Зміст

ВСТУП

Метою даної курсової роботи було написання програмного комплексу, який б дозволяв наочне створення і конфігурування комп’ютерної мережі довільної топології.

Комп'ютерна мережа — система зв'язку між двома чи більше комп'ютерами. В ширшому розумінні комп'ютерна мережа — це система зв'язку через кабельне чи повітряне середовище, самі комп'ютери різного функціонального призначення і мережеве обладнання. Для передачі інформації можуть бути використані різні фізичні явища, як правило — різні види електричних сигналів чи електромагнітного випромінювання.

Середовищами передавання у комп'ютерних мережах можуть бути телефонні кабелі, коаксіальні кабелі, виті пари та волоконно-оптичні кабелі.

Мережі – найпоширеніший спосіб обміну даними, тому правильна організація мережі з метою підвищення ефективності її роботи є досить важливим завданням. Досягти цієї мети неможливо без створення моделі мережі на етапі її проектування та проведення досліджень на цій моделі. Для створення моделі можуть застосовуватися як аналітичні так і імітаційні методи дослідження. В даній роботі пропонується імітаційна модель та її реалізація, які дозволяють проводити дослідження функціонування систем передачі даних на четвертому (транспортному) рівні еталонної моделі OSI. Аналізується передача даних з різними параметрами та різної конфігурації. Програмний продукт дає можливість заздалегідь визначити ефективність мережі, а саме її слабкі місця, що допоможе зекономити час та кошти на придбання неефективного обладнання.

Наочна модель мережі є інтуїтивно зрозумілою і забезпечує зменшення часу процесу моделювання.

1. Постановка задачі

Розробити програмне забезпечення, яке моделює процеси маршрутизації у мережі передачі даних. Ця система повинна мати лише досить базові параметри конфігурації, та підтримувати лише один режим маршрутизації – метод каталогів, орієнтованих на сеанс, але все одно дозволяти досліджувати реальні мережі. Метод маршрутизації є цетнралізованим. Супервізор, який вручну обирається серед існуючих в мережі, повинен зберігати загальну таблицю маршрутизації для всіх вузлів, мінімальні та альтернативні шляхи. Мінімальні шляхи використовуються для імітації передачі даних з попереднім встановленням логічного з'єднання, альтернативні – без встановлення з'єднання, в дейтаграмному режимі. Алгоритм Дейкстра використовується для пошуку шляхів, як мінімальних, так і альтернативних.

2. Основні теоретичні відомості

Мережа передачі даних являє собою сукупність вузлів, зв’язаних каналами передачі даних. В кожен вузол може бути як джерелом так і приймачем повідомлень, що розбиваються на пакети в вузлі-джерелі та збираються в повідомлення в вузлі-приймачі. Також він виконує передачу до наступного вузла пакетів, що поступили через вхідні канали чи були введені безпосередньо в вузол. Наступний вузол в маршруті передачі визначається таблицею маршрутизації. Процес визначення маршруту називається маршрутизацією, а алгоритми визначення маршруту називаються алгоритмами маршрутизації. Обладнання, що виконує маршрутизацію називаються маршрутизатором.

Таблиця маршрутизації містить відстані від даного вузла до вузлів через різні вихідні канали зв’язку. Відстані можуть враховувати довжину каналу, завантаженість каналу та інше.

Статичні алгоритми маршрутизації використовують таблиці маршрутизації, що створюються системним адміністратором і, як правило, не змінюються під час роботи мережі. Такі алгоритми не враховують динамічних показників системи, таких як відмова та завантаженість каналів зв’язку. Але дані алгоритми не вимагають передачі службової інформації через мережу. Також можливе динамічне врахування завантаженості вихідних каналів даного вузла, що покращує роботу маршрутизатора.

Динамічні алгоритми з деяким періодом оновлюють таблиці маршрутизації. Маршрутизатори, що використовують дані алгоритми обмінюються інформацією про відстані та таким чином враховують завантаженість мережі. Але це може призвести до використання значного обсягу пропускної спроможності під службові повідомлення.

Вузли мережі передачі даних можуть бути зв'язані двома типами каналів: дуплексним та напівдуплексним. Дуплексні канали дозволяють

одночасно передавати та приймати сигнали на різних частотах. В напівдуплексних каналах, на відміну від дуплексних, передача ведеться по одному й тому ж каналу зв’язку в обох напрямках, тобто в кожен момент часу передача ведеться тільки в одному напрямку. Також канали мережі передачі даних можуть бути супутниковими. В такому випадку затримка розповсюдження сигналу в каналі може становити від 240 до 400мс.

Tymnet – міжнародна мережа передачі даних, що використовує технологію комутації віртуальних пакетів та інтерфейси X.25, SNA/SDLC, ASCII, BSC для підключення до серверів. Користувачі зазвичай підключаються до цієї мережі за допомогою комутованих або асинхронних з’єднань. Мережа Tymnet має в наш час 450 комутаційних центрів, надає доступ до 175 регіонів США, довжина її орендованих ліній складає 125 тис. миль. Мережа призначена в основному для обслуговування користувачів в інтерактивному режимі та широко використовується американськими ученими і фахівцями для вирішення наукових і технічних завдань. Мережа об'єднує комп'ютери різноманітного типу – IBM, PDP, Univak, CDC. Вона розрахована на підключення як великого числа низькошвидкісних терміналів, що працюють зі швидкістю 300 біт/с, так і орендованих, де швидкість допускається до 9600 біт/с. Середня швидкість передачі даних в мережі Tymnet – 2000 біт/с. У години пік у мережі Tymnet працює близько 500 термінальних концентраторів. Значення MTU (maximum transmission unit), тобто максимальних розмір пакету, визначається за стандартом відповідного інтерфейсу. Враховуючи те, що мережа Tymnet використовує інтерфейс X.25, значення MTU – 576 байт.

3. Опис алгоритму

В мережі Tymnet алгоритм маршрутизації є централізованим та реалізується в деякому спеціальному вузлі – супервізорі. Використовуються логічні зв’язки, і маршрут встановлюється тільки в момент встановлення логічного зв’язку. Супервізор, отримавши запит на встановлення зв’язку, призначає найкоротших шлях, що з’єднує вузол-відправник з вузлом-адресатом даного логічного зв’язку. Супервізор слідкує за поточними значеннями довжин всіх ліній та розраховує найкоротші шляхи. Довжина кожної лінії може залежати від різних характеристик, наприклад пропускної здатності, типу зв’язку, для якого обирається маршрут. У цього алгоритму нема тенденції до коливання, оскільки використовуються логічні зв’язки. Розмір службового пакету – 20байт.

Для передачі даних в дейтаграмному режимі використовується модифікація заданого алгоритму, адже в дейтаграмному режимі попереднє з’єднання не встановлюється. Вузол-відправник звертається до супервізора, який містить таблицю альтернативних шляхів з метою отримання шляхів передачі пакетів. Супервізор надає необхідну інформацію, якщо такі шляхи є. Пакети відправляються вузлу-адресату по різних маршрутах та сортуються при досягненні цільового вузла. Розмір службового пакету – 8 байт. Таким чином зберігається цілісність повідомлення, якщо це можливо. Проте дейтаграмний режим не гарантує цілісність повідомлень, адже працює без встановлення попереднього з’єднання. Таким чином при режимі із встановленням логічного з’єднання передача даних є більш повільною, але надійною. При дейтаграмному режимі – навпаки, цілісність даних не гарантується, але швидкість передачі є значно вищою.

Для пошуку маршрутів передачі даних використовується алгоритм Дейкстра. Алгоритм Дейкстра — алгоритм на графах, відкритий Дейкстрою. Знаходить найкоротший шлях від однієї вершини графа до всіх інших вершин.

Класичний алгоритм Дейкстра працює тільки для графів без циклів від'ємної довжини.

Як біло зазначено, алгоритм Дейкстра призначений для пошуку маршруту найкоротшої довжини від заданої вершини графа до всіх інших. В процесі виконання даного алгоритму при переході від вершини *i* до вершини *j* використовується спеціальна процедура, яка кожній вершині графа присвоює відповідну мітку.

