Розроблене програмне забезпечення складається із наступних модулів:

- ядро моделі (app.simulator) мережі вміщує класи, що відносяться до моделювання канального та фізичного рівнів;
- ядро моделі маршрутизації (app.simulator.packets, routing) вміщує класи, що відносяться до моделювання мережевого рівня;
- інтерфейс користувача (app.simulator.gui) вміщує класи, що відносяться до візуалізації та керування процесом моделювання.

3.1. Опис інтерфейсу користувача

Інтерфейс користувача (рис.3.1) складається з наступних елементів:

- строчка меню
- панель завдання вхідних параметрів моделі (рис 3.2а)
- панель завдання вихідних параметрів моделі (рис 3.2б)
- візуальне представлення графу мережі

Можна задавати наступні вхідні параметри:

- алгоритм маршрутизації
- режим алгоритму
- кількість вузлів
- кількість зв’язків вузла
- кількість вузлів, що відправляють пакети
- відхилення топології від регулярної
- щільність вузлів
- кількість мобільних вузлів
- кількість мілісекунд у одному тику

У якості вихідних даних можуть бути:

- кількість відправлених пакетів
- кількість отриманих пакетів
- кількість загублених пакетів
- кількість передач пакетів
- кількість передач службових пакетів алгоритму

- кількість пакетів, що знаходяться у черзі
- кількість маршрутів, що розшукується
- кількість знайдених маршрутів

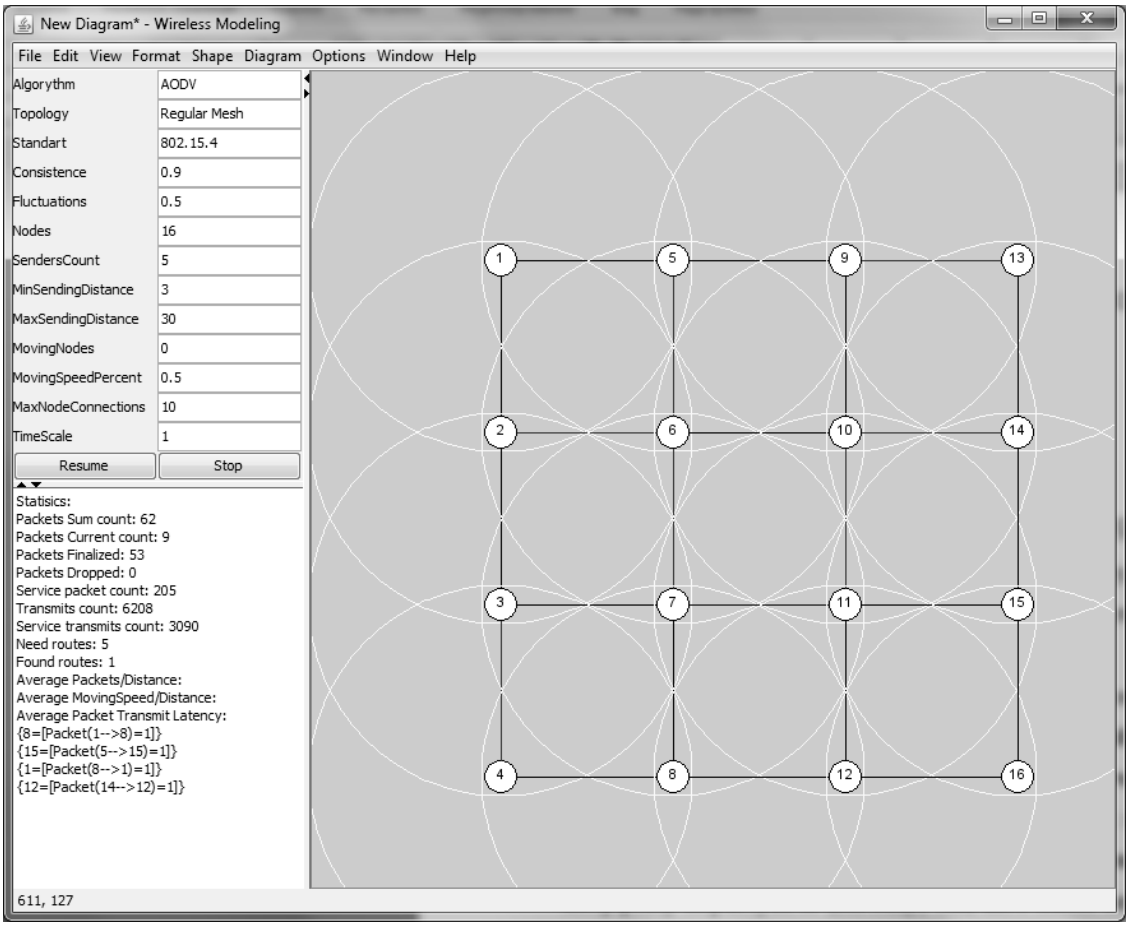
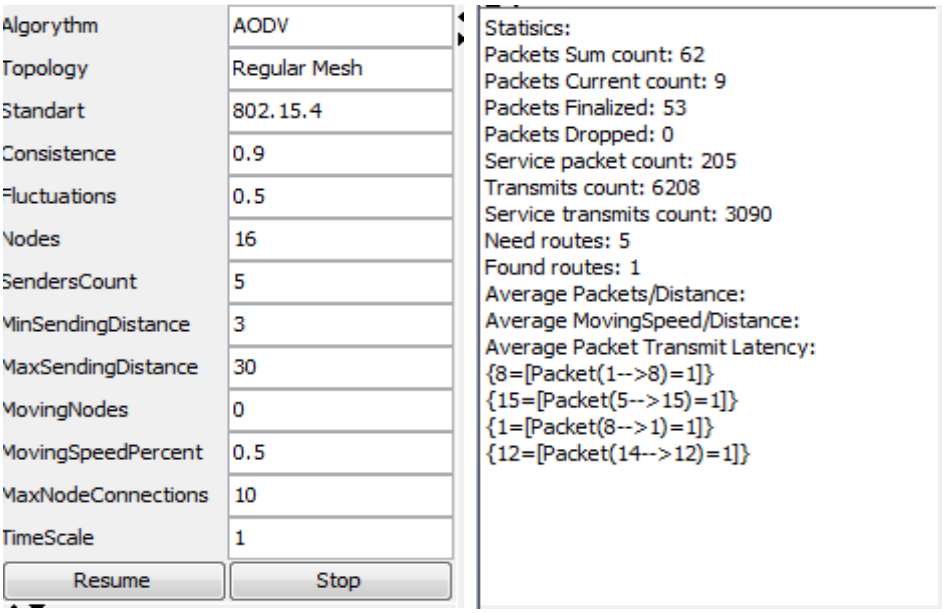


Рисунок 3.1 – Інтерфейс користувача



а) вхідні параметри б) вихідні параметри

Рисунок 3.2 – Панелі вхідних та вихідних параметрів.

