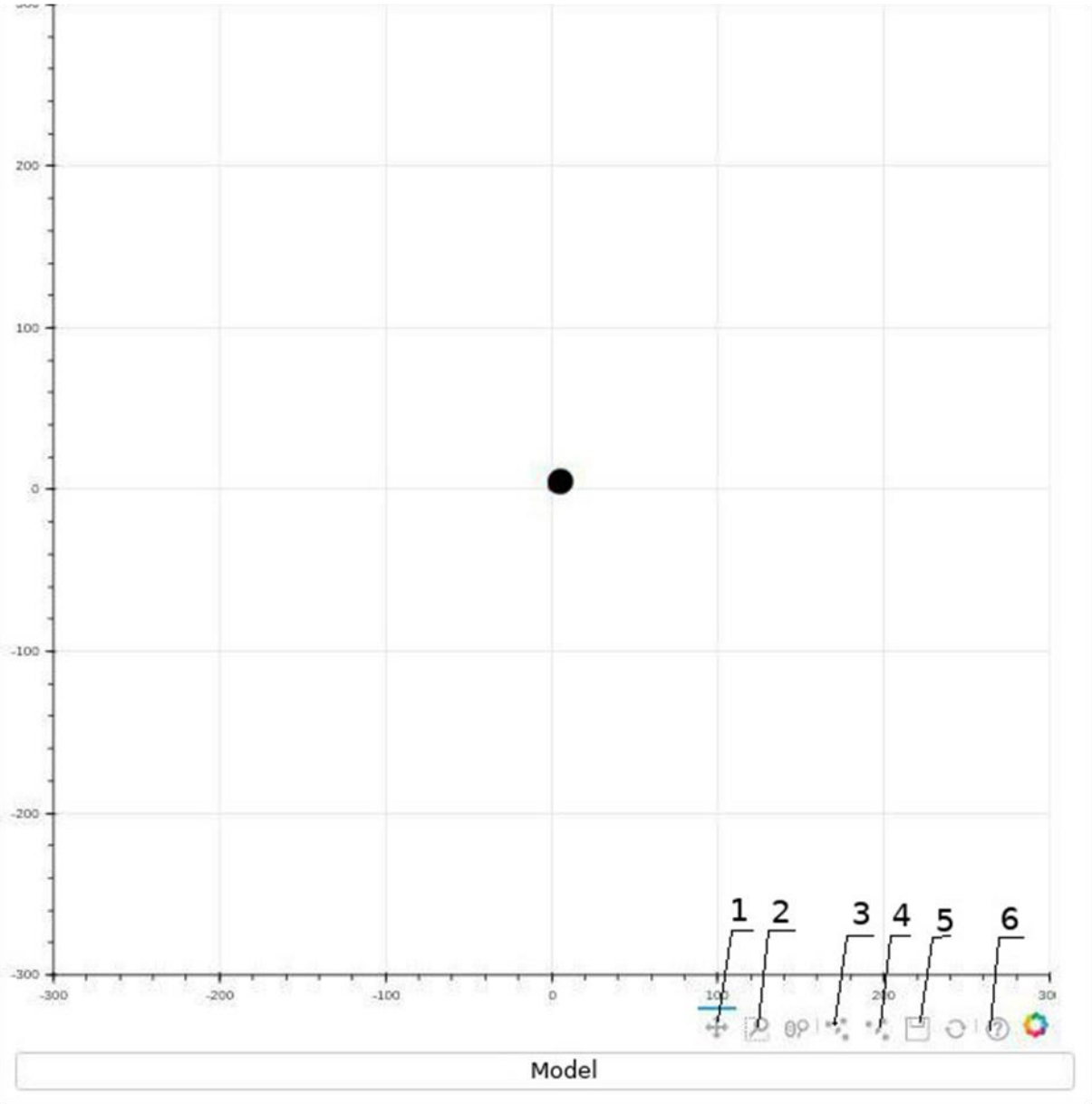


Рисунок 4.1 – Інтерфейс програми

При запуску додатка з`являється сторінка (рис. 4.1).

Побудова мережі складається з наступних кроків:

1) натискання кнопки мишки при розміщенні курсору на місце, де має розташовуватися базова станція;
2) натискання кнопки мишки при розміщенні курсору на кнопку компонента 3 та розташуйте слабкі вузли;

3) натискання кнопки мишки при розміщенні курсору на кнопку компонента 4 та розташуйте потужні вузли;
4) натискання кнопки мишки при розміщенні курсору на кнопку “Model”.

В новому вікні відкриється зображення первинної структури мережі, що побудована завдяки алгоритму балансування та через деякий час в консолі з’явиться результат моделювання (Рис 4.2).

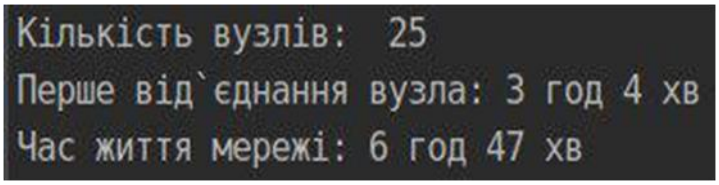


Рисунок 4.2 – Результат моделювання в консолі

В цілому цей процес можна записати в три кроки:

1) задати вузли мережі (Рис 4.3);
2) отримати побудовану мережу (Рис 4.4);
3) отримати результат моделювання в консолі (Рис 4.2).