Позначимо через u_i найкоротшу відстань від початкової вершини 1 до вершини *i*, через d_{ij} — довжину ребра *(i, j)*. Тоді для вершини *j* мітка $[u_j, i]$ визначається наступним чином: $[u_j, i] = [u_i + d_{ij}, i]$, $d_{ij} >= 0$.

Мітки в алгоритмі Дейкстри можуть бути двох типів: тимчасові або постійні. Тимчасова мітка може бути замінена на нову також тимчасову, якщо в процесі виконання алгоритму буде знайдено більш короткий маршрут до даної вершини. Якщо ж в процесі виконання алгоритму виявиться, що більш короткого маршруту від початкової до даної вершини не існує, то статус тимчасової мітки змінюється на постійну.

Розв'язок за алгоритмом Дейкстри складається з наступних етапів:

1. Початковій вершині (вершина №1) присвоюється постійна мітка *[0,-] , i=1*.
2. Для всіх вершин *j*, які поєднані ребром з вершиною *i*, і не мають міток, статус яких становить «постійна», обчислюються тимчасові мітки $[u_i + d_{ij}, i]$. Якщо для вершини *j* вже існує деяка мітка $[u_j, k]$, то слід провірити виконання умови $u_i + d_{ij} < u_j$. Виконання даної умови говорить про те, що необхідно мітку $[u_j, k]$ замінити на $[u_i + d_{ij}, i]$.

Даний процес продовжується до тих пір, поки всім вершинам графа не буде присвоєно мітку зі статусом постійна.

Для пошуку найкоротшого шляху за кількістю транзитних ділянок на вхід алгоритму Дейкстри подається граф, вага всіх ребер якого дорівнює 1. Таким чином найкоротшим буде той маршрут, який використовує найменшу кількість вершин.

Для пошуку всіх шляхів в графі на етапі 2 умова $u_i + d_{ij} < u_j$ не перевіряється, а знайдений шлях запам'ятовується у списку шляхів від вершини *i* до *j*.

Час передачі повідомлень розраховується в залежності від ваги лінії, швидкості передачі, затримки в буфері вузлів, кількості пакетів та їх ваги. Вага лінії, відповідно і її довжина, в даному випадку означає час передачі пакету по даному каналу із заданою швидкістю. Таким чином пошук найкоротших шляхів є виправданим.

4. Опис розробленої програми

4.1 Призначення та структура розробленого програмного продукту

Розроблений програмний продукт моделює процеси маршрутизації у мережі передачі даних. Ця система має базові параметри конфігурації, та підтримує один режим маршрутизації – метод каталогів, орієнтованих на сеанс. Призначення розробленого програмного забезпечення - дослідження передачі даних в комп'ютерній мережі з різними характеристиками.

Структура програмного продукту наведена в додатку 1.

4.2 Опис класів, що реалізовано в програмі

1) class AddNodeMouseListener – додає вузол у мережу
2) class ClearMouseListener – видаляє мережу
3) class Connection – описання каналу зв'язку
4) enum ConnectionType – перелік типів каналу зв'язку
5) class ConnectMouseListener – додає канал зв'язку в мережу
6) class DefaultTopology – запам'ятовує топологію мережі
7) class DeleteConnectionMouseListener – видаляє зв'язок між указаними вузлами
8) class DeleteNodeMouseListener – видаляє вузол
9) class DijkstraAlgorithm - реалізує алгоритм Дейкстри (Додаток2)
10) class LoadTopologyListener – відображає задану в класі DefaultTopology топологію мережі
11) class MoveNodeMouseListener – пересуває вузли
12) class PaintTopology – відображає поточну топологію
13) class SaveMouseListener – зберігає поточну топологію мережі
14) class SelectSupervisorMouseListener – обирає супервізор

15) class SendMessageMouseListener – відправляє повідомлення
16) class TymnetRouting – виконує маршрутизацію заданим методом
17) class NetworkGUI – створює загальний графічний інтерфейс
18) class NodeInfoGUI – створює графічний інтерфейс для відображення інформації на вузлах

4.3 Опис графічного інтерфейсу.

Графічний інтерфейс розробленої системи представлений на рис. 1. Поля та кнопки графічного інтерфейсу описані в таблиці 1

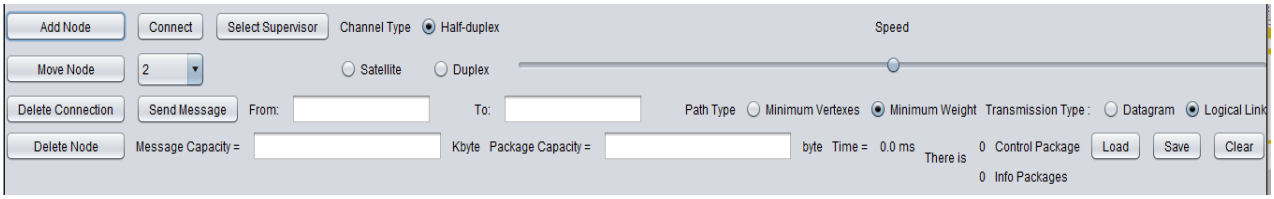
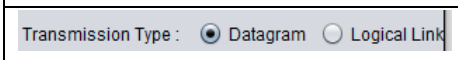
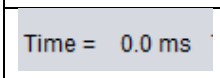
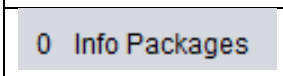
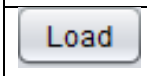
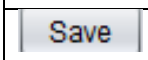
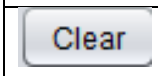


Рис. 1 Графічний інтерфейс програмної системи

Таблиця 1

Значення полів та кнопок графічного інтерфейсу

Поле, кнопка	Призначення
Add Node	Додати вузол в обраній точці
Move Node	Пересунути обраний вузол
Connect	З'єднати обрані вузли
Channel Type Half-duplex Satellite Duplex	Встановити тип каналу
2	Обрати вагу каналу серед наявних
Delete Connection	Видалити зв'язок між обраними вузлами
Delete Node	Видалити обраний вузол
Select Supervisor	Встановити супервізор
Send Message	Відправити повідомлення

	Вказати вузол-відправник із наявних в топології
	Вказати вузол-адресат із наявних в топології
	Вказати розмір повідомлення
	Вказати розмір інформаційних пакетів
	Обрати режим передачі
	Спосіб пошуку мінімального шляху
	Час відправки повідомлення
	Кількість службових пакетів
	Кількість інформаційних пакетів
	Відобразити топологію за замовченням
	Зберегти поточну топологію
	Видалити поточну топологію

4.4 Таблиці маршрутів та відстаней в супервізорному вузлі мережі передачі даних.

Досліджувана топологія мережі передачі даних складається з 30 вузлів, один з яких (за вибором) є супервізором; каналів зв'язку, вага яких обирається випадковим чином з діапазону 2, 4, 5, 7, 8, 12, 15, 17, 18, 22, 25, 32. 3 канали – супутникові. (рис. 2)

При виборі супервізора в ньому генеруються дві таблиці: таблиця мінімальних маршрутів (рис. 3, рис. 4) та таблиця відстаней від кожного вузла до кожного (рис. 5). При дейтаграмному режимі передачі даних таблиця маршрутів модифікується. Відображаються не тільки мінімальні, а й альтернативні шляхи (рис. 6).