Візуалізація графу мережі виконується за допомогою бібліотеки JGraph. Оскільки JGraph побудовано за патерном MVC (рис.3.3.) візуалізація моделі ядра покладалася у тому, що модель JGraph було пов'язано з об'єктами класів ядра, а контролер із ініціалізацією цих об'єктів. Було використано наступні класи бібліотеки:

- mxGraph – клас, що агрегує інші класи JGraph, та реалізує контролер.
- mxGraphModel – клас моделі.
- mxCell – клас елементу графу, що може описувати як вершини так і ребра графу.
- mxView – клас виду, що відповідає за візуалізацію усього графу або конкретного елементу.

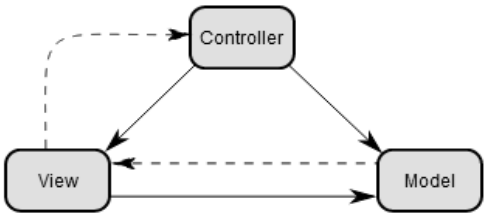


Рисунок 3.3 – Архітектурний шаблон «Моде́ль-вид-контро́лер».

На базі цих класів, та класу SimpleEditor (каркас для побудування редакторів) із пакету utils і було побудовано інтерфейс моделюючого ПЗ.

Класи моделі mxGraph були зв'язані з класами моделі ядра (Link і Router) за допомогою властивості Data класу mxCell та обробників подій addCell (додавання елементу графу), removeCell (видалення елементу графу). Налаштування інтерфейсу полягало у конфігуруванні об'єкту класу mxView.

3.2. Опис ядра моделі бездротової мережі

Можна виділити основні класи ядра (рис 3.4.):

- клас, що описує вузли (Router)
- клас, що описує зв'язки між вузлами (Link)

- клас, що описує пакети у мережі (Packet)

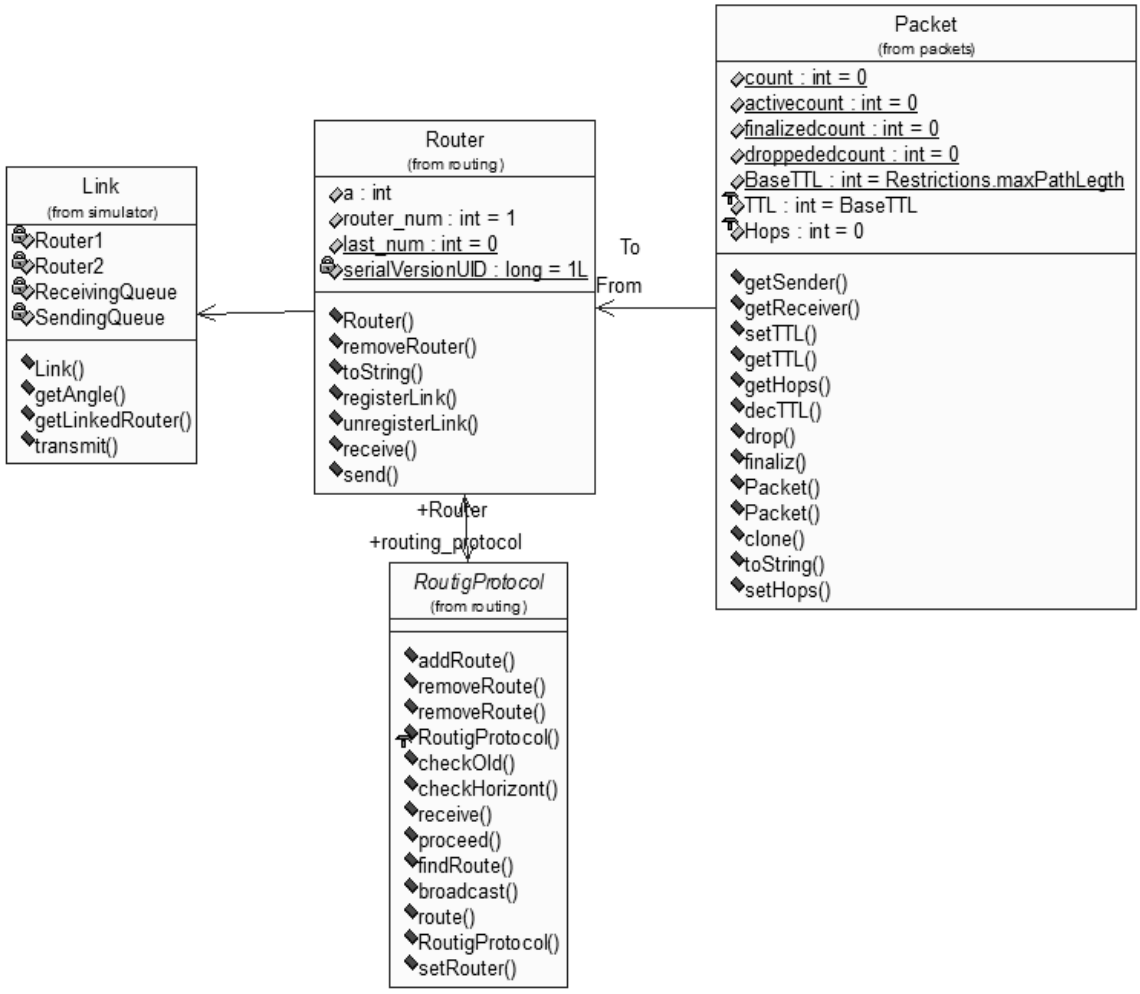


Рисунок 3.4 – Основні класи ядра моделі мережі.

Найбільш важливим класом, що описує вузол, є клас «Router». Об’єкти цього класу обмінюються пакетами через об’єкти класу «Link». Підключення каналів зв’язку до вузла реалізується за допомогою методів «registerLink» і «unregisterLink». У ході моделювання пакет, після передачі по каналу передається через «Router» за допомогою методів send і receive. Ці методи виконуються у потоці того об’єкту «Link», який витягнув пакет із черги.

Маршрутизація пакетів винесена у клас «RoutingProtocol». Пакети обробляються методом «route» цього класу, якщо пунктом їх призначення не є сам роутер, або вузол у його підмережі. Цей метод направляє пакет на потрібний зв’язкок за допомогою таблиці маршрутизації.

На кожному зв’язку організована черга пакетів. Оскільки після того як пакет попав у чергу, керування повинно повертатися до роутеру, вивід пакетів з черги виконується у потоці об’єкту «Link», який буде цей пакет передавати іншому роутеру (рис 3.5). Таким чином організується паралельна передача пакетів.