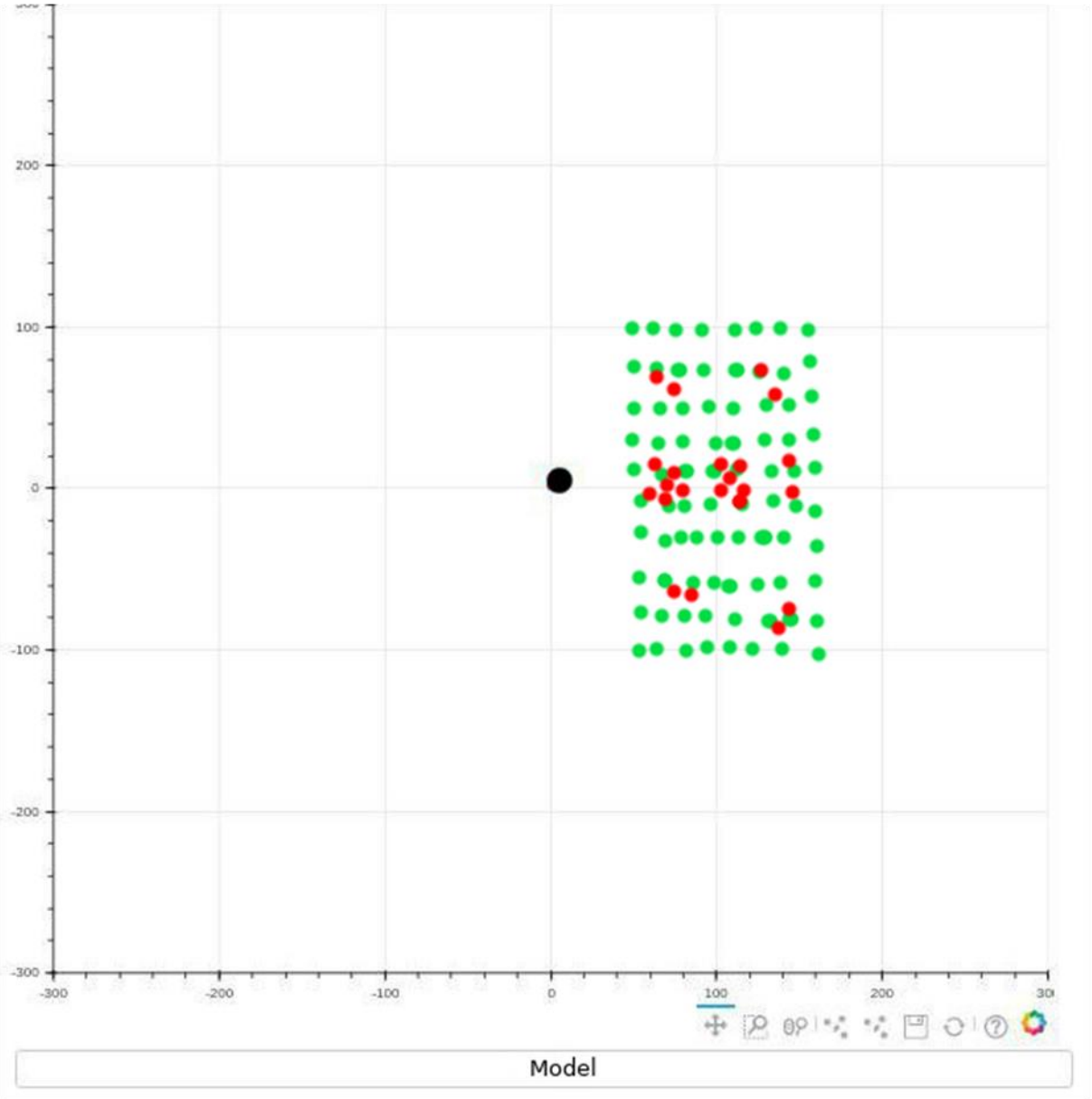


Рисунок 4.3 – Задавання вузлів мережі

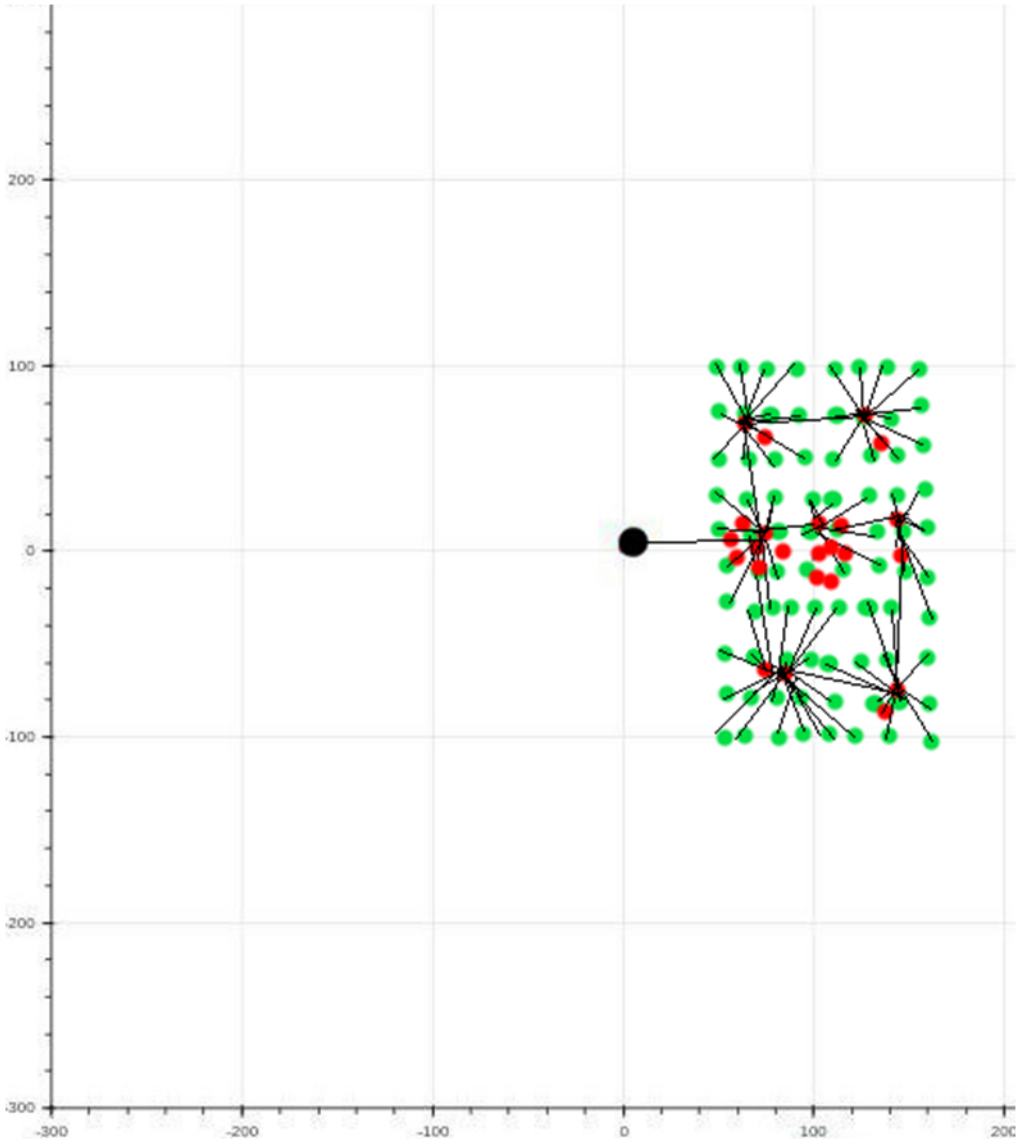


Рисунок 4.4 – Результат моделювання мережі

4.2. Обґрунтування вибору технологій

В процесі розробки моделі було використано технологію Python 3 та фреймворк Bokeh. Дані технології дозволяють швидко та гнучко створювати веб додатки для аналізу чи збору інформації у вигляді графіків, діаграм, тощо. В нашому випадку вони дозволяють створити мапу, розташувати на ній вручну вузли та показати результат роботи.

4.3. Опис використаних програмних модулів та компонентів

Проект складається з чотирьох пакетів:

- network;
- routing;
- sleep_scheduling;
- utils.

Network містить основну частину класів програми. Він має реалізацію класів, що дозволяють будувати мережу: Node, Connection, EnergySource, AggregationModel, Network.

Routing містить алгоритм балансування навантаження та реалізацію передачі даних між вузлами.

Sleep Scheduling містить певну логіку, пов’язану з переходом вузлів у сплячий режим.

Utils містить різні споміжні класи та функції.

4.4. Висновки до розділу

В даному розділі описано процес роботи додатку, з яких компонентів він складається та як ним користуватися. Обрані технології цілком задовольнили потребам, що ставить програмний засіб даного проекту.

ВИСНОВКИ

В процесі роботи було проаналізовано практичні приклади використання безпровідних сенсорних мереж та систем: розглянуті особливості будови схожих систем, проаналізовані проблеми, що виникають при реалізації даних систем. Розглянуто технології та рекомендації щодо ефективного впровадження безпровідних сенсорних мереж у обрану предметну галузь. Проведено аналіз та порівняння існуючих алгоритмів балансування навантаження. Розглянуто функціонал, який вони надають, проаналізовано їх недоліки та переваги, що дозволяє знайти шляхи для вдосконалення цих систем.

При підборі алгоритму навантаження враховувалися апаратні засоби, на яких працює мережа, а також особливості предметної галузі. При тестуванні було встановлено оптимальну конфігурацію мережі, що задовольняє необхідному часу роботи та ресурсам електроенергії, що є в наявності.

Результати моделювання показали, що мережа може працювати зазначений у тестах час (24 години), використовуючи досить обмежену кількість ресурсів електроенергії.

Розроблене програмне забезпечення дало досить точні результати вимірювань характеристик функціонування мережі, обрані технології цілком задовольнили потребам, що ставить програмний засіб даного проекту.

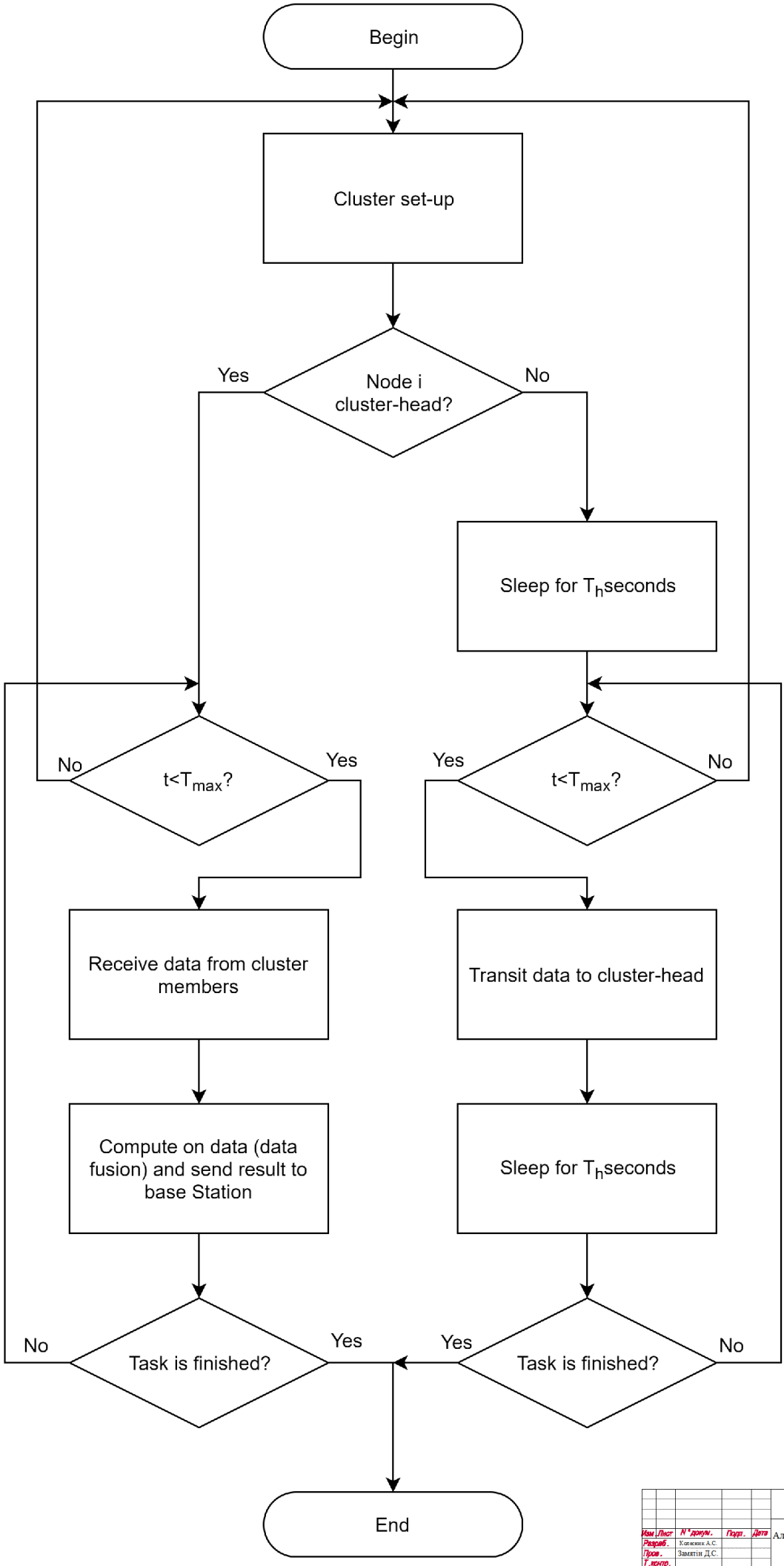
СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. CrossbowMOTE-KIT MICA2 datasheet – Режим доступу до ресурсу: http://www.cmt-gmbh.de/Produkte/WirelessSensorNetworks/Datenblaetter/MOTE-KIT_MICA2_Datasheet.pdf
2. The Power of Sensitivity Improving range with receiver sensitivity – Режим доступу до ресурсу: http://ftp1.digi.com/support/images/Whitepaper-The-power-of-sensitivity.pdf%20(58.79KB).pdf
3. LOAD BALANCING BASED APPROACH TO IMPROVE LIFETIME OF WIRELESS SENSOR NETWORK – Режим доступу до ресурсу: https://www.researchgate.net/publication/276200017_Load_Balancing_Based_Approach_To_Improve_Lifetime_Of_Wireless_Sensor_Network
4. An Overview of Embedded Sensor Networks – Режим доступу до ресурсу: https://www.isi.edu/~johnh/PAPERS/Heidemann04a.pdf
5. A Survey on Load Balancing Techniques for Wireless Sensor Networks – Режим доступу до ресурсу: https://ijarcce.com/upload/2017/february-17/IJARCCE%2079.pdf
6. THRESHOLD-BASED ALGORITHMS FOR POWER-AWARE LOAD BALANCING IN SENSOR NETWORKS – Режим доступу до ресурсу: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.182.7110&rep=rep1&type=pdf
7. LEACH-I Algorithm for WSN Monika1 – Режим доступу до ресурсу: http://www.ijircce.com/upload/2016/march/129_LEACH.pdf
8. User Manual, CC1000DK Development Kit – Режим доступу до ресурсу: https://www.i-components.fi/pdf/b0-CC1000DK-433.pdf