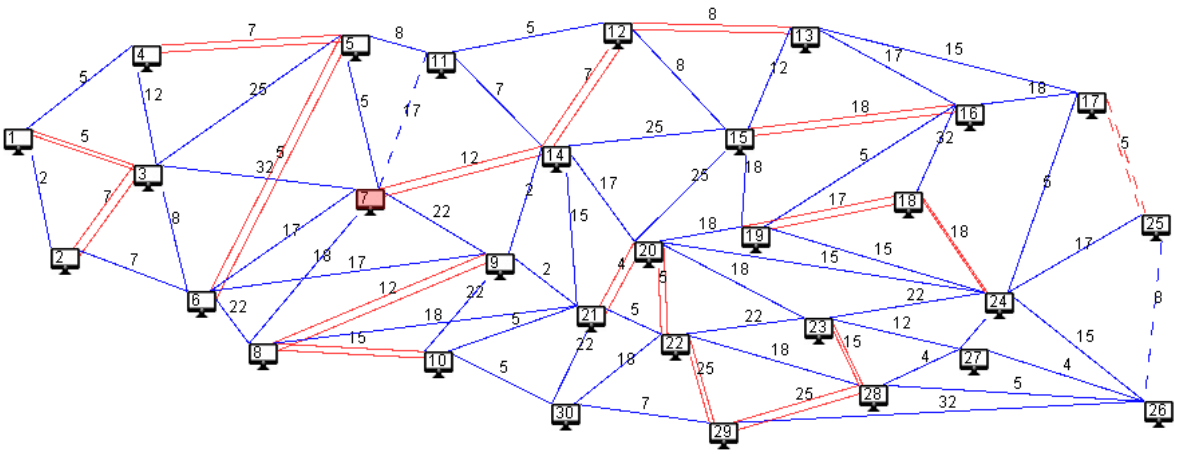


Рис. 2 Комп’ютерна мережа, що досліджується

Маршрути розраховуються від кожного вузла до кожного, кількість вузлів – 30, відповідно кількість рядків в таблиці маршрутизації – 900. На рис.3 представлені початкові значення таблиці мінімальних маршрутів, на рис.4 – кінцеві

Node Info

Weights | Paths

From<>To	Path
1<>1	no path
1<>2	[1, 2]
1<>3	[1, 3]
1<>4	[1, 4]
1<>5	[1, 4, 5]
1<>6	[1, 2, 6]
1<>7	[1, 4, 5, 7]
1<>8	[1, 2, 6, 8]
1<>9	[1, 2, 6, 9]
1<>10	[1, 2, 6, 9, 21, 10]
1<>11	[1, 4, 5, 11]
1<>12	[1, 4, 5, 11, 12]
1<>13	[1, 4, 5, 11, 12, 13]
1<>14	[1, 4, 5, 11, 14]
1<>15	[1, 4, 5, 11, 12, 15]
1<>16	[1, 4, 5, 11, 12, 13, 16]
1<>17	[1, 4, 5, 11, 12, 13, 17]
1<>18	[1, 2, 6, 9, 21, 20, 24, 18]
1<>19	[1, 2, 6, 9, 21, 20, 19]
1<>20	[1, 2, 6, 9, 21, 20]
1<>21	[1, 2, 6, 9, 21]
1<>22	[1, 2, 6, 9, 21, 22]
1<>23	[1, 2, 6, 9, 21, 20, 23]
1<>24	[1, 2, 6, 9, 21, 20, 24]
1<>25	[1, 4, 5, 11, 12, 13, 17, 25]
1<>26	[1, 2, 6, 9, 21, 22, 28, 26]
1<>27	[1, 2, 6, 9, 21, 20, 24, 27]
1<>28	[1, 2, 6, 9, 21, 22, 28]
1<>29	[1, 2, 6, 9, 21, 10, 30, 29]
1<>30	[1, 2, 6, 9, 21, 10, 30]
2<>1	[2, 1]
2<>2	no path
2<>3	[2, 3]
2<>4	[2, 1, 4]
2<>5	[2, 6, 5]
2<>6	[2, 6]
2<>7	[2, 6, 5, 7]
2<>8	[2, 6, 8]

Рис. 2 Таблиця мінімальних маршрутів. Початкові значення

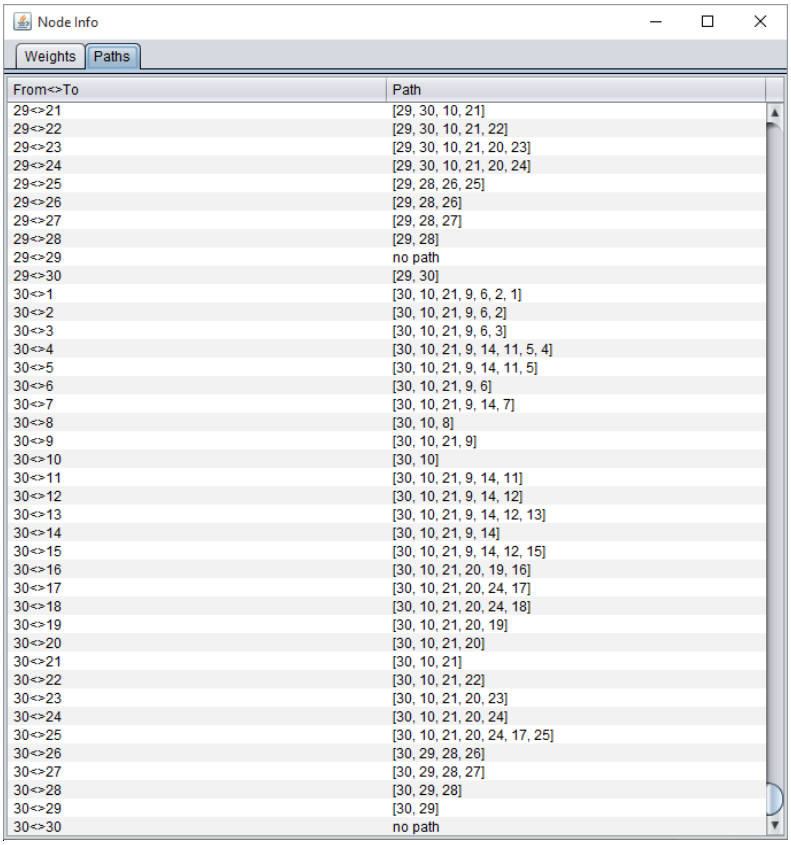
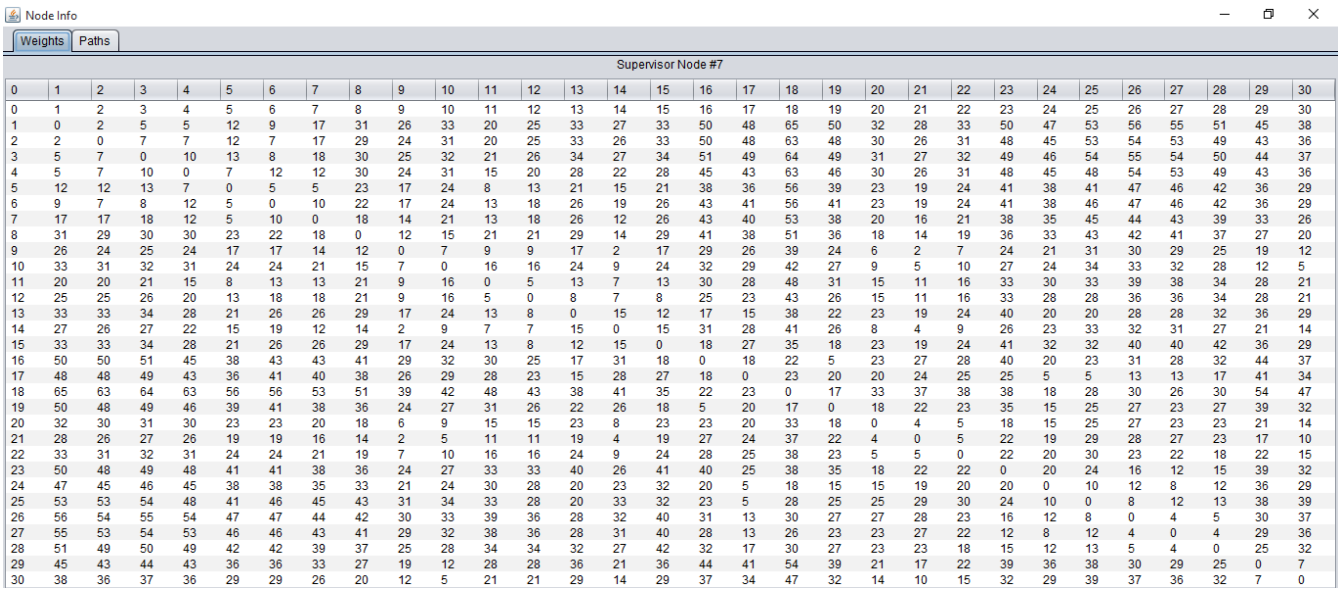