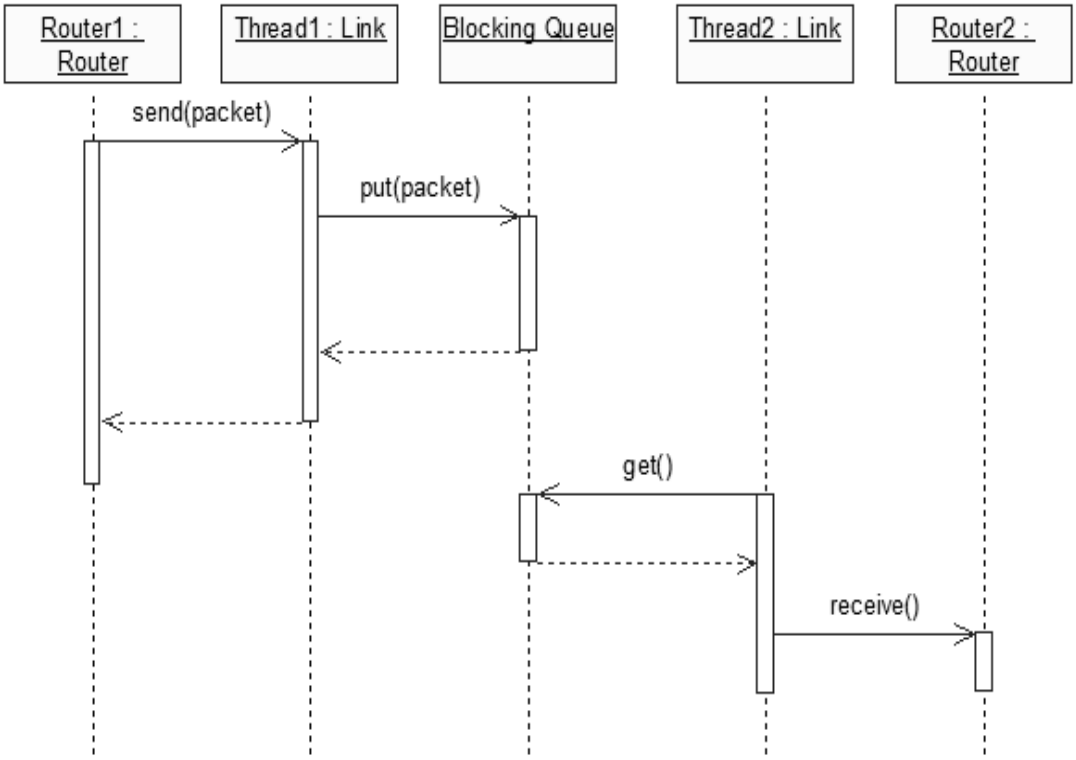


Рисунок 3.5 – Організація паралельного моделювання передачі пакетів

3.3. Розробка ядра моделі маршрутизації

Первинні функції протоколу маршрутизації описані у абстрактному класі RoutingProtocol, що є суперкласом для імплементацій алгоритмів. У ньому реалізовані наступні методи:

- addRoute – додає запис у таблицю маршрутизації
- removeRoute – видаляє запис із таблиці маршрутизації
- route – маршрутизує пакет за таблицею маршрутизації (рис 3.6.)
- broadcast – широкомовно розсилає пакет (рис 3.7.)
- checkHorizont – перевірка на повторну пересилку пакету при широкомовленні

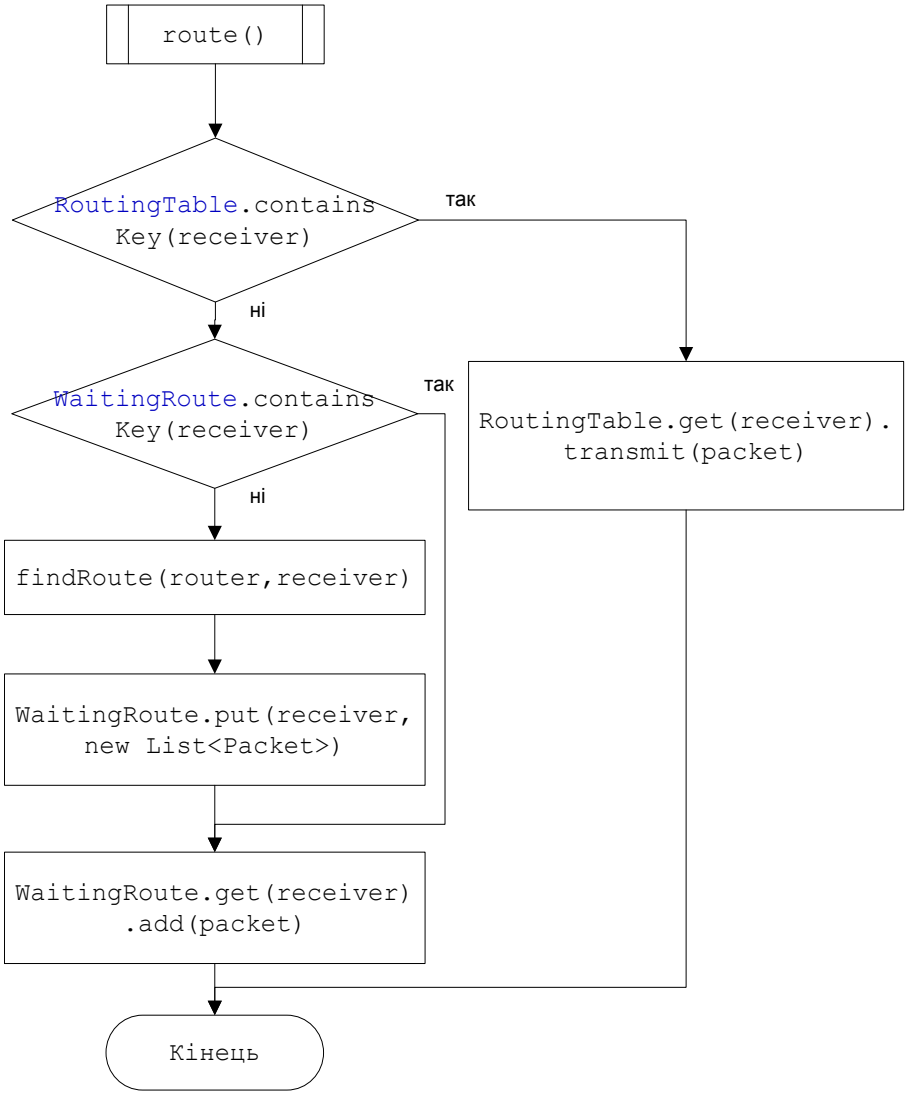


Рисунок 3.6 – Блок-схема алгоритму методу «route».