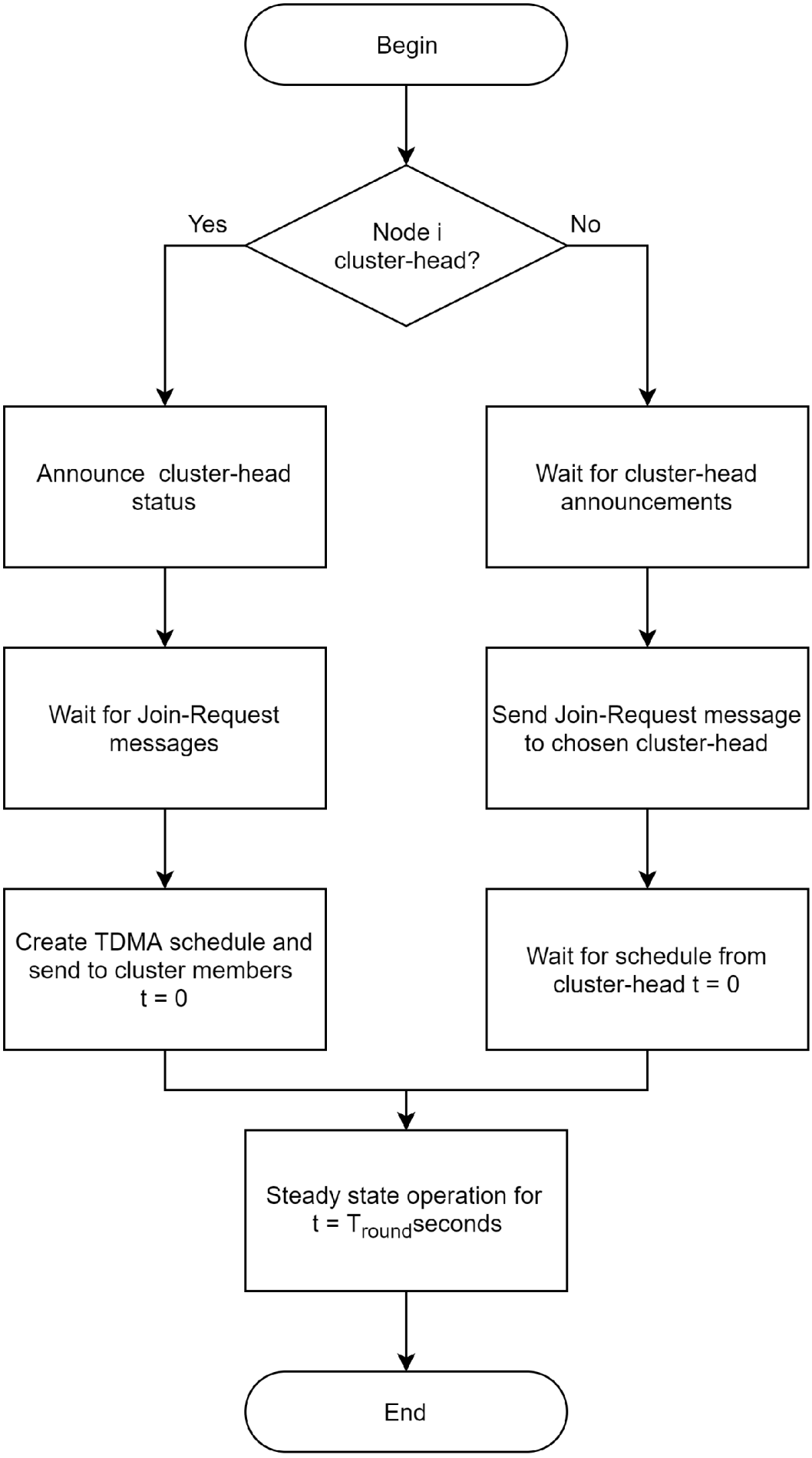
9. Визначення якості води методами біоіндикації – Режим доступу до ресурсу: http://www.necu.org.ua/wp-content/uploads/bioindikacia_2011.pdf

Додаток 1

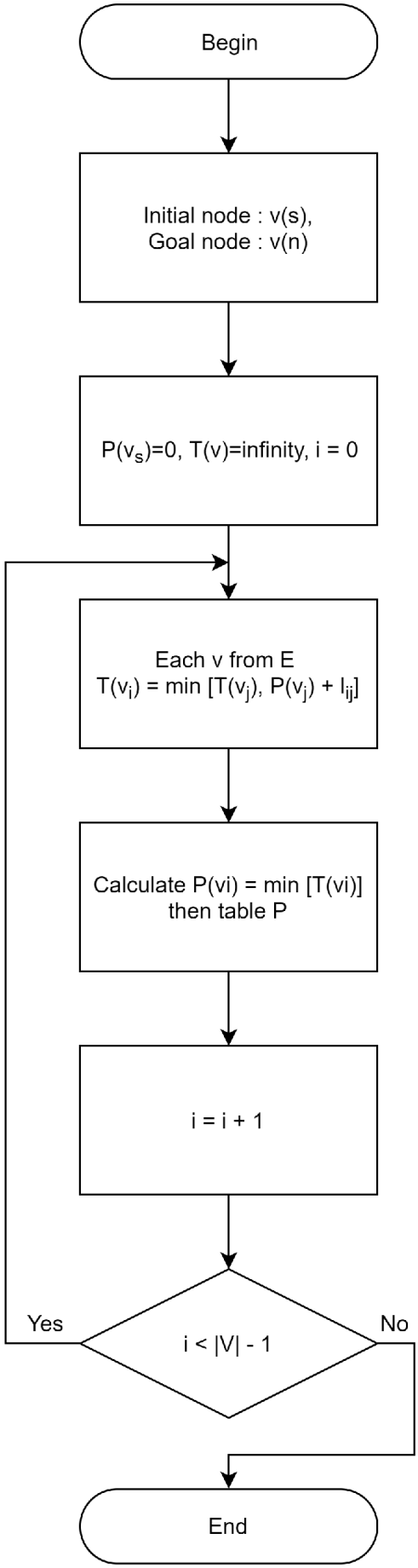
Копії графічних матеріалів



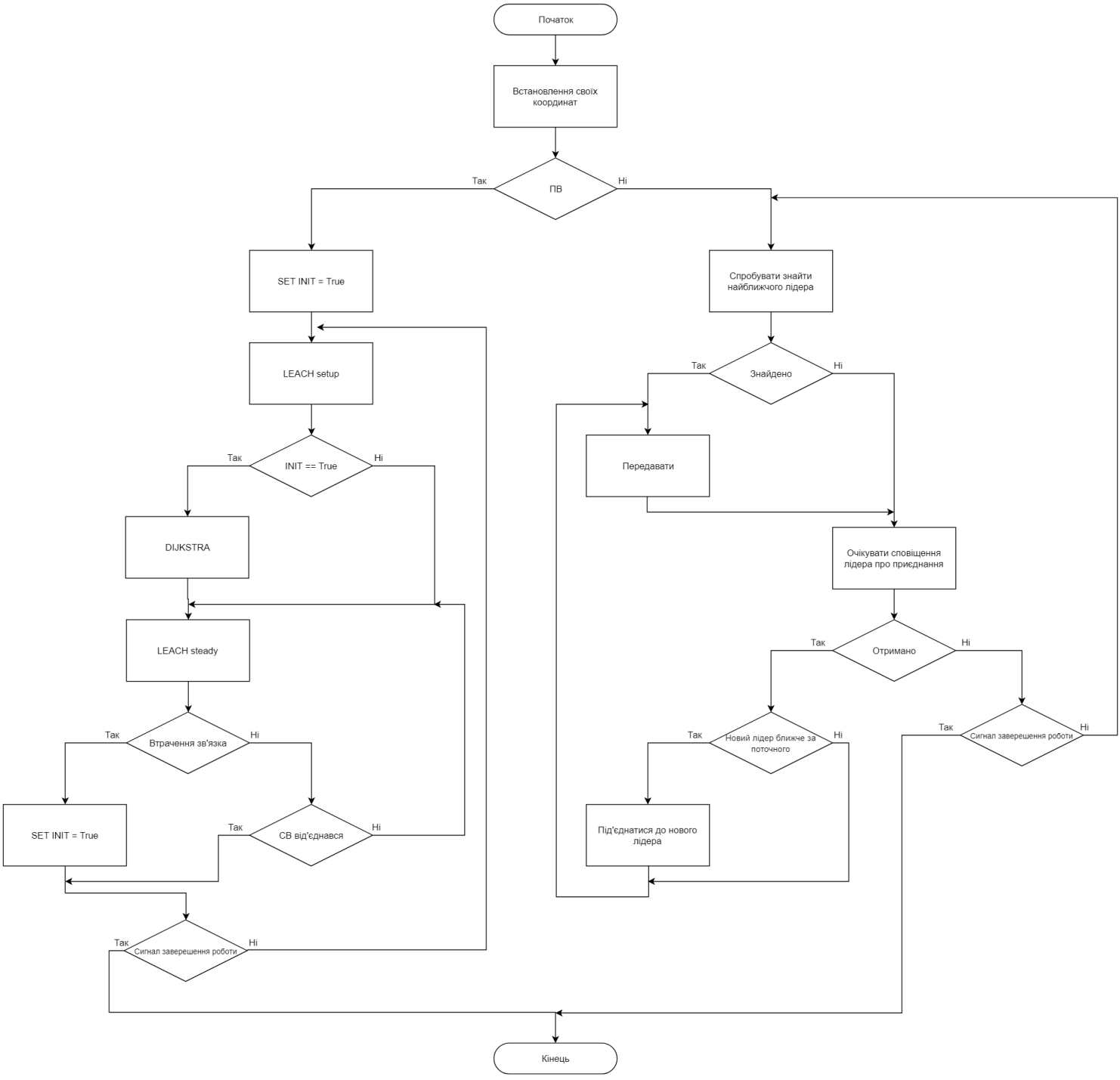
					ІАЛЦ.045492.005 Д1	
Изм.	Лист	N° докум.	Подп.	Дата		Лит. / Масса / Масштаб
Разраб.		Колесник А.С.			Алгоритм LEACH - setup phase.	
Пров.		Замятін Д.С.			**Схема алгоритму**	Лист 1 / Листов 1
Т.контр.						
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФІПМ КВ-51
Утв.		Тарасенко В.П.				



					ІАЛЦ.045492.006 Д2		
Изм.	Лист	N° докум.	Подп.	Дата		Лит	Масса	Масштаб
Разраб.		Клевсик А.С.			Алгоритм LEACH -steadyphase.			
Пров.		Замятін Д.С.			Схема алгоритму			
Т.контр.						Лист 1	Листов 1	
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФІПМ КВ-51		
Утв.		Тарасенко В.П.						



					ІАЛЦ.045492.007 Д3	
Изм.	Лист	N докум.	Подп.	Дата	Алгоритм Дейкстри.	Лит. / Масса / Масштаб
Разраб.		Клевсник А.С.			**Схема алгоритму**	
Пров.		Замятін Д.С.				Лист 1 / Листов 1
Т.контр.						
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФІПМ КВ-51
Утв.		Тарасенко В.П.				



						ІАЛЦ.045492.008 Д4	
Изм.	Лист	N° докум.	Подп.	Дата			
Разраб.		Клевсник А.С.			Схема функціонування вузла.	Лит	Маєса	Масштаб
Пров.		Замятін Д.С.			**Схема алгоритму**			
Т.контр.						Лист 1	Листов 1	
Н.контр.		Клятченко Я.М.				КПІ ім. Ігоря Сікорського, ФІПМ КВ-51		
Утв.		Тарасенко В.П.						