Рис. 4 Таблиця мінімальних маршрутів. Кінцеві значення

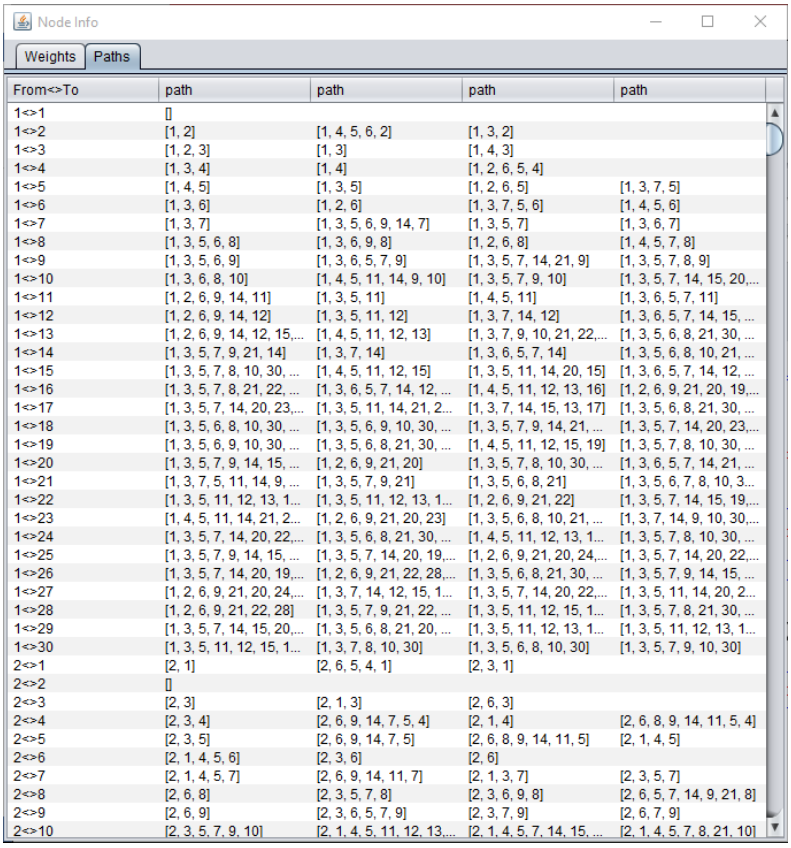
Відстані від кожного вузла до кожного розраховуються на основі ваги ліній. Таблиця відстаней представлена на рис. 5. Перший рядок та стовпець таблиці – номер вузла з/до якого розраховується відстань. Наприклад, на перетині третього рядка з другим стовпцем отримаємо відстань від третього вузла до другого.

Таблиця альтернативних маршрутів відображає можливі маршрути від кожного вузла до кожного, в даному випадку має розмір 900x4. На рис. 6 та рис. 7 представлені початкові та кінцеві значення цієї таблиці.



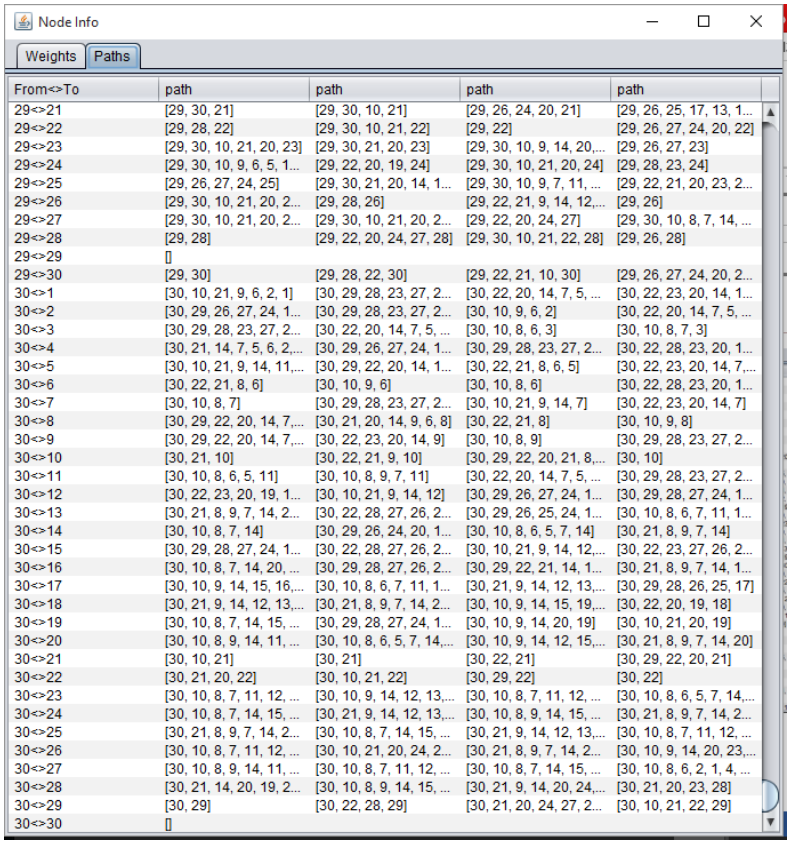
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	2	5	5	12	9	17	31	26	33	20	25	33	27	33	50	48	65	50	32	28	33	50	47	53	56	55	51	45	38
2	2	0	7	7	12	7	17	29	24	31	20	25	33	26	33	50	48	63	48	30	26	31	48	45	53	54	53	49	43	36
3	5	7	0	10	13	8	18	30	25	32	21	26	34	27	34	51	49	64	49	31	27	32	49	46	54	55	54	50	44	37
4	5	7	10	0	7	12	12	30	24	31	15	20	28	22	28	45	43	63	46	30	26	31	48	45	48	54	53	49	43	36
5	12	12	13	7	0	5	5	23	17	24	8	13	21	15	21	38	36	56	39	23	19	24	41	38	41	47	46	42	36	29
6	9	7	8	12	5	0	10	22	17	24	13	18	26	19	26	43	41	56	41	23	19	24	41	38	46	47	46	42	36	29
7	17	17	18	12	5	10	0	18	14	21	13	18	26	12	26	43	40	53	38	20	16	21	38	35	45	44	43	39	33	26
8	31	29	30	30	23	22	18	0	12	15	21	21	29	14	29	41	38	51	36	18	14	19	36	33	43	42	41	37	27	20
9	26	24	25	24	17	17	14	12	0	7	9	9	17	2	17	29	26	39	24	6	2	7	24	21	31	30	29	25	19	12
10	33	31	32	31	24	24	21	15	7	0	16	16	24	9	24	32	29	42	27	9	5	10	27	24	34	33	32	28	12	5
11	20	20	21	15	8	13	13	21	9	16	0	5	13	7	13	30	28	48	31	15	11	16	33	30	33	39	38	34	28	21
12	25	25	26	20	13	18	18	21	9	16	5	0	8	7	8	25	23	43	26	15	11	16	33	28	28	36	36	34	28	21
13	33	33	34	28	21	26	26	29	17	24	13	8	0	15	12	17	15	38	22	23	19	24	40	20	20	28	28	32	36	29
14	27	26	27	22	15	19	12	14	2	9	7	7	15	0	15	31	28	41	26	8	4	9	26	23	33	32	31	27	21	14
15	33	33	34	28	21	26	26	29	17	24	13	8	12	15	0	18	27	35	18	23	19	24	41	32	32	40	40	42	36	29
16	50	50	51	45	38	43	43	41	29	32	30	25	17	31	18	0	18	22	5	23	27	28	40	20	23	31	28	32	44	37
17	48	48	49	43	36	41	40	38	26	29	28	23	15	28	27	18	0	23	20	20	24	25	25	5	5	13	13	17	41	34
18	65	63	64	63	56	56	53	51	39	42	48	43	38	41	35	22	23	0	17	33	37	38	38	18	28	30	26	30	54	47
19	50	48	49	46	39	41	38	36	24	27	31	26	22	26	18	5	20	17	0	18	22	23	35	15	25	27	23	27	39	32
20	32	30	31	30	23	23	20	18	6	9	15	15	23	8	23	23	20	33	18	0	4	5	18	15	25	27	23	23	21	14
21	28	26	27	26	19	19	16	14	2	5	11	11	19	4	19	27	24	37	22	4	0	5	22	19	29	28	27	23	17	10
22	33	31	32	31	24	24	21	19	7	10	16	16	24	9	24	28	25	38	23	5	5	0	22	20	30	23	22	18	22	15
23	50	48	49	48	41	41	38	36	24	27	33	33	40	26	41	40	25	38	35	18	22	22	0	20	24	16	12	15	39	32
24	47	45	46	45	38	38	35	33	21	24	30	28	20	23	32	20	5	18	15	15	19	20	20	0	10	12	8	12	36	29
25	53	53	54	48	41	46	45	43	31	34	33	28	20	33	32	23	5	28	25	25	29	30	24	10	0	8	12	13	38	39
26	56	54	55	54	47	47	44	42	30	33	39	36	28	32	40	31	13	30	27	27	28	23	16	12	8	0	4	5	30	37
27	55	53	54	53	46	46	43	41	29	32	38	36	28	31	40	28	13	26	23	23	27	22	12	8	12	4	0	4	29	36
28	51	49	50	49	42	42	39	37	25	28	34	34	32	27	42	32	17	30	27	23	23	18	15	12	13	5	4	0	25	32
29	45	43	44	43	36	36	33	27	19	12	28	28	36	21	36	44	41	54	39	21	17	22	39	36	38	30	29	25	0	7
30	38	36	37	36	29	29	26	20	12	5	21	21	29	14	29	37	34	47	32	14	10	15	32	29	39	37	36	32	7	0