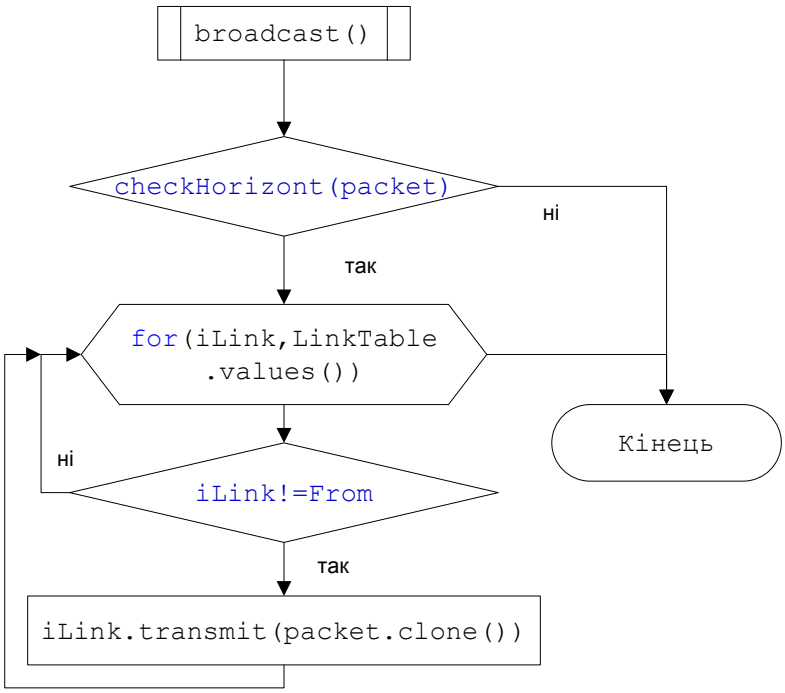


Рисунок 3.7 – Блок-схема алгоритму методу «broadcast».

Метод route перевіряє таблицю маршрутизації на наявність запису про місце призначення пакету і, якщо такий існує, відправляє пакет за вказаним зв'язком. Таблиця маршрутизації, як і більшість допоміжних таблиць у програмі, представлена у вигляді асоціативного списку із хеш-пошуком. Ключем списку є вузол призначення, тому пошук необхідного маршруту виконується достатньо швидко. Запис таблиці маршрутизації та допоміжних таблиць описуються наступними класами (рис.3.8):

- Route описує звичайний маршрут, що зберігається у таблиці і характеризується дескриптором зв'язку та метрикою;
- MAODVRoute розширює клас Route. Тому він зберігається у тій же таблиці і використовується розробленим алгоритмом маршрутизації. Цей клас описує додаткові параметри маршруту;
- Transit є записом таблиці транзитів вузла, що також використовується у службових пакетах імплементації розробленого алгоритму. Цей клас використовується розробленим алгоритмом для додаткового аналізу відповідей маршруту.

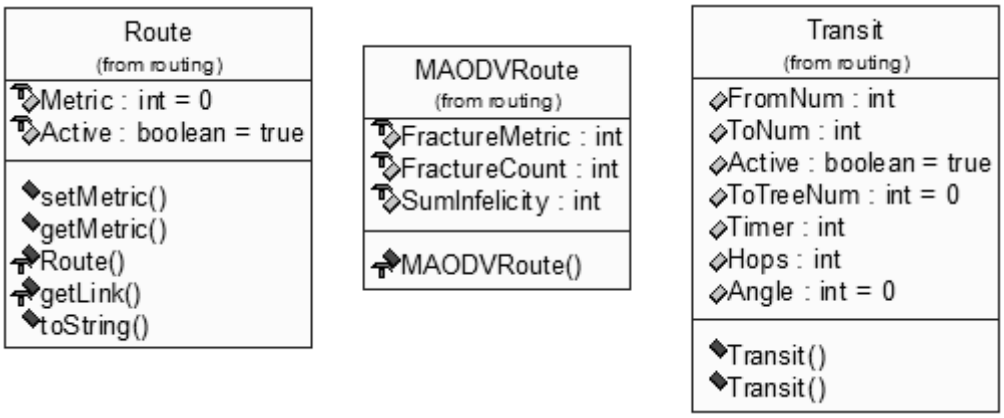


Рисунок 3.8 – Класи, що описують маршрути.

Також у цьому класі описані абстрактні методи, що повинні реалізовуватися у конкретних протоколах (тобто базовому і розробленому):

- receive визивається при отриманні службового пакету
- proceed визивається при проходженні службового пакету
- findRoute визивається якщо необхідно встановити новий маршрут

Метод findRoute використовується методом route у випадку, коли не знайдено маршрут для пакету. У такому разі сам пакет тимчасово зберігається у черзі WaitingRoute і відсилається при доданні необхідного маршруту у таблицю маршрутизації.

Від класу RoutingProtocol наслідується клас AODV (рис.3.9.), що імплементує базовий алгоритм. У цьому класі доповнено метод route відсилкою повідомлення про помилку та реалізвані методи receive, proceed і findRoute. Аналогічно клас MAODV, що описує розробений алгоритм розширює клас AODV.

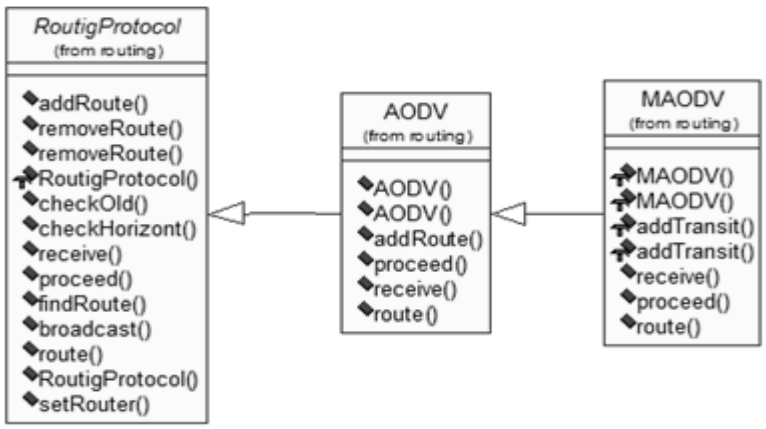


Рисунок 3.9 – Діаграма класів, що описують протоколи.

Обидва класи використовують спеціальні типи для службових пакетів. Усі вони розширюють клас RoutingPacket, що унаслідував від класу Packet наступні методи:

- getSender і getReceiver видають дескриптори відправника і приймальника пакету
- decTTL зменшує TTL пакету при кожному визову методу route
- drop ігнорує пакет, якщо у того закінчився TTL або якщо необхідний для пакету маршрут не знайдено
- finaliz визивається коли пакет дійшов до цілі
- clone создає точну копію пакету при широкомовленні

Клас RoutingPacket є абстрактним і не реалізує жодного методу. Він використовується для перевірки типу у методі route. Якщо пакет реалізує цей клас, його обробляють методи proceed або receive.