Додаток 2

Лістинг програми

```
import config as cf
import logging
from python.network.node import *
from python.utils.grid import *
import matplotlib.pyplot as plt
from python.utils.utils import *
from python.utils.tracer import *
from python.sleep_scheduling.sleep_scheduler import *
from multiprocessing.dummy import Pool as ThreadPool

class Network(list):
  def __init__(self, init_nodes=None):
    logging.debug('Instantiating nodes...')
    if init_nodes:
      self.extend(init_nodes)
    else:
      nodes = [Node(i, self) for i in range(0, cf.NB_NODES)]
      self.extend(nodes)
      # last node in nodes is the base station
      base_station = Node(cf.BSID, self)
      base_station.pos_x = cf.BS_POS_X
      base_station.pos_y = cf.BS_POS_Y
      self.append(base_station)

    self._dict = {}
    for node in self:
      self._dict[node.id] = node

    self.perform_two_level_comm = 1
    self.round = 0
    self.centroids = []
    self.routing_protocol = None
    self.sleep_scheduler_class = None

    self.initial_energy = self.get_remaining_energy()
    self.first_depletion = 0
    self.per30_depletion = 0
    self.energy_spent = []

  def reset(self):
    for node in self:
      node.energy_source.recharge()
      node.reactivate()

    # allows for updates of BS position between simulations
    self[-1].pos_x = cf.BS_POS_X
    self[-1].pos_y = cf.BS_POS_Y

    self.round = 0
    self.centroids = []
    self.energy_spent = []

    self.routing_protocol = None
    self.sleep_scheduler_class = None

    self.first_depletion = 0
    self.per30_depletion = 0
    self.perform_two_level_comm = 1

  def simulate(self):
    tracer = Tracer()
```

```
    self.routing_protocol.pre_communication(self)

    all_alive = 1
    percent70_alive = 1
    self.deaths_this_round = 0

    if self.sleep_scheduler_class:
      self._sleep_scheduler = SleepScheduler(self,
self.sleep_scheduler_class)

    for round_nb in range(0, cf.MAX_ROUNDS):
      self.round = round_nb
      print_args = (round_nb, self.get_remaining_energy())
      print("round %d: total remaining energy: %f" % print_args)
      nb_alive_nodes = self.count_alive_nodes()
      if nb_alive_nodes == 0:
        break
      tracer['alive_nodes'][2].append(nb_alive_nodes)
      if cf.TRACE_ENERGY:
        tracer['energies'][2].append(self.get_remaining_energy())

      if self.sleep_scheduler_class:
        log = self._sleep_scheduler.schedule()
        for key, value in log.iteritems():
          tracer[key][2].append(value)

      self.routing_protocol.setup_phase(self, round_nb)

      # check if someone died
      if self.deaths_this_round != 0:
        if all_alive == 1:
          all_alive = 0
          self.first_depletion = round_nb
        if float(nb_alive_nodes)/float(cf.NB_NODES) < 0.7 and \
           percent70_alive == 1:
          percent70_alive = 0
          self.per30_depletion = round_nb

      # clears dead counter
      self.deaths_this_round = 0
      self.routing_protocol.broadcast(self)

      self._run_round(round_nb)

    tracer['first_depletion'][2].append(self.first_depletion)
    tracer['30per_depletion'][2].append(self.per30_depletion)

    return tracer

  def _run_round(self, round):
    before_energy = self.get_remaining_energy()
    for i in range(0, cf.MAX_TX_PER_ROUND):
      self._sensing_phase()
      self._communication_phase()
    after_energy = self.get_remaining_energy()
    self.energy_spent.append(before_energy - after_energy)

  def _sensing_phase(self):
    for node in self.get_alive_nodes():
      node.sense()
```

```
  def _communication_phase(self):
    """Each node transmits respecting its hierarchy: leaves start the
    communication, then cluster heads forward the messages, until all
    messages reach the base station. This method works for any hierar-
    chy (even for LEACH).
    """
    #ordinary_nodes = self.get_ordinary_nodes()
    #heads = self.get_ch_nodes()
    #msg = str("%d ordinary nodes, %d heads." % (len(ordinary_nodes),
len(heads)))
    #logging.debug("Hierarchical communication: %s" % (msg))

    alive_nodes = self.get_alive_nodes()
    if self.perform_two_level_comm == 1:
      self._two_level_comm(alive_nodes)
    else:
      self._recursive_comm(alive_nodes)

  def _recursive_comm(self, alive_nodes):
    next_alive_nodes = alive_nodes[:]
    for node in alive_nodes:
      #check if other nodes must send info to this node
      depends_on_other_node = 0
      for other_node in alive_nodes:
        #if other_node == node:
        #  continue
        if other_node.next_hop == node.id:
          depends_on_other_node = 1
          break

      if not depends_on_other_node:
        node.transmit()
        next_alive_nodes = [n for n in next_alive_nodes if n != node]

    if len(next_alive_nodes) == 0:
      return
    else:
      self._recursive_comm(next_alive_nodes)

  def _two_level_comm(self, alive_nodes):
    # heads wait for all ordinary nodes, then transmit to BS
    for node in self.get_ordinary_nodes():
      node.transmit()
    for node in self.get_heads():
      node.transmit()

  def get_alive_nodes(self):
    return [node for node in self[0:-1] if node.alive]

  def get_active_nodes(self):
    is_active = lambda x: x.alive and not x.is_sleeping
    return [node for node in self[0:-1] if is_active(node)]

  def get_ordinary_nodes(self):
    return [node for node in self if node.is_ordinary() and node.alive]

  def get_heads(self, only_alives=1):
    input_set = self.get_alive_nodes() if only_alives else self
    return [node for node in input_set if node.is_head()]
```

```
  def get_sensor_nodes(self):
    """Return all nodes except base station."""
    return [node for node in self[0:-1]]

  def get_average_energy(self):
    return np.average(self.energy_spent)

  def someone_alive(self):
    for node in self[0:-1]:
      if node.alive == 1:
        return 1
    return 0

  def count_alive_nodes(self):
    return sum(x.alive for x in self[:-1])

  def get_BS(self):
    # intention: make code clearer for non-Python readers
    return self[-1]

  def get_node(self, id):
    """By default, we assume that the id is equal to the node's posi-
    tion in the list, but that may not be always the case.
    """
    return self._dict[id]

  def notify_position(self):
    for node in self.get_alive_nodes():
      node.transmit(msg_length=cf.MSG_LENGTH, destination=self.get_BS())

  def broadcast_next_hop(self):
    base_station = self.get_BS()
    for node in self.get_alive_nodes():
      base_station.transmit(msg_length=cf.MSG_LENGTH, destination=node)

  def get_nodes_by_membership(self, membership, only_alives=1):
    input_set = self.get_alive_nodes() if only_alives else self
    condition = lambda node: node.membership == membership and node.id !=
cf.BSID
    return [node for node in input_set if condition(node)]

  def get_remaining_energy(self, ignore_nodes=None):
    set = self.get_alive_nodes()
    if len(set) == 0:
      return 0
    if ignore_nodes:
      set = [node for node in set if node not in ignore_nodes]
    transform = lambda x: x.energy_source.energy
    energies = [transform(x) for x in set]
    return sum(x for x in energies)

  def set_aggregation_function(self, function):
    for node in self:
      node.aggregation_function = function

  def split_in_clusters(self, nb_clusters=cf.NB_CLUSTERS):
    clusters = []
    for cluster_idx in range(0, nb_clusters):
      nodes = self.get_nodes_by_membership(cluster_idx)
      cluster = Network(init_nodes=nodes)
      cluster.append(self.get_BS())
```

```
    clusters.append(cluster)
  return clusters

def _calculate_nb_neighbors(self, target_node):
  # if number of neighbors was calculated at least once
  # skips calculating the distance
  if target_node.nb_neighbors != -1:
    # only check if there are dead nodes
    all_neighbors = target_node.neighbors
    nb_dead_neighbors = sum(1 for x in all_neighbors if not x.alive)
    target_node.neighbors[:] = [x for x in all_neighbors if x.alive]
    return target_node.nb_neighbors - nb_dead_neighbors

  nb_neighbors = 0
  shortest_distance = cf.COVERAGE_RADIUS*2
  for node in self.get_alive_nodes():
    if node == target_node:
      continue
    distance = calculate_distance(target_node, node)
    if distance <= cf.COVERAGE_RADIUS:
      nb_neighbors += 1
      target_node.neighbors.append(node)
      if distance < shortest_distance:
        shortest_distance = distance

  if shortest_distance != cf.INFINITY:
    exclusive_radius = shortest_distance - cf.COVERAGE_RADIUS
    if exclusive_radius < 0:
      exclusive_radius = 0.0

  node.nb_neighbors = nb_neighbors
  node.exclusive_radius = exclusive_radius

def update_neighbors(self):
  for node in self.get_alive_nodes():
    self._calculate_nb_neighbors(node)

  self.update_sleep_prob()

def update_sleep_prob(self):
  for node in self.get_alive_nodes():
    node.update_sleep_prob()
```

```
from python.routing.priodict import priorityDictionary

def Dijkstra(G,start,end=None):
  D = {}  # dictionary of final distances
  P = {}  # dictionary of predecessors
  Q = priorityDictionary()  # estimated distances of non-final vertices
  Q[start] = 0

  for v in Q:
    D[v] = Q[v]
    if v == end: break

    for w in G[v]:
      vwLength = D[v] + G[v][w]
      if w in D:
        if vwLength < D[w]:
          raise ValueError("Dijkstra: found better path to already-final
vertex")
      elif w not in Q or vwLength < Q[w]:
        Q[w] = vwLength
        P[w] = v

  return (D,P)

def shortestPath(G,start,end):
  D,P = Dijkstra(G,start,end)
  Path = []
  while 1:
    Path.append(end)
    if end == start: break
    end = P[end]
  Path.reverse()
  return Path
```

Додаток 3

Презентація

Дипломний проект
на тему:
«Система балансування навантаження в децентралізованих розподілених мережах»

Виконав студент групи КВ-51
Колесник О.С

Керівник дипломного проекту
Доц. к.т.н, доц. кафедри СПСКС Замятін Д.С.

Алгоритм балансування навантаженя

- Поєднання алгоритму LEACH та Дейкстри

Фази алгоритму LEACH:

1) налаштування (set-up phase);

- Обирання лідера;
- Налаштування кластера;
- Створення таблиці поділу часу.

2) стабільна (steady phase)

- відправка даних

Мережа

Мережа датчиків розрахована на водойму до 800 метрів завдовжки.