Рис. 3 Таблиця відстаней від кожного вузла до кожного



From<>To	path	path	path	path
1<>1	[]			
1<>2	[1, 2]	[1, 4, 5, 6, 2]	[1, 3, 2]	
1<>3	[1, 2, 3]	[1, 3]	[1, 4, 3]	
1<>4	[1, 3, 4]	[1, 4]	[1, 2, 6, 5, 4]	
1<>5	[1, 4, 5]	[1, 3, 5]	[1, 2, 6, 5]	[1, 3, 7, 5]
1<>6	[1, 3, 6]	[1, 2, 6]	[1, 3, 7, 5, 6]	[1, 4, 5, 6]
1<>7	[1, 3, 7]	[1, 3, 5, 6, 9, 14, 7]	[1, 3, 5, 7]	[1, 3, 6, 7]
1<>8	[1, 3, 5, 6, 8]	[1, 3, 6, 9, 8]	[1, 2, 6, 8]	[1, 4, 5, 7, 8]
1<>9	[1, 3, 5, 6, 9]	[1, 3, 6, 5, 7, 9]	[1, 3, 5, 7, 14, 21, 9]	[1, 3, 5, 7, 8, 9]
1<>10	[1, 3, 6, 8, 10]	[1, 4, 5, 11, 14, 9, 10]	[1, 3, 5, 7, 9, 10]	[1, 3, 5, 7, 14, 15, 20,...
1<>11	[1, 2, 6, 9, 14, 11]	[1, 3, 5, 11]	[1, 4, 5, 11]	[1, 3, 6, 5, 7, 11]
1<>12	[1, 2, 6, 9, 14, 12]	[1, 3, 5, 11, 12]	[1, 3, 7, 14, 12]	[1, 3, 6, 5, 7, 14, 15, ...
1<>13	[1, 2, 6, 9, 14, 12, 15,...	[1, 4, 5, 11, 12, 13]	[1, 3, 7, 9, 10, 21, 22,...	[1, 3, 5, 6, 8, 21, 30, ...
1<>14	[1, 3, 5, 7, 9, 21, 14]	[1, 3, 7, 14]	[1, 3, 6, 5, 7, 14]	[1, 3, 5, 6, 8, 10, 21, ...
1<>15	[1, 3, 5, 7, 8, 10, 30, ...	[1, 4, 5, 11, 12, 15]	[1, 3, 5, 11, 14, 20, 15]	[1, 3, 6, 5, 7, 14, 12, ...
1<>16	[1, 3, 5, 7, 8, 21, 22, ...	[1, 3, 6, 5, 7, 14, 12, ...	[1, 4, 5, 11, 12, 13, 16]	[1, 2, 6, 9, 21, 20, 19,...
1<>17	[1, 3, 5, 7, 14, 20, 23,...	[1, 3, 5, 11, 14, 21, 2...	[1, 3, 7, 14, 15, 13, 17]	[1, 3, 5, 6, 8, 21, 30, ...
1<>18	[1, 3, 5, 6, 8, 10, 30, ...	[1, 3, 5, 6, 9, 10, 30, ...	[1, 3, 5, 7, 9, 14, 21, ...	[1, 3, 5, 7, 14, 20, 23,...
1<>19	[1, 3, 5, 6, 9, 10, 30, ...	[1, 3, 5, 6, 8, 21, 30, ...	[1, 4, 5, 11, 12, 15, 19]	[1, 3, 5, 7, 8, 10, 30, ...
1<>20	[1, 3, 5, 7, 9, 14, 15, ...	[1, 2, 6, 9, 21, 20]	[1, 3, 5, 7, 8, 10, 30, ...	[1, 3, 6, 5, 7, 14, 21, ...
1<>21	[1, 3, 7, 5, 11, 14, 9, ...	[1, 3, 5, 7, 9, 21]	[1, 3, 5, 6, 8, 21]	[1, 3, 5, 6, 7, 8, 10, 3...
1<>22	[1, 3, 5, 11, 12, 13, 1...	[1, 3, 5, 11, 12, 13, 1...	[1, 2, 6, 9, 21, 22]	[1, 3, 5, 7, 14, 15, 19,...
1<>23	[1, 4, 5, 11, 14, 21, 2...	[1, 2, 6, 9, 21, 20, 23]	[1, 3, 5, 6, 8, 10, 21, ...	[1, 3, 7, 14, 9, 10, 30,...
1<>24	[1, 3, 5, 7, 14, 20, 22,...	[1, 3, 5, 6, 8, 21, 30, ...	[1, 4, 5, 11, 12, 13, 1...	[1, 3, 5, 7, 8, 10, 30, ...
1<>25	[1, 3, 5, 7, 9, 14, 15, ...	[1, 3, 5, 7, 14, 20, 19,...	[1, 2, 6, 9, 21, 20, 24,...	[1, 3, 5, 7, 14, 20, 22,...
1<>26	[1, 3, 5, 7, 14, 20, 19,...	[1, 2, 6, 9, 21, 22, 28,...	[1, 3, 5, 6, 8, 21, 30, ...	[1, 3, 5, 7, 9, 14, 15, ...
1<>27	[1, 2, 6, 9, 21, 20, 24,...	[1, 3, 7, 14, 12, 15, 1...	[1, 3, 5, 7, 14, 20, 22,...	[1, 3, 5, 11, 14, 20, 2...
1<>28	[1, 2, 6, 9, 21, 22, 28]	[1, 3, 5, 7, 9, 21, 22, ...	[1, 3, 5, 11, 12, 15, 1...	[1, 3, 5, 7, 8, 21, 30, ...
1<>29	[1, 3, 5, 7, 14, 15, 20,...	[1, 3, 5, 6, 8, 21, 20, ...	[1, 3, 5, 11, 12, 13, 1...	[1, 3, 5, 11, 12, 13, 1...
1<>30	[1, 3, 5, 11, 12, 15, 1...	[1, 3, 7, 8, 10, 30]	[1, 3, 5, 6, 8, 10, 30]	[1, 3, 5, 7, 9, 10, 30]
2<>1	[2, 1]	[2, 6, 5, 4, 1]	[2, 3, 1]	
2<>2	[]			
2<>3	[2, 3]	[2, 1, 3]	[2, 6, 3]	
2<>4	[2, 3, 4]	[2, 6, 9, 14, 7, 5, 4]	[2, 1, 4]	[2, 6, 8, 9, 14, 11, 5, 4]
2<>5	[2, 3, 5]	[2, 6, 9, 14, 7, 5]	[2, 6, 8, 9, 14, 11, 5]	[2, 1, 4, 5]
2<>6	[2, 1, 4, 5, 6]	[2, 3, 6]	[2, 6]	
2<>7	[2, 1, 4, 5, 7]	[2, 6, 9, 14, 11, 7]	[2, 1, 3, 7]	[2, 3, 5, 7]
2<>8	[2, 6, 8]	[2, 3, 5, 7, 8]	[2, 3, 6, 9, 8]	[2, 6, 5, 7, 14, 9, 21, 8]
2<>9	[2, 6, 9]	[2, 3, 6, 5, 7, 9]	[2, 3, 7, 9]	[2, 6, 7, 9]
2<>10	[2, 3, 5, 7, 9, 10]	[2, 1, 4, 5, 11, 12, 13,...	[2, 1, 4, 5, 7, 14, 15, ...	[2, 1, 4, 5, 7, 8, 21, 10]