У класі AODVPacket, що розширює клас RoutingPacket, додана можливість використання послідовних номерів пакетів, їх відправників та приймачів. Це необхідно для того щоб ігнорувати застарілі пакети або маршрути. Для кожного нового пакету генерується новий номер, що не співпадає з іншими номерами пакетів з тим же пунктом призначення. різниця між пакетами службовими пакетами AODV та MAODV полягає у тому що у розробленому алгоритмі використовуються транзити. У програмі реалізовані наступні типи пакетів (рис 3.10):

- AODVRREQ та MAODVRREQ призначені для розсилки широкомовного запиту маршруту, але MAODVRREQ відсилається з меншим TTL і від проміжного вузла, а не від ініціатору
- AODVRREP та MAODVRREP призначені для посилки відповіді на запит маршруту, а також для оновлення номеру маршруту при корекції і сповіщення про переміщення вузла
- AODVERR призначено для повідомлення про відсутність маршруту у випадку коли неможливо відправити пакет за допомогою route (рис.3.11).

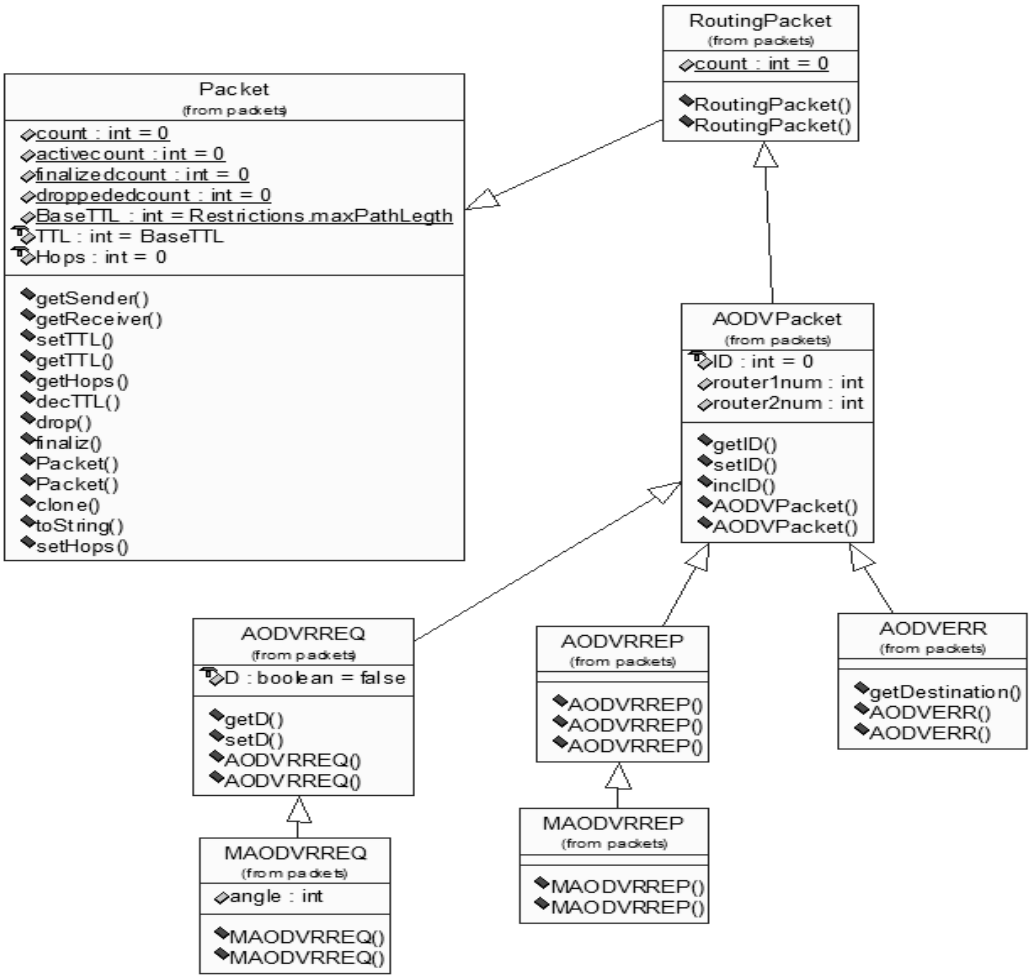


Рисунок 3.10 – Діаграма класів, що описують службові пакети.

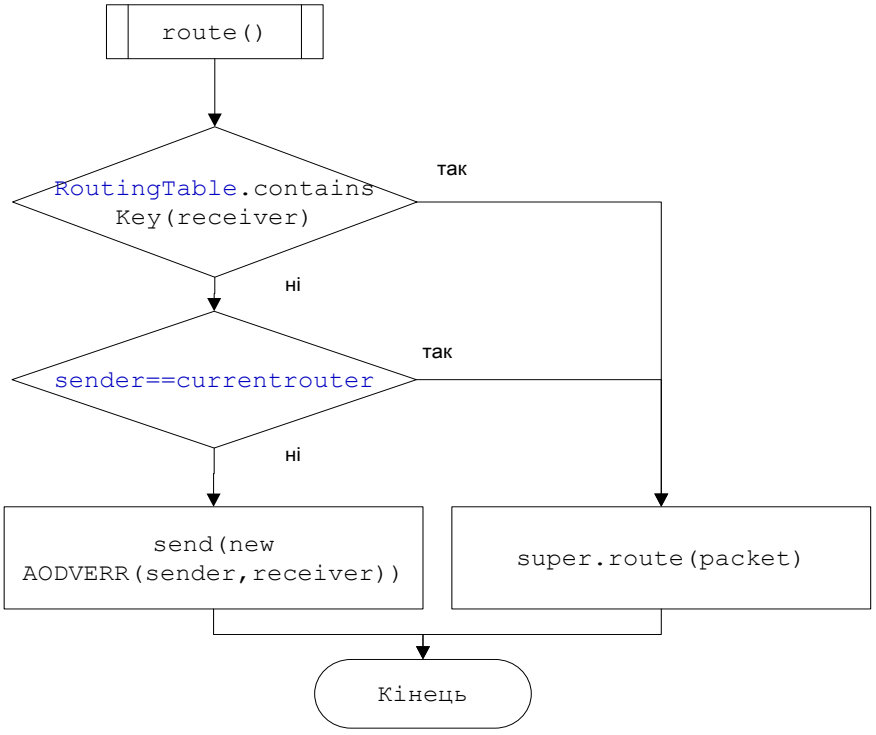


Рисунок 3.11 – Блок-схема алгоритму методу «route» класу AODV.