Використовується 3 типи вузлів:

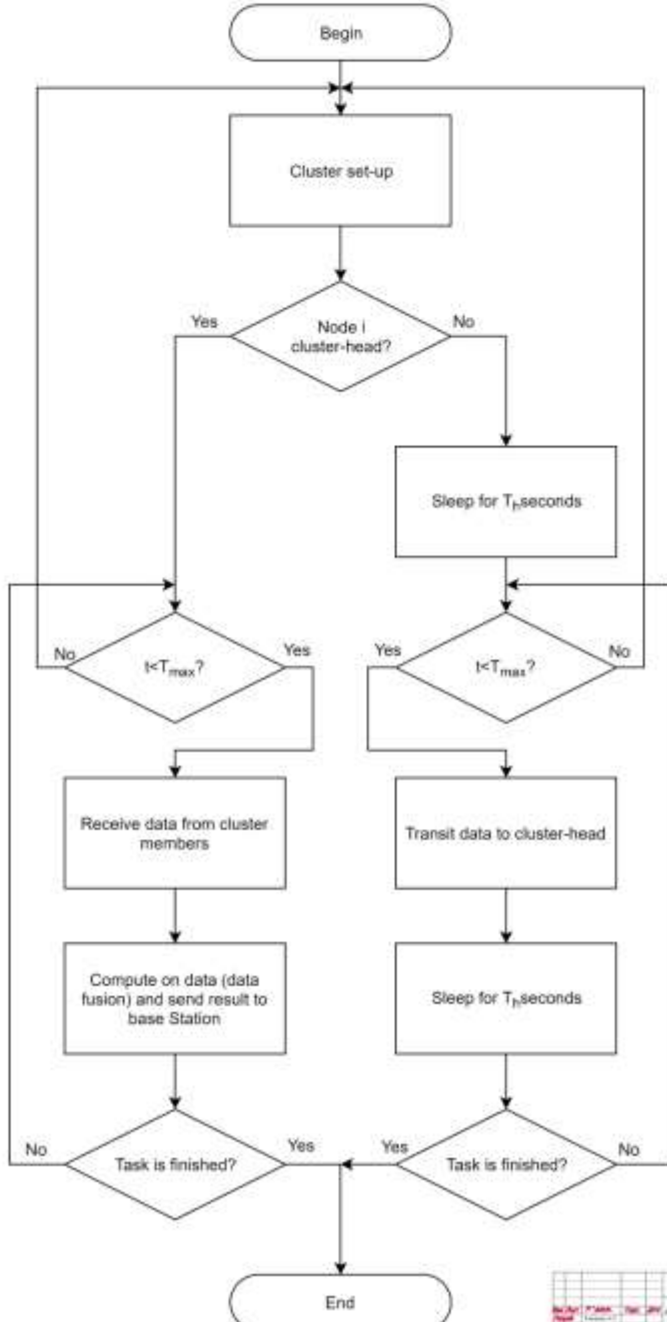
- слабкий вузол (надалі СВ)
- потужний вузол (надалі ПВ)
- базова станція (БС)

Сильні вузли використовуються як концентратори трафіку. Вони є краще за апаратними характеристиками. Основні з них, що враховуються для балансування трафіку в даній мережі:

- запас електроенергії
- кількість каналів радіозв`язку.

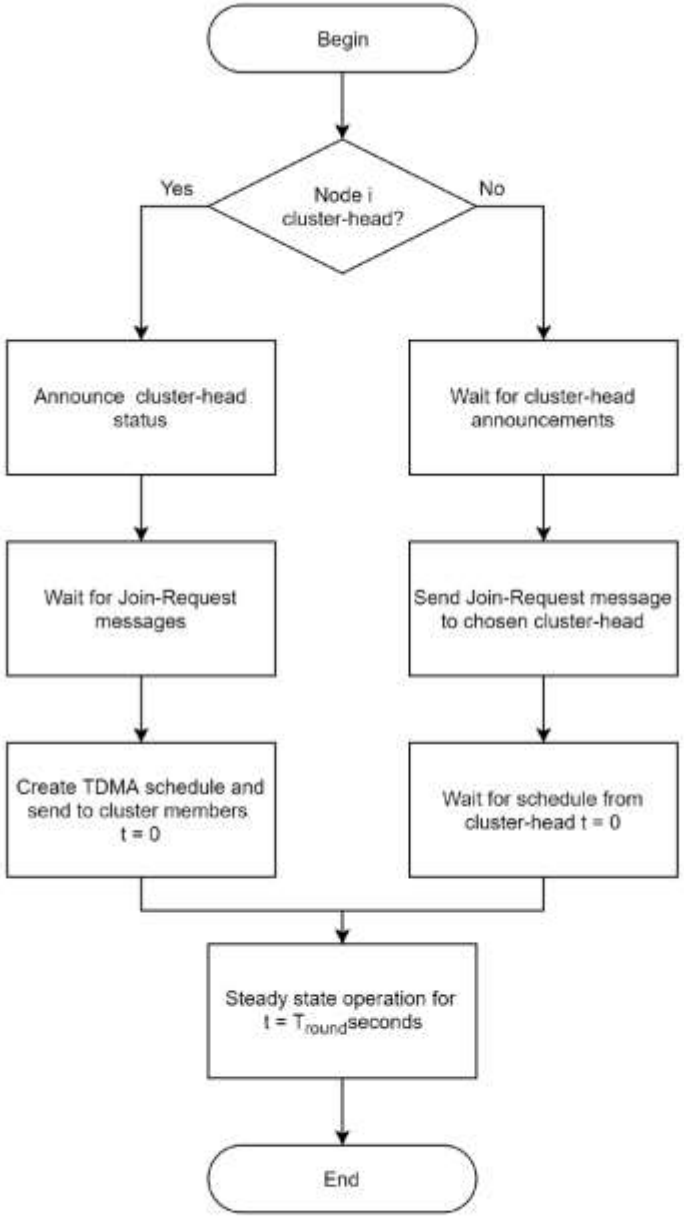
Побудова кластерів

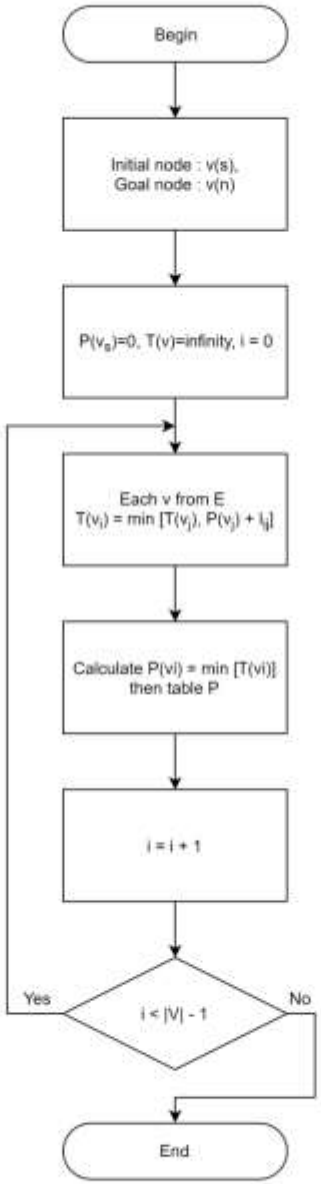
- Протягом першого кроку вузли вирішують чи будуть вони головами кластерів за допомогою спеціальної формули (рис. 2.4), де T(n) — порогове значення. Вузол стає лідером на поточний раунд коли число менше за порогове. Коли вузол обраний лідером, він не може стати лідером знову доки всі потужні вузли не побудуть у ролі лідера один раз. Це покращує споживання енергії.
- Лідер обирається серед потужних вузлів. Перевага надається вузлу з більшим запасом енергії та найменш навантаженим каналом зв`язку.

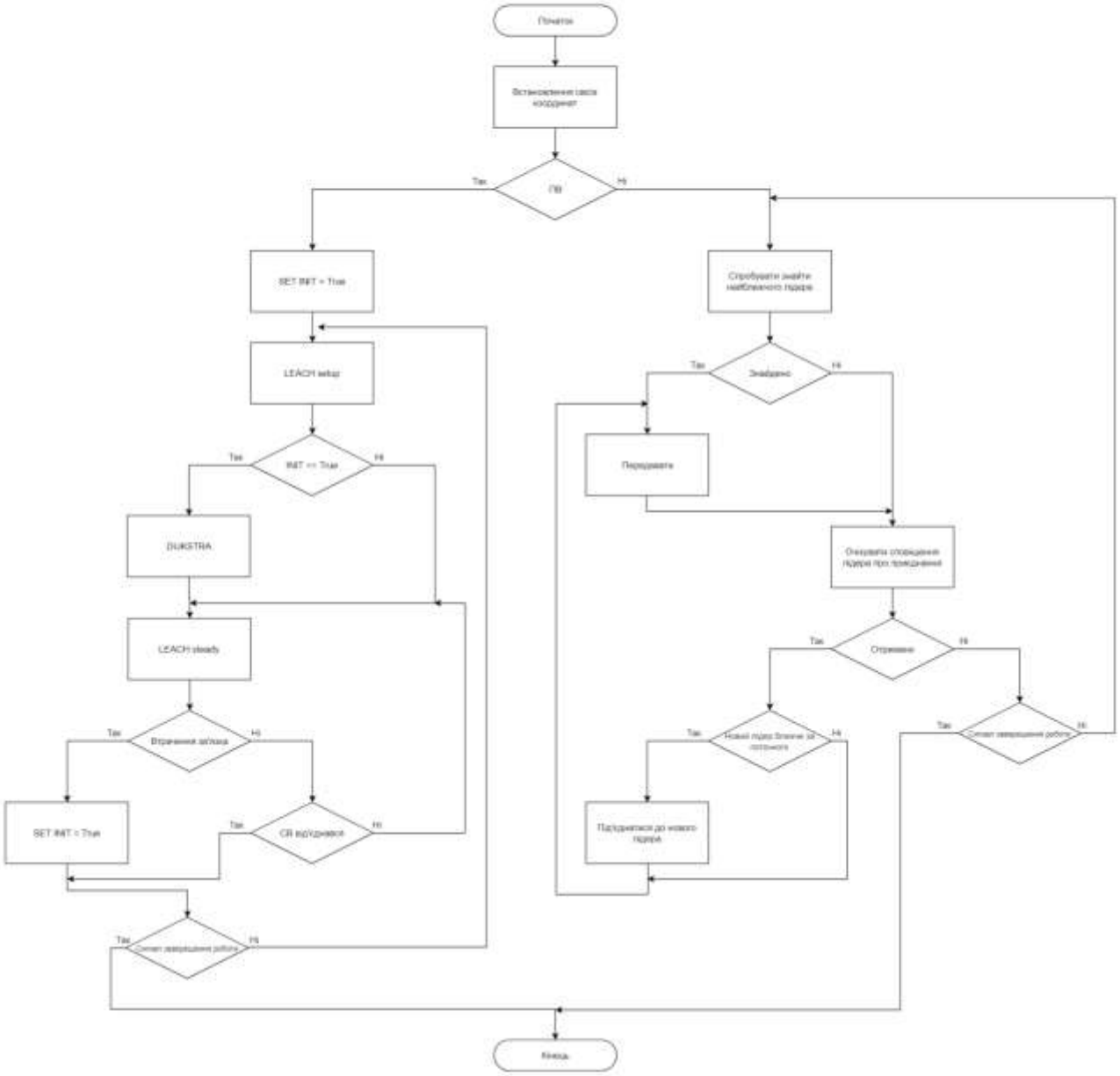


ІАЛЦ.045492.005 Д1

Схема алгоритму


Begin
Node i cluster-head?
Yes
No
Announce cluster-head status
Wait for cluster-head announcements
Wait for Join-Request messages
Send Join-Request message to chosen cluster-head
Create TDMA schedule and send to cluster members t = 0
Wait for schedule from cluster-head t = 0
Steady state operation for t = T_{round}seconds
End


Begin
Initial node : v(s),
Goal node : v(n)
P(v_s)=0, T(v)=infinity, i = 0
Each v from E
T(v_i) = min [T(v_j), P(v_j) + l_ij]
Calculate P(vi) = min [T(vi)]
then table P
i = i + 1
Yes
i < |V| - 1
No
End


Початок
Встановлення своїх координат
ГВ
Так
Ні
SET INIT = True
LEACH setup
INIT == True
Так
Ні
DIJKSTRA
LEACH steady
Втрачення зв'язку
Так
Ні
SET INIT = True
CR відсутній
Так
Ні
Сигнал завершення роботи
Так
Ні
Спробувати знайти найближчого підряд
Знайдено
Так
Ні
Передавати
Очікувати сповіщення підряд про приєднання
Отримано
Так
Ні
Новий підряд Знаючи за попереднього
Так
Ні
Підєднатися до нового підряд
Сигнал завершення роботи
Так
Ні
Кінець

Апаратна реалізацяія вузлів мережі

Використовуються компоненти MPR400CB (рис. 2.1), MPR410CB комплекту розробки сенсорних мереж MICA2

Загальна характеристика комплекту:

- платформа для комерційної розробки;
- мікросхеми процесорних пристроїв та радіопередавачів - приймальників з надійним мережевим зв’язком;
- мультифункціональні плати, зручний зовнішній сенсорний інтерфейс, послідовні порти та виходи Ethernet;
- постачається з Windows додатком для створення топології мережі, доступу до сенсорних даних, аналізу поточних даних та історії спостереження;
- інструменти для налагодження та розробки, включаючи модуль радіозв’язку та всі його компоненти.