Рис. 6 Таблиця відстаней від кожного вузла до кожного. Початкові значення



From<>To	path	path	path	path
29<>21	[29, 30, 21]	[29, 30, 10, 21]	[29, 26, 24, 20, 21]	[29, 26, 25, 17, 13, 1...
29<>22	[29, 28, 22]	[29, 30, 10, 21, 22]	[29, 22]	[29, 26, 27, 24, 20, 22]
29<>23	[29, 30, 10, 21, 20, 23]	[29, 30, 21, 20, 23]	[29, 30, 10, 9, 14, 20,...	[29, 26, 27, 23]
29<>24	[29, 30, 10, 9, 6, 5, 1...	[29, 22, 20, 19, 24]	[29, 30, 10, 21, 20, 24]	[29, 28, 23, 24]
29<>25	[29, 26, 27, 24, 25]	[29, 30, 21, 20, 14, 1...	[29, 30, 10, 9, 7, 11, ...	[29, 22, 21, 20, 23, 2...
29<>26	[29, 30, 10, 21, 20, 2...	[29, 28, 26]	[29, 22, 21, 9, 14, 12,...	[29, 26]
29<>27	[29, 30, 10, 21, 20, 2...	[29, 30, 10, 21, 20, 2...	[29, 22, 20, 24, 27]	[29, 30, 10, 8, 7, 14, ...
29<>28	[29, 28]	[29, 22, 20, 24, 27, 28]	[29, 30, 10, 21, 22, 28]	[29, 26, 28]
29<>29	[]			
29<>30	[29, 30]	[29, 28, 22, 30]	[29, 22, 21, 10, 30]	[29, 26, 27, 24, 20, 2...
30<>1	[30, 10, 21, 9, 6, 2, 1]	[30, 29, 28, 23, 27, 2...	[30, 22, 20, 14, 7, 5, ...	[30, 22, 23, 20, 14, 1...
30<>2	[30, 29, 26, 27, 24, 1...	[30, 29, 28, 23, 27, 2...	[30, 10, 9, 6, 2]	[30, 22, 20, 14, 7, 5, ...
30<>3	[30, 29, 28, 23, 27, 2...	[30, 22, 20, 14, 7, 5, ...	[30, 10, 8, 6, 3]	[30, 10, 8, 7, 3]
30<>4	[30, 21, 14, 7, 5, 6, 2,...	[30, 29, 26, 27, 24, 1...	[30, 29, 28, 23, 27, 2...	[30, 22, 28, 23, 20, 1...
30<>5	[30, 10, 21, 9, 14, 11,...	[30, 29, 22, 20, 14, 1...	[30, 22, 21, 8, 6, 5]	[30, 22, 23, 20, 14, 7,...
30<>6	[30, 22, 21, 8, 6]	[30, 10, 9, 6]	[30, 10, 8, 6]	[30, 22, 28, 23, 20, 1...
30<>7	[30, 10, 8, 7]	[30, 29, 28, 23, 27, 2...	[30, 10, 21, 9, 14, 7]	[30, 22, 23, 20, 14, 7]
30<>8	[30, 29, 22, 20, 14, 7,...	[30, 21, 20, 14, 9, 6, 8]	[30, 22, 21, 8]	[30, 10, 9, 8]
30<>9	[30, 29, 22, 20, 14, 7,...	[30, 22, 23, 20, 14, 9]	[30, 10, 8, 9]	[30, 29, 28, 23, 27, 2...
30<>10	[30, 21, 10]	[30, 22, 21, 9, 10]	[30, 29, 22, 20, 21, 8,...	[30, 10]
30<>11	[30, 10, 8, 6, 5, 11]	[30, 10, 8, 9, 7, 11]	[30, 22, 20, 14, 7, 5, ...	[30, 29, 28, 23, 27, 2...
30<>12	[30, 22, 23, 20, 19, 1...	[30, 10, 21, 9, 14, 12]	[30, 29, 26, 27, 24, 1...	[30, 29, 28, 27, 24, 1...
30<>13	[30, 21, 8, 9, 7, 14, 2...	[30, 22, 28, 27, 26, 2...	[30, 29, 26, 25, 24, 1...	[30, 10, 8, 6, 7, 11, 1...
30<>14	[30, 10, 8, 7, 14]	[30, 29, 26, 24, 20, 1...	[30, 10, 8, 6, 5, 7, 14]	[30, 21, 8, 9, 7, 14]
30<>15	[30, 29, 28, 27, 24, 1...	[30, 22, 28, 27, 26, 2...	[30, 10, 21, 9, 14, 12,...	[30, 22, 23, 27, 26, 2...
30<>16	[30, 10, 8, 7, 14, 20, ...	[30, 29, 28, 27, 26, 2...	[30, 29, 22, 21, 14, 1...	[30, 21, 8, 9, 7, 14, 1...
30<>17	[30, 10, 9, 14, 15, 16,...	[30, 10, 8, 6, 7, 11, 1...	[30, 21, 9, 14, 12, 13,...	[30, 29, 28, 26, 25, 17]
30<>18	[30, 21, 9, 14, 12, 13,...	[30, 21, 8, 9, 7, 14, 2...	[30, 10, 9, 14, 15, 19,...	[30, 22, 20, 19, 18]
30<>19	[30, 10, 8, 7, 14, 15, ...	[30, 29, 28, 27, 24, 1...	[30, 10, 9, 14, 20, 19]	[30, 10, 21, 20, 19]
30<>20	[30, 10, 8, 9, 14, 11, ...	[30, 10, 8, 6, 5, 7, 14,...	[30, 10, 9, 14, 12, 15,...	[30, 21, 8, 9, 7, 14, 20]
30<>21	[30, 10, 21]	[30, 21]	[30, 22, 21]	[30, 29, 22, 20, 21]
30<>22	[30, 21, 20, 22]	[30, 10, 21, 22]	[30, 29, 22]	[30, 22]
30<>23	[30, 10, 8, 7, 11, 12, ...	[30, 10, 9, 14, 12, 13,...	[30, 10, 8, 7, 11, 12, ...	[30, 10, 8, 6, 5, 7, 14,...
30<>24	[30, 10, 8, 7, 14, 15, ...	[30, 21, 9, 14, 12, 13,...	[30, 10, 8, 9, 14, 15, ...	[30, 21, 8, 9, 7, 14, 2...
30<>25	[30, 21, 8, 9, 7, 14, 2...	[30, 10, 8, 7, 14, 15, ...	[30, 21, 9, 14, 12, 13,...	[30, 10, 8, 7, 11, 12, ...
30<>26	[30, 10, 8, 7, 11, 12, ...	[30, 10, 21, 20, 24, 2...	[30, 21, 8, 9, 7, 14, 2...	[30, 10, 9, 14, 20, 23,...
30<>27	[30, 10, 8, 9, 14, 11, ...	[30, 10, 8, 7, 11, 12, ...	[30, 10, 8, 7, 14, 15, ...	[30, 10, 8, 6, 2, 1, 4, ...
30<>28	[30, 21, 14, 20, 19, 2...	[30, 10, 8, 9, 14, 15, ...	[30, 21, 9, 14, 20, 24,...	[30, 21, 20, 23, 28]
30<>29	[30, 29]	[30, 22, 28, 29]	[30, 21, 20, 24, 27, 2...	[30, 10, 21, 22, 29]
30<>30	[]			

Рис. 4 Таблиця відстаней від кожного вузла до кожного. Початкові значення

5. Моделювання процесу передачі повідомлень в мережі передачі даних заданої структури.