При визові методів proceed і receive відправник пакету додається у таблицю маршрутизації. Метод proceed протоколу AODV при проходженні пакету RREQ перевіряє таблицю маршрутизації на наявність запису про розшукує мий маршрут. Якщо запис знайдено і прапорець «D» у пакеті не встановлено, посилається відповідь RREP з початковим значенням поля Hops рівним метриці маршруту (рис 3.12). Але у першу чергу цей метод перевіряє, чи не отримувався вже пакет цей пакет RREQ за допомогою аналізу його номеру послідовності функцією checkHorizont.

Метод receive у базовому протоколі поводиться аналогічно методу proceed, тільки не перевіряє прапорець «D» і початкове значення поля Hops дорівнює нулю (рис.3.13).

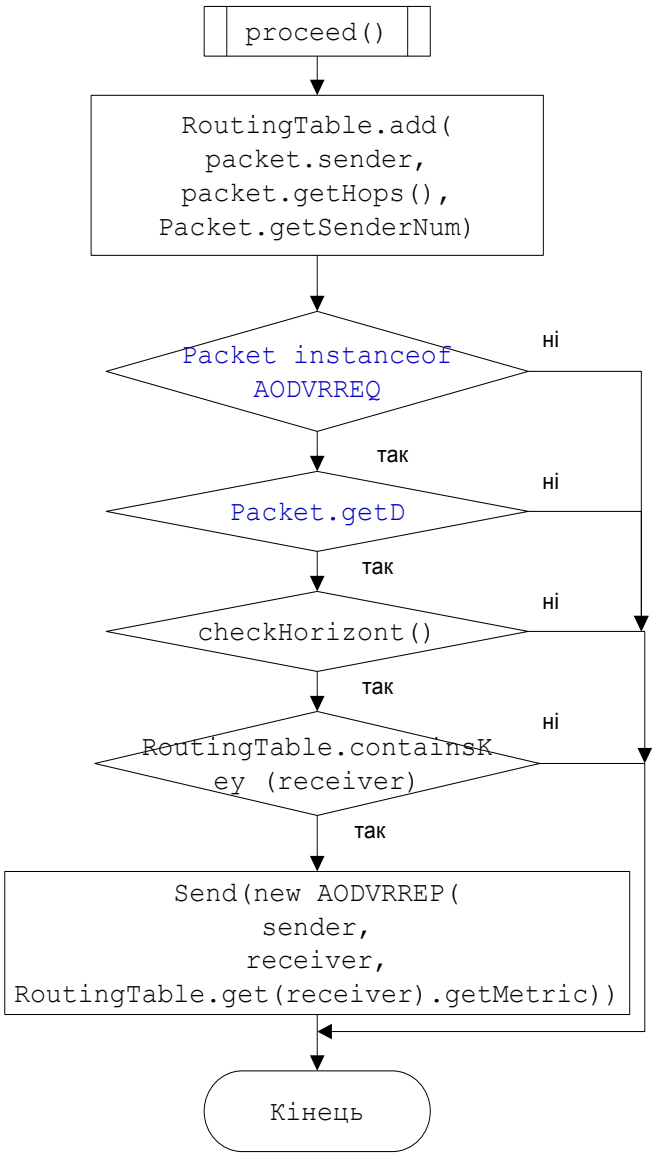


Рисунок 3.12 – Блок-схема алгоритму методу «proceed» класу AODV.

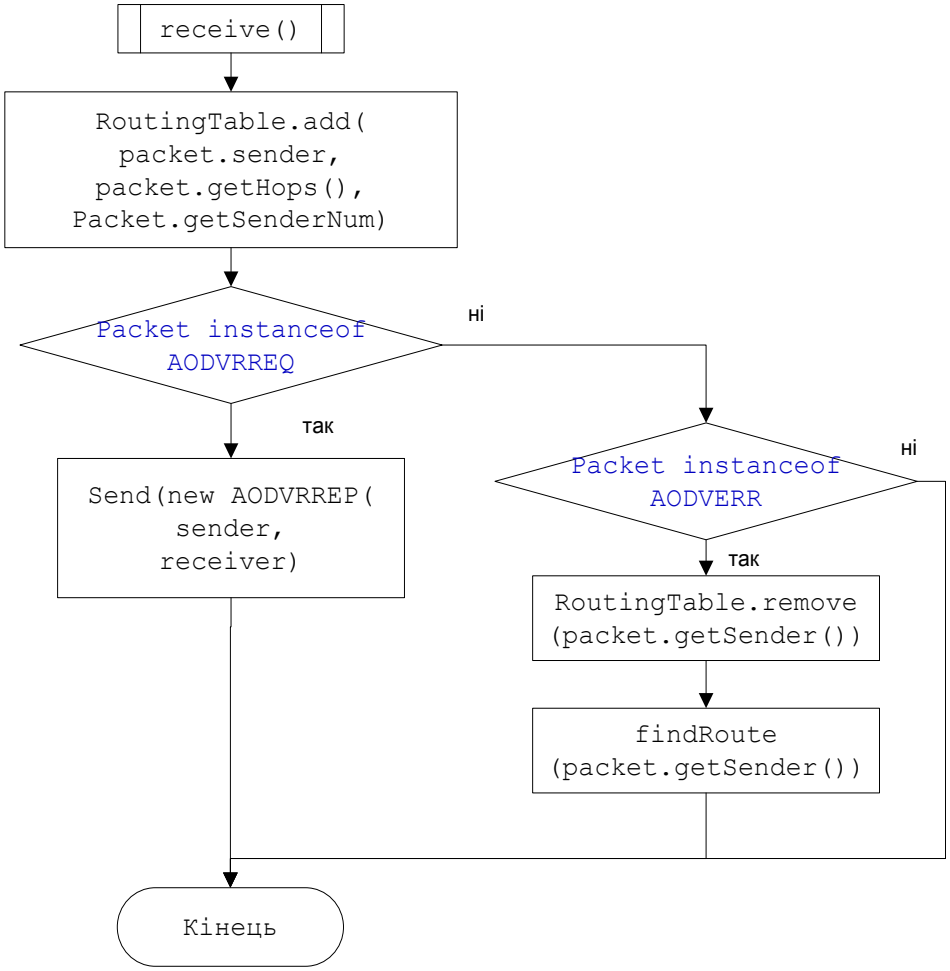


Рисунок 3.13 – Блок-схема алгоритму методу «receive» класу AODV.

Метод route розробленого алгоритму розширюючи базовий метод route додатково перевіряє на відсутність маршруту до вузла. У разі якщо на відсутній маршрут посилається таблиця транзитів, проводиться спроба відновити маршрут за допомогою розсилки пакету MAODVRREQ (рис.3.14). Пакет, замість того щоб загубитися, додається у чергу очікування маршруту.