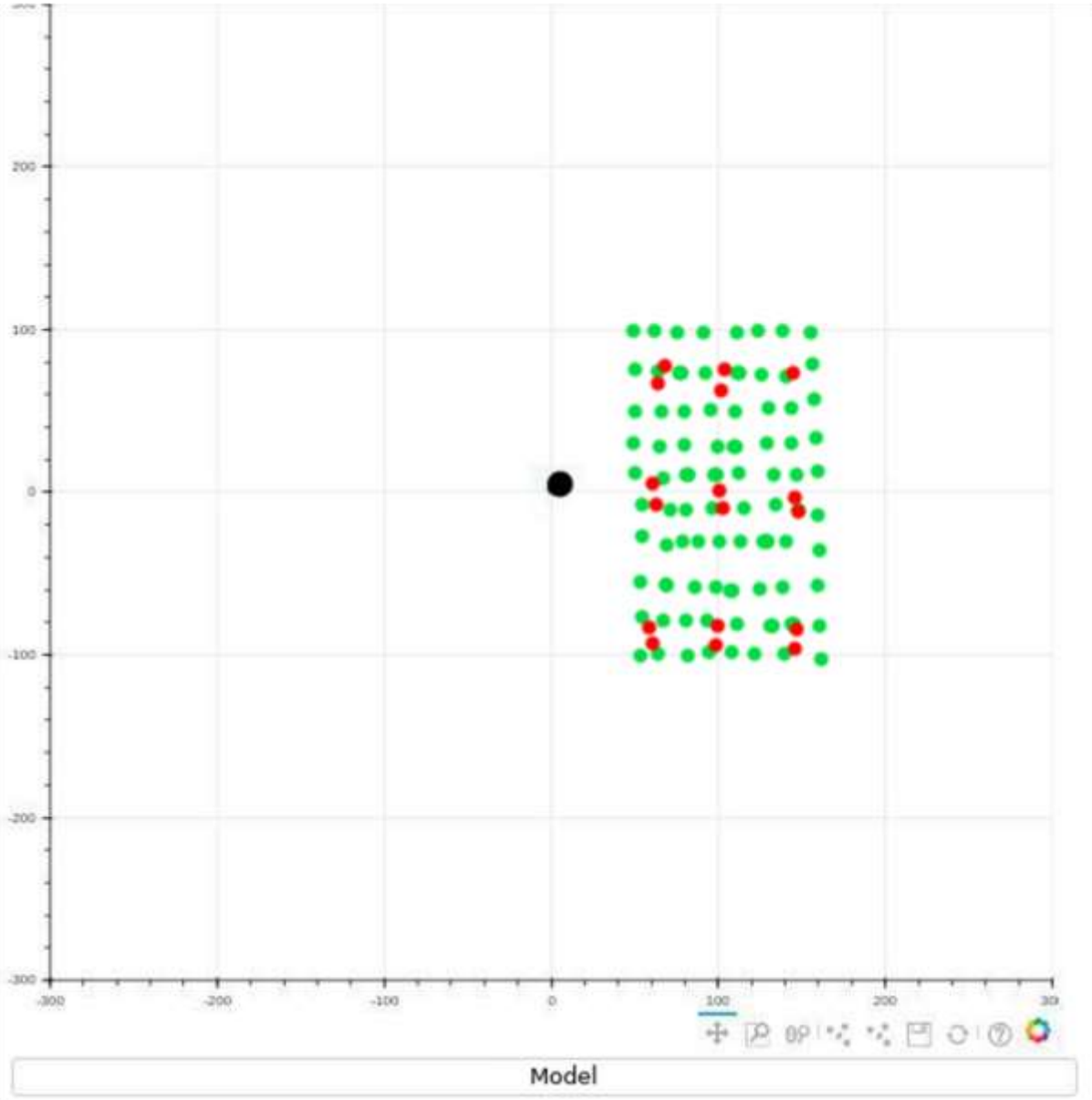
MPR400CB

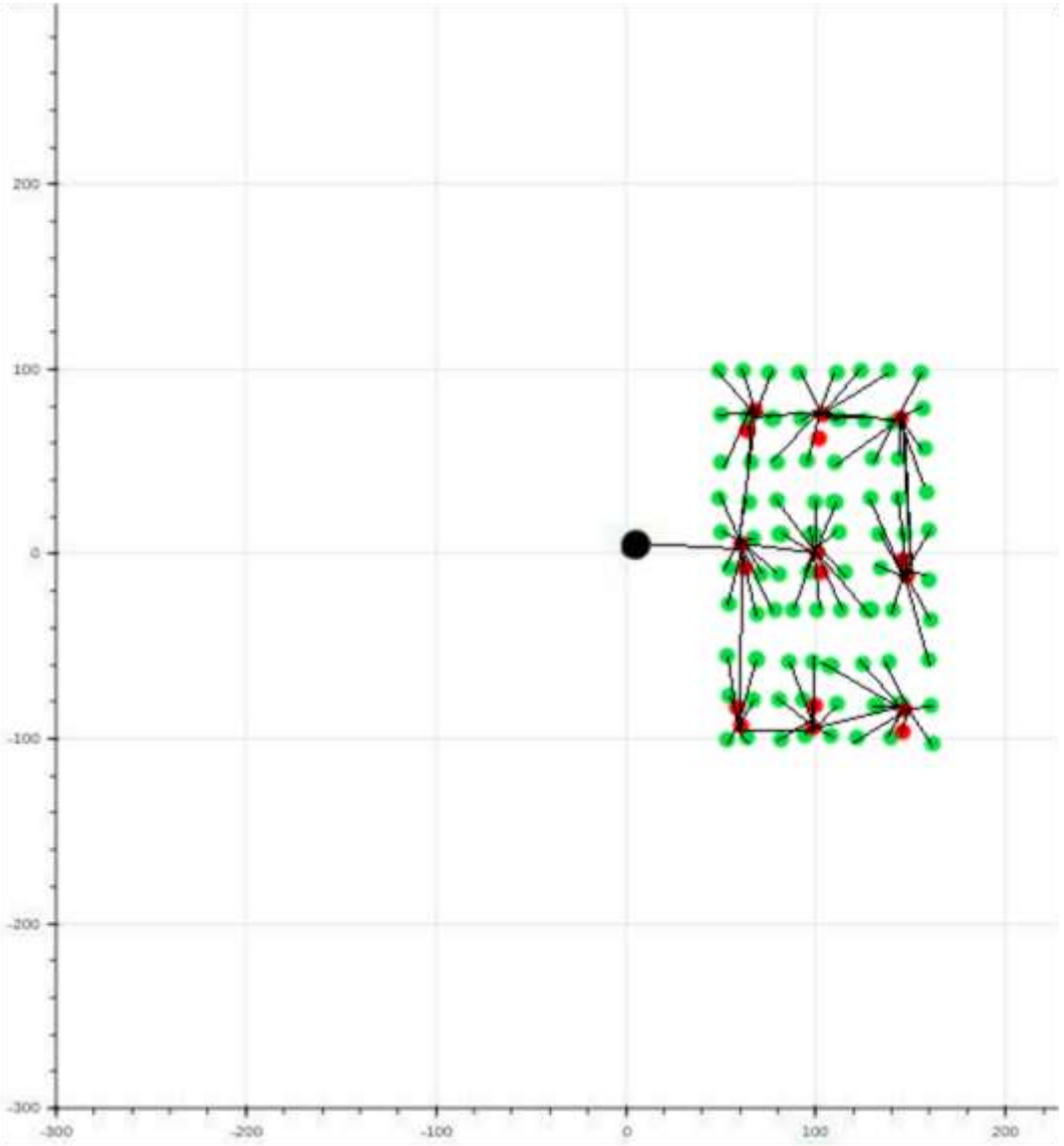
Основні властивості компоненту для створення моута:

- 868/916MHz мультиканальний приймальник-передавач;
- TinyOS (TOS) Distributed Software Operating System v1.0 з покращеним мережевим стеком та інструментами налагодження;
- підтримка бездротового віддаленого перепрограмування;
- широкий спектр мікросхем сенсорів та додаткових чипів;