Для моделювання та аналізу передачі повідомлень в мережі передачі даних обраної топології було проведено серію тестів:

1) Тест №1 (табл. 2). Тест призначений для аналізу часу передачі даних при віддаленості вузла-відправника повідомлення (вузол №1) від супервізора (вузол №3) 5 мс.
2) Тест №2 (таблиця 3). Тест призначений для аналізу часу передачі даних при віддаленості вузла-відправника повідомлення (вузол №1) від супервізора (вузол №24) 47 мс.
3) Тест №3 (таблиця 4). Тест призначений для аналізу часу передачі даних при віддаленості вузла-відправника повідомлення (вузол №1) від супервізора (вузол №10) 33 мс.
4) Тест №4 (таблиця 5). Тест призначений для аналізу часу передачі даних при різній віддаленості вузлів один від одного. Так, відстань між вузлами №1 і №2 складає 2 мс; між вузлами №19 і №22 – 23 мс; між вузлами №3 і №18 – 64 мс.
5) Тест №5 (таблиця 6). Тест призначений для аналізу передачі даних при різному значенні розміру інформаційних пакетів.
6) Тест №6 (таблиця 7). Тест призначений для аналізу передачі повідомлень різного розміру.
7) Тест №7 (таблиця 8). Тест повторює тест №5 в умовах несправності каналу між вузлами №1 та №4

Таблиця 2

Тест №1. Результати передачі даних при віддаленості вузла відправника (вузол №1) від супервізора (вузол №3) 5 мс

Розмір повідомлення, Кбайт	Розмір інформа-ційних пакетів, байт	Кількість інформа-ційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Парамет-ри пошуку	Час, мс
100	20	5000	5006	1	11	Логічний зв’язок.	Вага	129643
100	20	5000	5006	1	11	Логічний зв’язок	Кіль-кість вузлів	129646
100	20	5000	5029	1	11	Дейта-грамний	-	1314
100	200	500	506	1	11	Логічний зв’язок	Вага	16618
100	200	500	506	1	11	Логічний зв’язок	Кіль-кість вузлів	16620
100	200	500	529	1	11	Дейта-грамний	-	8514
1000	500	2000	2006	1	11	Логічний зв’язок	Вага	90322
1000	500	2000	2006	1	11	Логічний зв’язок	Кіль-кість вузлів	90324
1000	500	2000	2029	1	11	Дейта-грамний	-	20514

Таблиця 3

Тест №2. Результати передачі даних при віддаленості вузла відправника (вузол №1) від супервізора (вузол №24) 47 мс

Розмір повідомлення, Кбайт	Розмір інформа-ційних пакетів, байт	Кількість інформа-ційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Парамет-ри пошуку	Час, мс
100	20	5000	5006	1	11	Логічний зв’язок.	Вага	162096
100	20	5000	5006	1	11	Логічний зв’язок	Кіль-кість вузлів	170211
100	20	5000	5043	1	11	Дейта-грамний	-	6385
100	200	500	506	1	11	Логічний зв’язок	Вага	19910
100	200	500	506	1	11	Логічний зв’язок	Кіль-кість вузлів	20739
100	200	500	543	1	11	Дейта-грамний	-	13584
1000	500	2000	2006	1	11	Логічний зв’язок	Вага	103334
1000	500	2000	2006	1	11	Логічний зв’язок	Кіль-кість вузлів	106589
1000	500	2000	2043	1	11	Дейта-грамний	-	25584

Таблиця 4

Тест №3. Результати передачі даних при віддаленості вузла відправника (вузол №1) від супервізора (вузол №10) 33 мс

Розмір повідомлення, Кбайт	Розмір інформа-ційних пакетів, байт	Кількість інформа-ційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Парамет-ри пошуку	Час, мс
100	20	5000	5006	1	11	Логічний зв’язок.	Вага	153977
100	20	5000	5006	1	11	Логічний зв’язок	Кіль-кість вузлів	162090
100	20	5000	5043	1	11	Дейта-грамний	-	4998
100	200	500	506	1	11	Логічний зв’язок	Вага	19081
100	200	500	506	1	11	Логічний зв’язок	Кіль-кість вузлів	19904
100	200	500	543	1	11	Дейта-грамний	-	12198
1000	500	2000	2006	1	11	Логічний зв’язок	Вага	100075
1000	500	2000	2006	1	11	Логічний зв’язок	Кіль-кість вузлів	103328
1000	500	2000	2043	1	11	Дейта-грамний	-	24198

Таблиця 5

Тест №4. Результати аналізу часу передачі даних при різній віддаленості вузлів один від одного.

Розмір повідомлення, Кбайт	Розмір інформа-ційних пакетів, байт	Кількість інформа-ційних пакетів	Кількість службових пакетів	Від вузла - До вузла	Відстань між вузлами, мс	Режим	Параме-три пошуку	Час, мс
1000	200	5000	5006	1-2	2	Логічний зв'язок.	Вага	149431
1000	200	5000	5006	1-2	2	Логічний зв'язок	Кіль-кість вузлів	157546
1000	200	5000	5015	1-2	2	Дейта-грамний	-	10240
1000	200	5000	5006	19-22	23	Логічний зв'язок	Вага	161593
1000	200	5000	5006	19-22	23	Логічний зв'язок	Кіль-кість вузлів	161593
1000	200	5000	5141	19-22	23	Дейта-грамний	-	15781
1000	200	5000	5006	3-18	64	Логічний зв'язок	Вага	222482
1000	200	5000	5006	3-18	64	Логічний зв'язок	Кіль-кість вузлів	250892
1000	200	5000	5185	3-18	64	Дейта-грамний	-	21175

Таблиця 6

Тест №5. Результати аналізу передачі даних при різному значенні розміру інформаційних пакетів

Розмір повідомлення, Кбайт	Розмір інформаційних пакетів, байт	Кількість інформаційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Параметри пошуку	Час, мс
1000	50	20000	20006	1	5	Логічний зв’язок.	Вага	558652
1000	50	20000	20006	1	5	Логічний зв’язок	Кількість вузлів	591065
1000	50	20000	20031	1	5	Дейтаграмний	-	5400
1000	250	4000	4006	1	5	Логічний зв’язок	Вага	143780
1000	250	4000	4006	1	5	Логічний зв’язок	Кількість вузлів	150273
1000	250	4000	4031	1	5	Дейтаграмний	-	10200
1000	576	1737	1743	1	5	Логічний зв’язок	Вага	85133
1000	576	1737	1743	1	5	Логічний зв’язок	Кількість вузлів	87961
1000	576	1737	1766	1	5	Дейтаграмний	-	18024

Таблиця 7

Тест №6. Результати аналізу передачі повідомлень різного розміру.