У всіх інших випадках визивається метод route суперкласу. Слід відмітити, що метод route класу AODV не має можливості відіслати повідомлення про помилку, тому воно відсилається механізмом повторної відсилки пакетів запиту маршруту, якщо той використав ліміт кількості спроб відновлення.

На усіх рівнях методу route використовується одна й та ж черга пакетів, що очікують маршрут. Це означає, що пакет може бути відіслано ще до того як повернеться відповідь, якщо буде додано маршрут до роутеру при отриманні або пересилці іншого пакету.

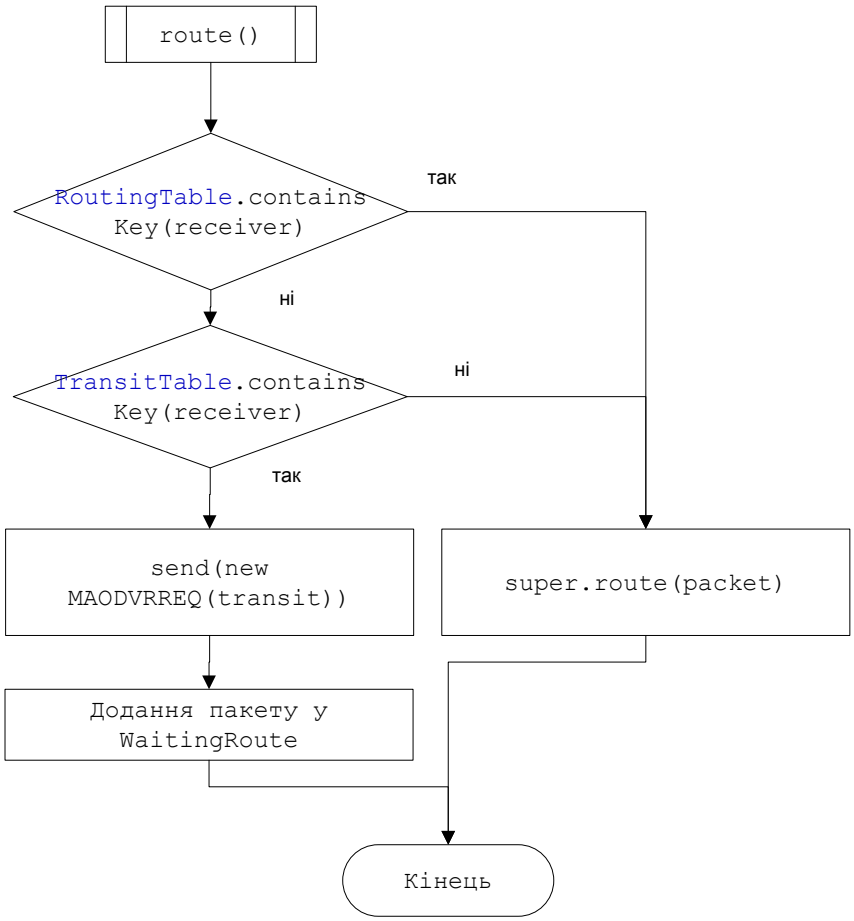


Рисунок 3.14 – Блок-схема алгоритму методу «route» класу MAODV.

Методи receive (рис 3.15) та proceed (рис 3.16.) класу MAODV перехоплюють наступні типи службових пакетів:

- пакети запиту при репарації MAODVRREQ, що б відіслати відповіді уразі наявності транзиту у таблиці транзитів.
- пакети відповіді MAODVRREP , що б репарувати ділянку маршруту
- пакети відповіді AODVRREP, що б зробити відмітку про новий транзит

Різниця у поведінці методів полягає у тому що метод proceed додатково перевіряє пакет на повторну пересилку, а також перевіряє чи знаходиться він зправа від ланки, що ініціювала репарацію. Перевірка проводиться за допомогою порівняння полів Hops запису транзиту пакету і запису у таблиці транзитів. Фактично це поле відображає номер ланки у маршруті. Метод receive додатково перевіряє оптимальність маршруту порівнюючи різницю між номерами ланок із відстанню до вузла.