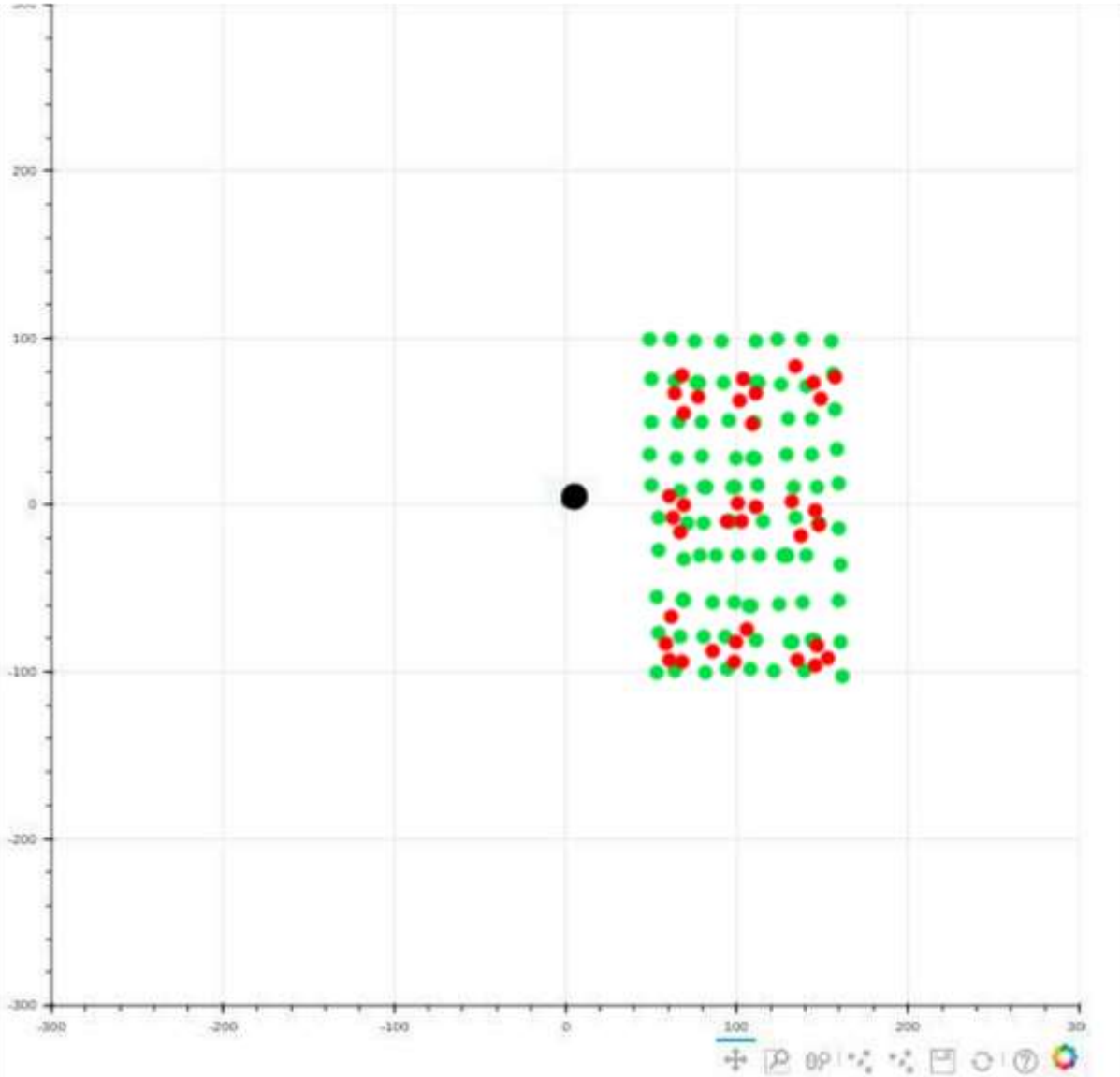
Тестування

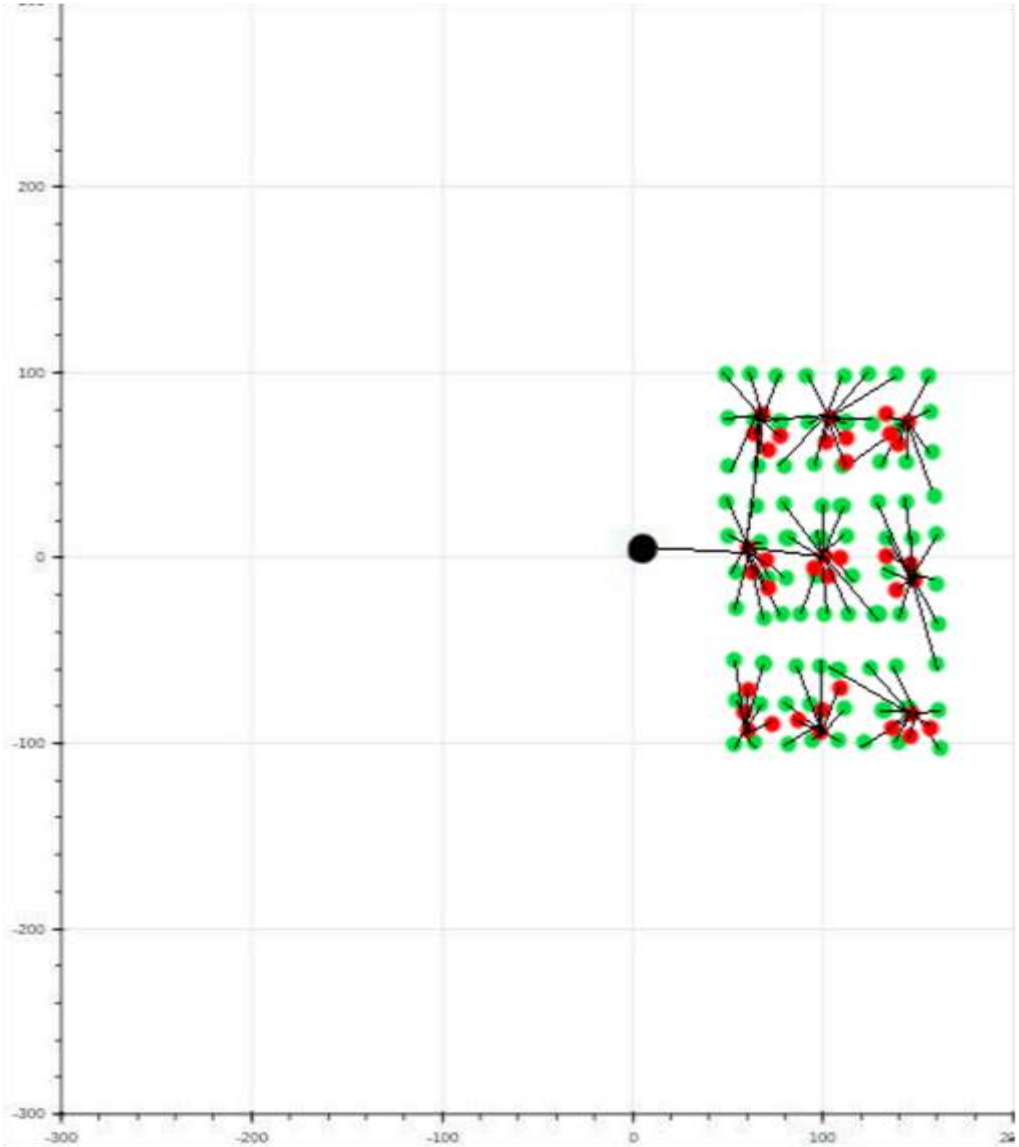
- Тести проводитимемо на для мережі з наступними параметрами.
- Ділянка - 100x200.
- Очікуваний час роботи — 12 годин.
- Дистанція між датчиками — 10 метрів.


200
100
0
-100
-200
-300
-300
-200
-100
0
100
200
Model



Кількість вузлів: 98;
Перше від’єднання вузла: 3 год. 21хв;
Час життя мережі: 7 год. 10хв.

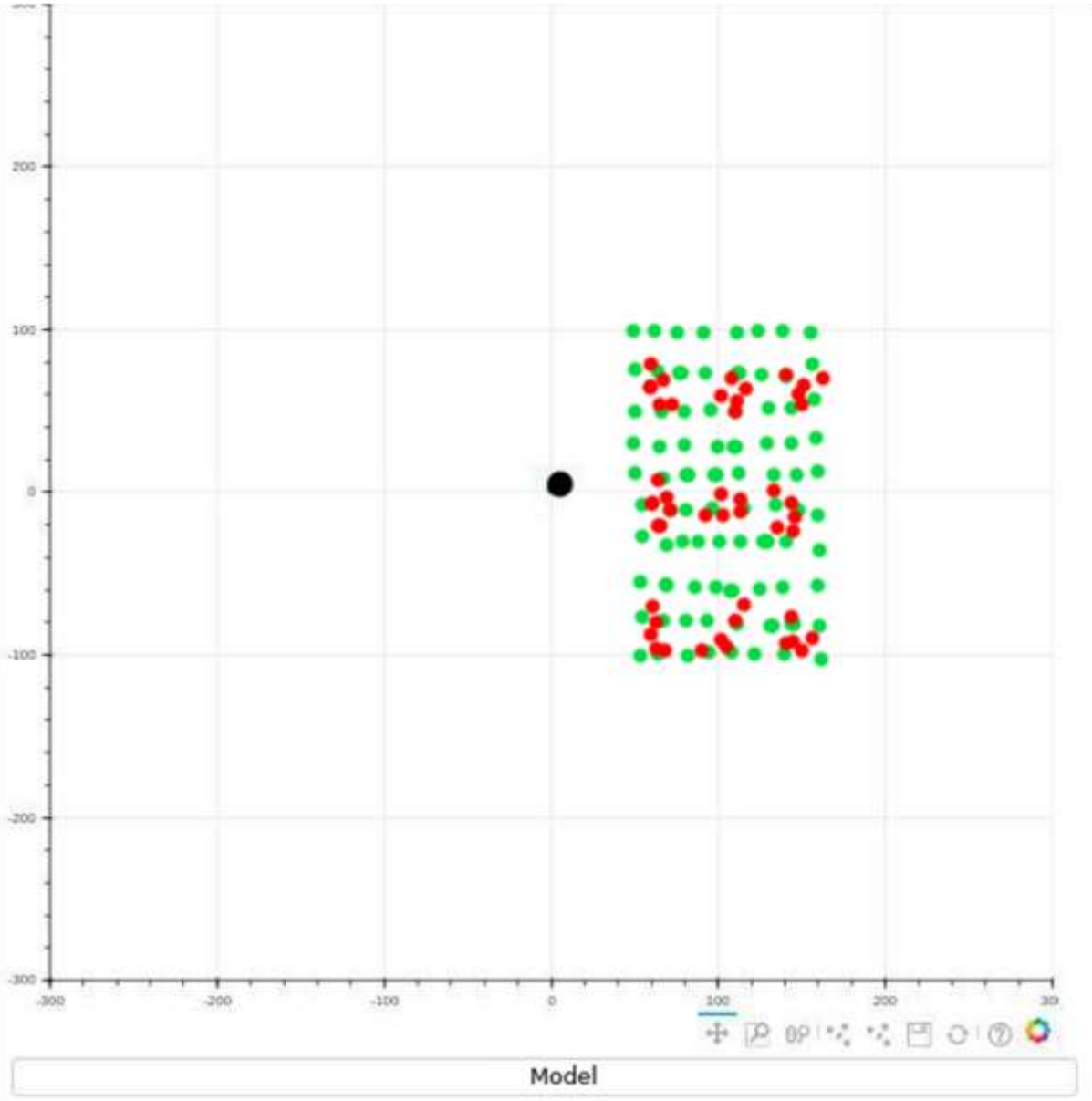

200
100
0
-100
-200
-300
-300
-200
-100
0
100
200
Model

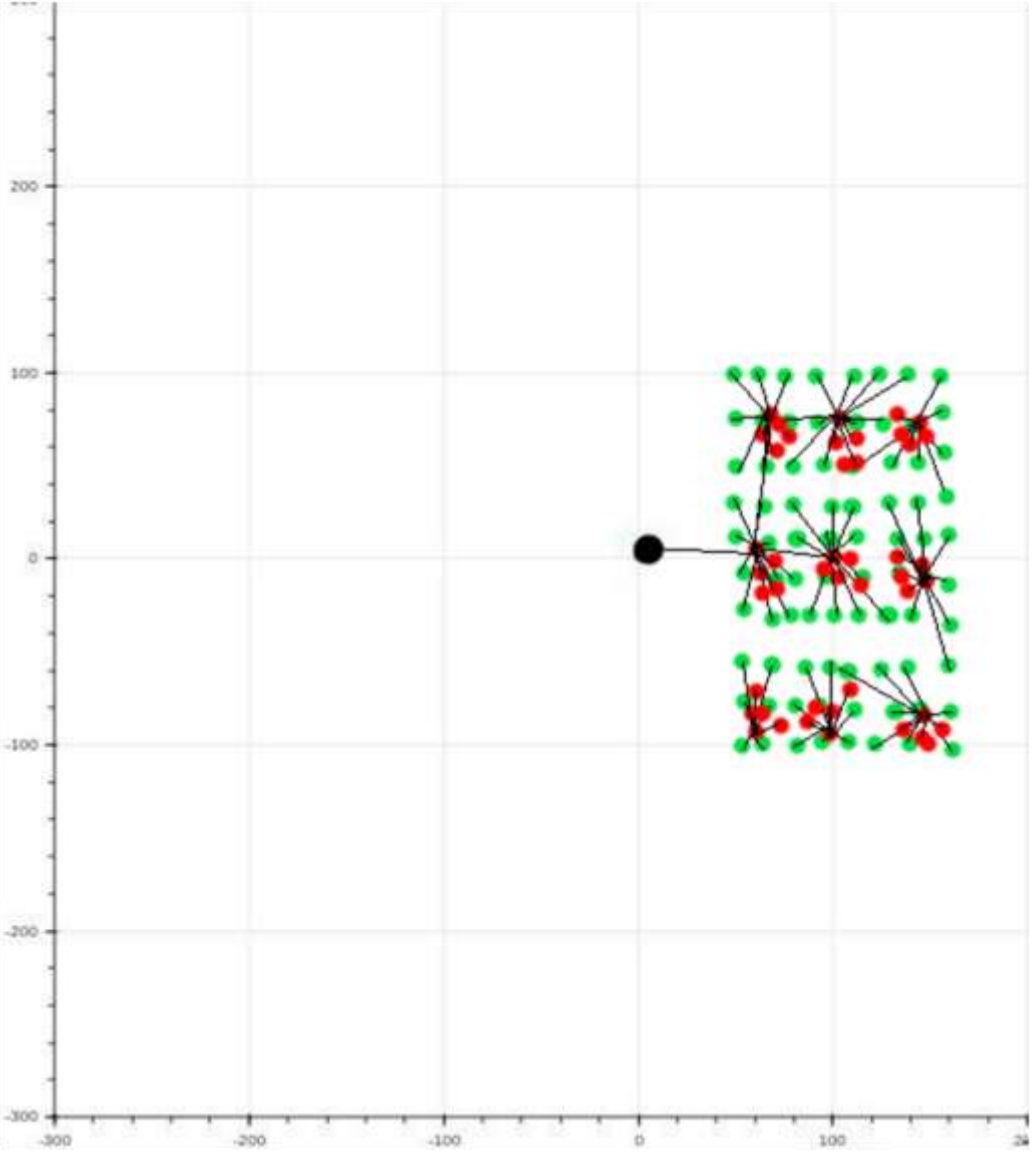


Кількість вузлів: 116;

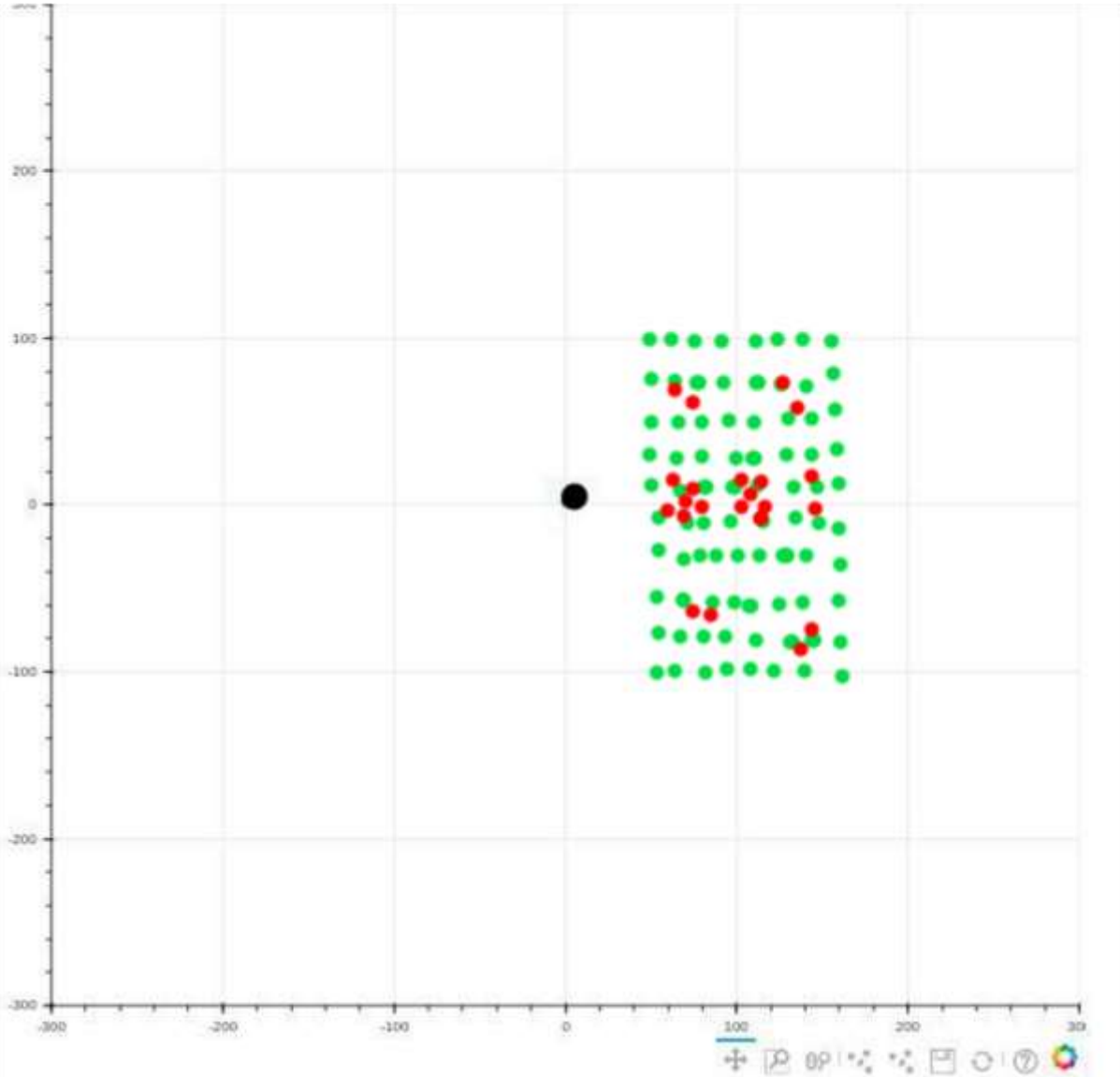
Перше від’єднання вузла: 3 год. 2хв;

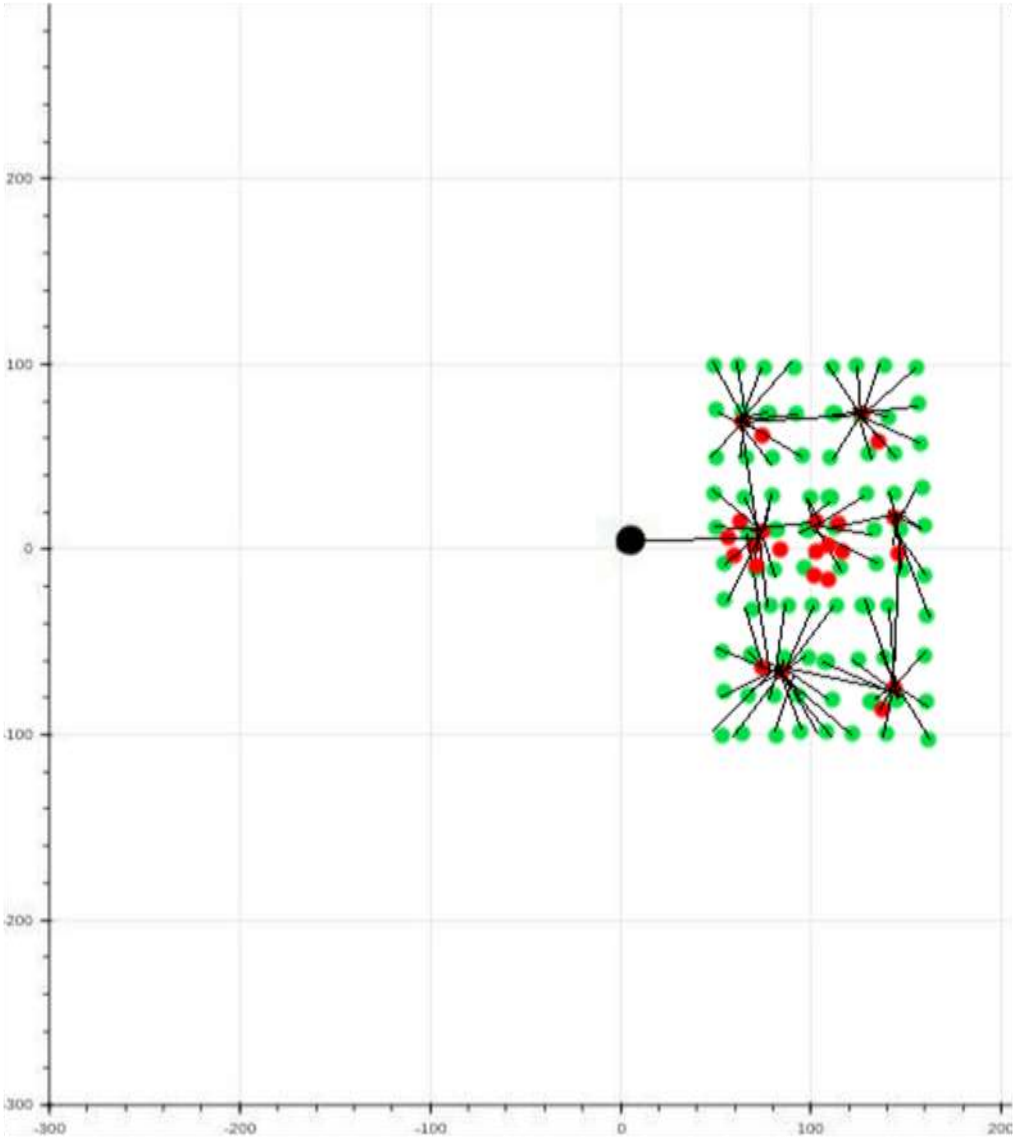
Час життя мережі: 8 год. 31хв.


200
100
0
-100
-200
-300
-300
-200
-100
0
100
200
Model



Кількість вузлів: 125;
Перше від’єднання вузла: 2 год. 57хв;
Час життя мережі: 8 год. 36хв.


200
100
0
-100
-200
-300
-300
-200
-100
0
100
200
Model



Кількість вузлів: 107;

Перше від’єднання вузла: 3 год. 4хв;

Час життя мережі: 12 год. 15хв.