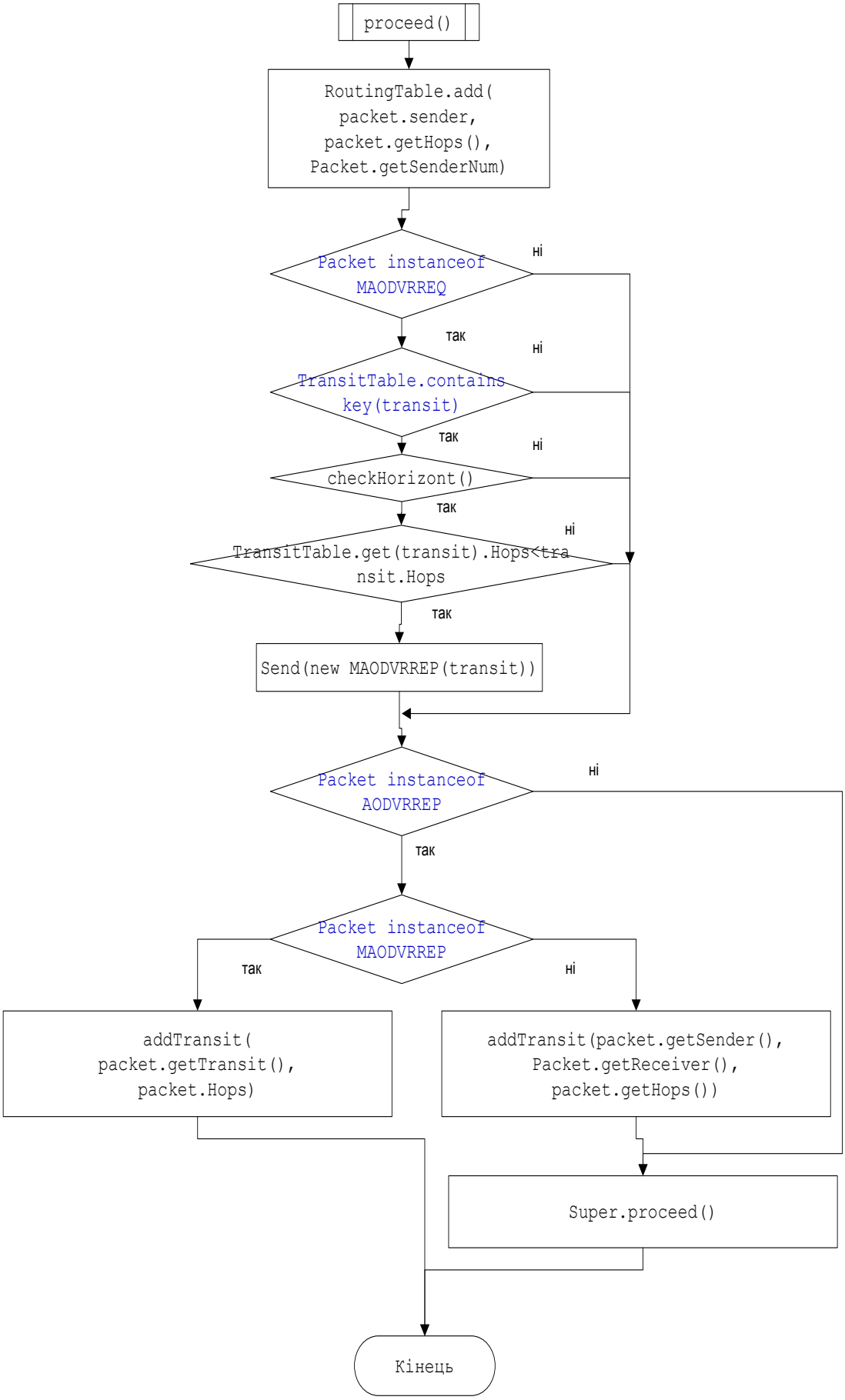


Рисунок 3.15 – Блок-схема алгоритму методу «proceed» класу MAODV.

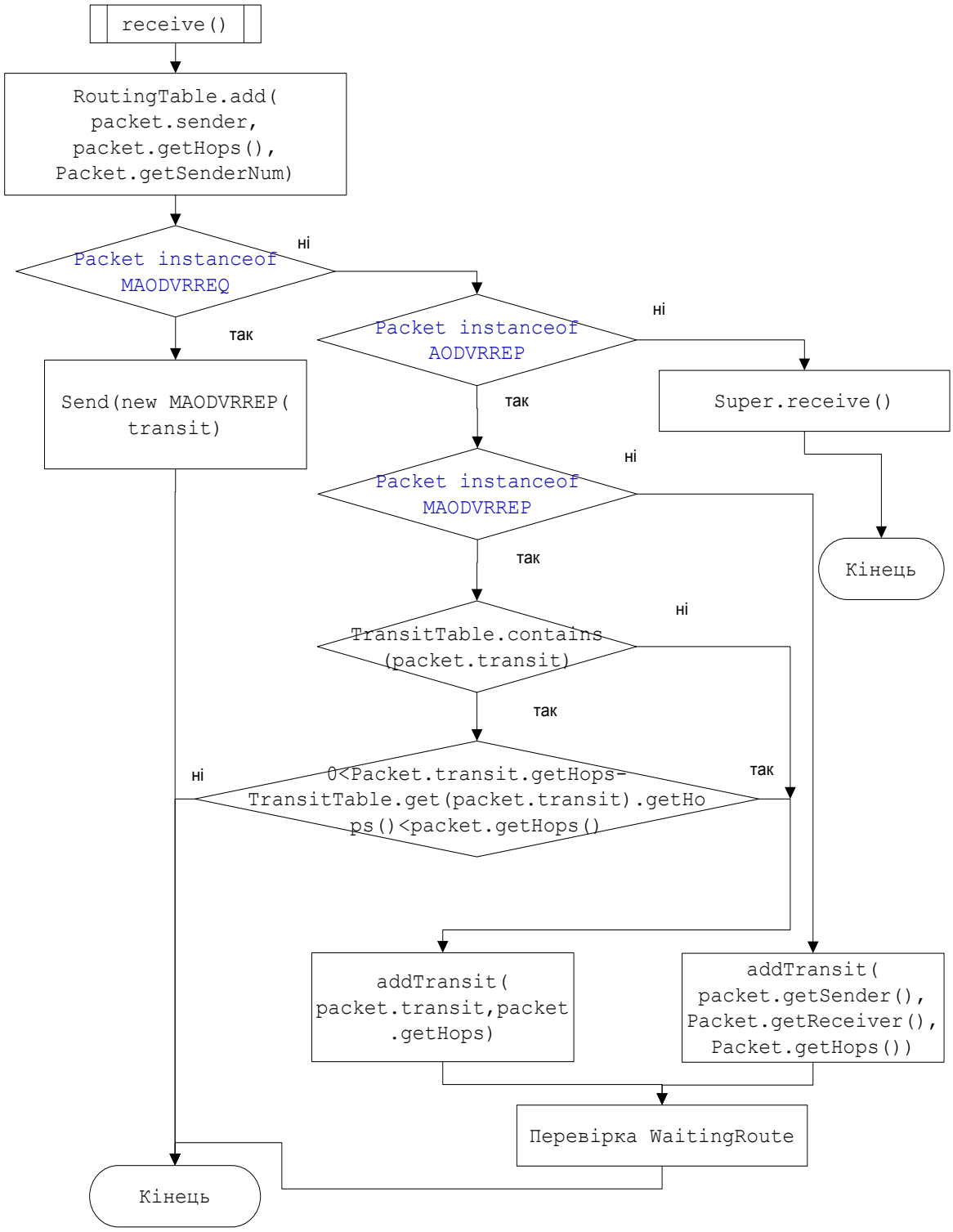


Рисунок 3.16 – Блок-схема алгоритму методу «receive» класу MAODV.

Механізм видалення застарілих записів таблиць маршрутизації у програмі не реалізовано, бо він не має значного впливу на моделювання окремих випадків. Оновлення номеру вузла виконується при кожному видаленні запису з нульовою метрикою із таблиці маршрутизації (тобто при

втраті зв’язку). Таким чином механізм нумерації вузлів дозволяє автоматично ігнорувати застарілі маршрути за рахунок перевірки номерів вузлів у службових пакетах, що у більшості випадків (достатніх для порівняння алгоритмів) компенсує відсутність видалення записів за таймером.

У програмі реалізовано механізм повторної відсилки пакетів запиту у разі, якщо пакет відповіді не приходить деякий час. До того ж при відсутності пакету відповіді TTL нового пакету запиту збільшується, реалізуючи таким чином механізм Echanged Ring Search. Метод route також може відсилати замість AODVERR пакет RREQ з TTL, що дорівнює його метриці до вузла призначення, реалізуючи механізм Local Repair.

Висновки

Розроблене моделююче програмне забезпечення дозволяє достатньо точно промоделювати поведінку алгоритмів маршрутизації для того, що б можна було отримати необхідні характеристики і порівняти ефективність алгоритмів. Візуалізація основних процесів пов’язаних із маршрутизацією дозволяє не тільки проводити заміри, але й перевіряти коректність поведінки розробленого алгоритму. Також розроблена програма підтримує можливість «гарячого» редагування топології безпосередньо у процесі моделювання та контроль швидкості моделювання, що дозволяє досліджувати реакції алгоритмів на зміну структури мережі.

Источники: ПЗ магистерской работы