Розмір повідомлення, Кбайт	Розмір інформаційних пакетів, байт	Кількість інформаційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Параметри пошуку	Час, мс
10	100	100	106	1	5	Логічний зв'язок.	Вага	3047
10	100	100	106	1	5	Логічний зв'язок	Кількість вузлів	3222
10	100	100	131	1	5	Дейтаграмний	-	6600
100	100	1000	1006	1	5	Логічний зв'язок	Вага	29984
100	100	1000	1006	1	5	Логічний зв'язок	Кількість вузлів	31617
100	100	1000	1031	1	5	Дейтаграмний	-	6432
1000	100	10000	10006	1	5	Логічний зв'язок	Вага	299354
1000	100	10000	10006	1	5	Логічний зв'язок	Кількість вузлів	315567
1000	100	10000	10031	1	5	Дейтаграмний	-	6100

Таблиця 8

Тест №7. Результати аналізу передачі даних при несправності каналу між вузлами №1 та №4

Розмір повідомлення, Кбайт	Розмір інформаційних пакетів, байт	Кількість інформаційних пакетів	Кількість службових пакетів	Від вузла	До вузла	Режим	Параметри пошуку	Час, мс
100	20	5000	5006	1	11	Логічний зв'язок.	Вага	129646
100	20	5000	5006	1	11	Логічний зв'язок	Кількість вузлів	149927
100	20	5000	5045	1	11	Дейтаграмний	-	1314
100	200	500	506	1	11	Логічний зв'язок	Вага	16620
100	200	500	506	1	11	Логічний зв'язок	Кількість вузлів	18683
100	200	500	545	1	11	Дейтаграмний	-	8514
1000	500	2000	2006	1	11	Логічний зв'язок	Вага	90324
1000	500	2000	2006	1	11	Логічний зв'язок	Кількість вузлів	98474
1000	500	2000	2045	1	11	Дейтаграмний	-	20514

6. Аналіз та порівняння отриманих результатів

На підставі проведених тестів можна зробити аналіз за наступними параметрами:

1) Віддаленість вузлів від супервізора. (рис.8, рис.9, рис.10)
2) Віддаленість вузлів один від одного (рис.11)
3) Розмір інформаційних пакетів (рис. 12)
4) Співвідношення інформаційних і службових пакетів (рис. 13)
5) Розмір повідомлення (рис. 14)
6) Несправність каналу 1<>4 (рис. 15)
7) Інтенсивність запитів від кінцевих вузлів, при якій час очікування його обслуговування більше заданого часу T (рис. 16, рис. 17)

1) Проаналізуємо час передачі даних при різній віддаленості вузла-відправника повідомлення від вузла-супервізора (рис.8, рис.9, рис.10). Діаграми на рис.8, рис.9 та рис.10 побудовані на основі тестів №1,2,3 відповідно (таблиці 2, 3, 4), відображають залежність віддаленості супервізора від вузла-відправника. Так, в першому випадку супервізор – вузол №3, віддаленість від вузла-відправника №1 – 5 мс. В другому випадку супервізор – вузол №10, віддаленість – 33 мс; в третьому – вузол №24, віддаленість – 47 мс.

Незалежно від розміру повідомлення та кількості пакетів час передачі даних збільшується зі збільшенням відстані до супервізора. Особливо це помітно при передачі даних із попереднім встановленням логічного з'єднання. Адже в цьому випадку звертання до супервізора відбувається частіше, ніж в дейтаграмному режимі, за рахунок відправки додаткових службових пакетів. Дейтаграмний режим працює без підтвердження отримання даних з супервізора, тому вузол-відправник звертається до супервізора лише один раз з метою дізнатися необхідну кількість шляхів до вузла-адресата.

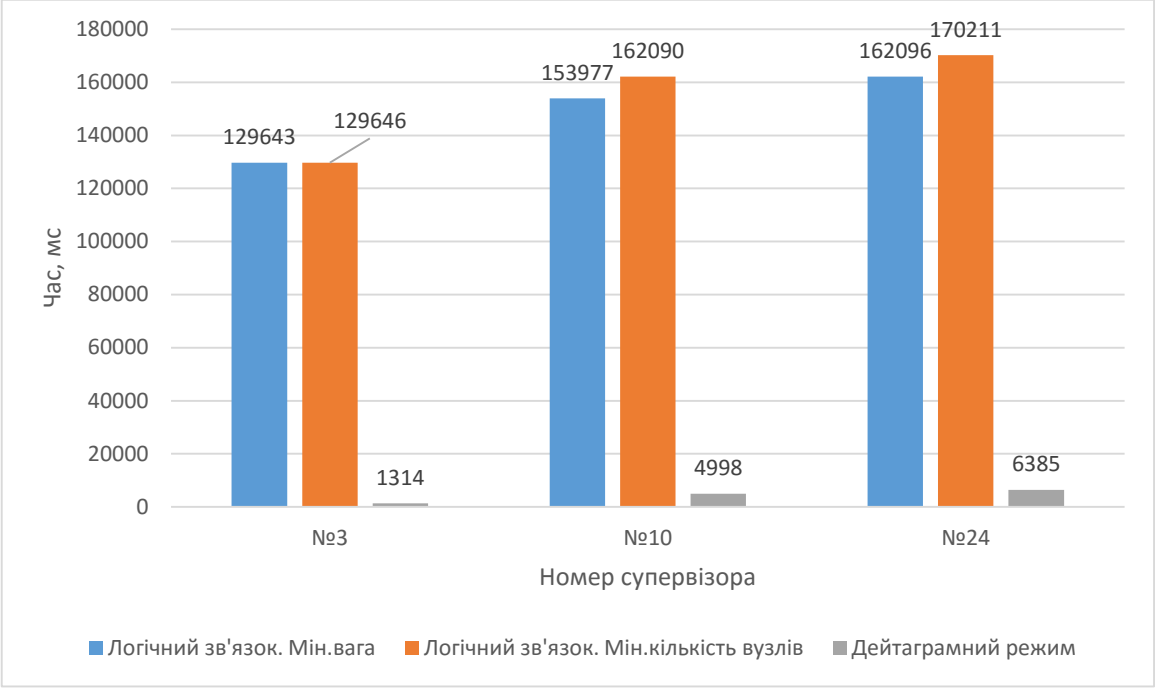


Рис. 8. Аналіз часу передачі даних за віддаленістю від супервізора. Розмір повідомлення 100Кбайт, розмір інформаційних пакетів - 20байт

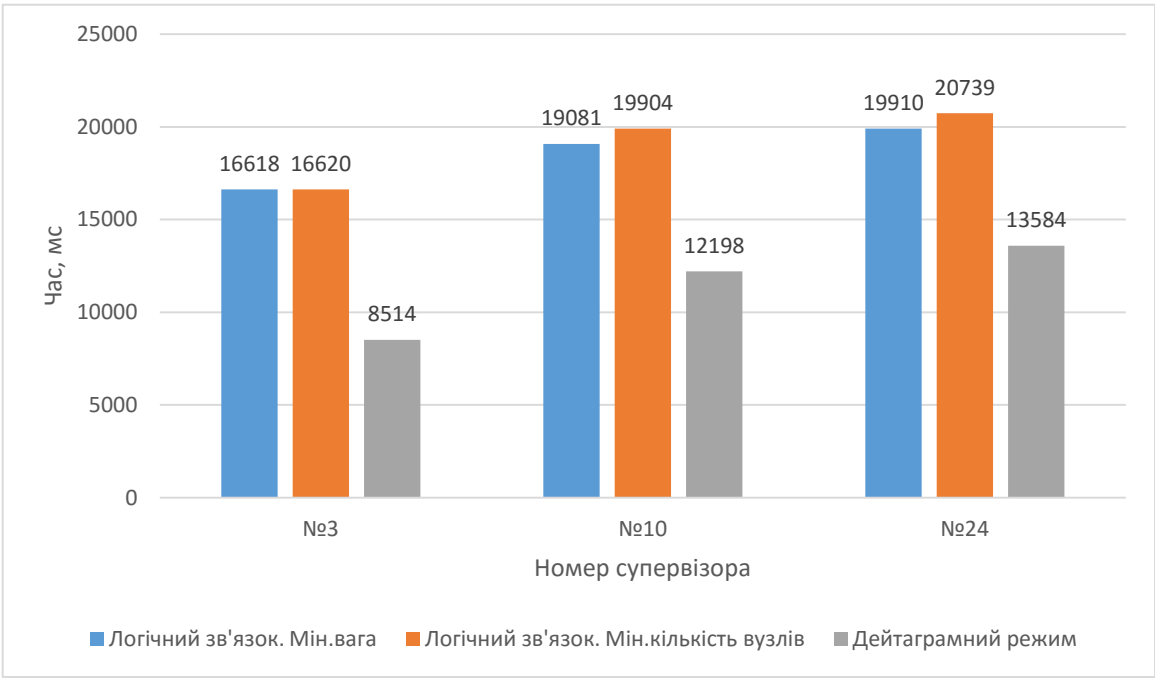


Рис. 9. Аналіз часу передачі даних за віддаленістю від супервізора. Розмір повідомлення 100Кбайт, розмір інформаційних пакетів - 200байт

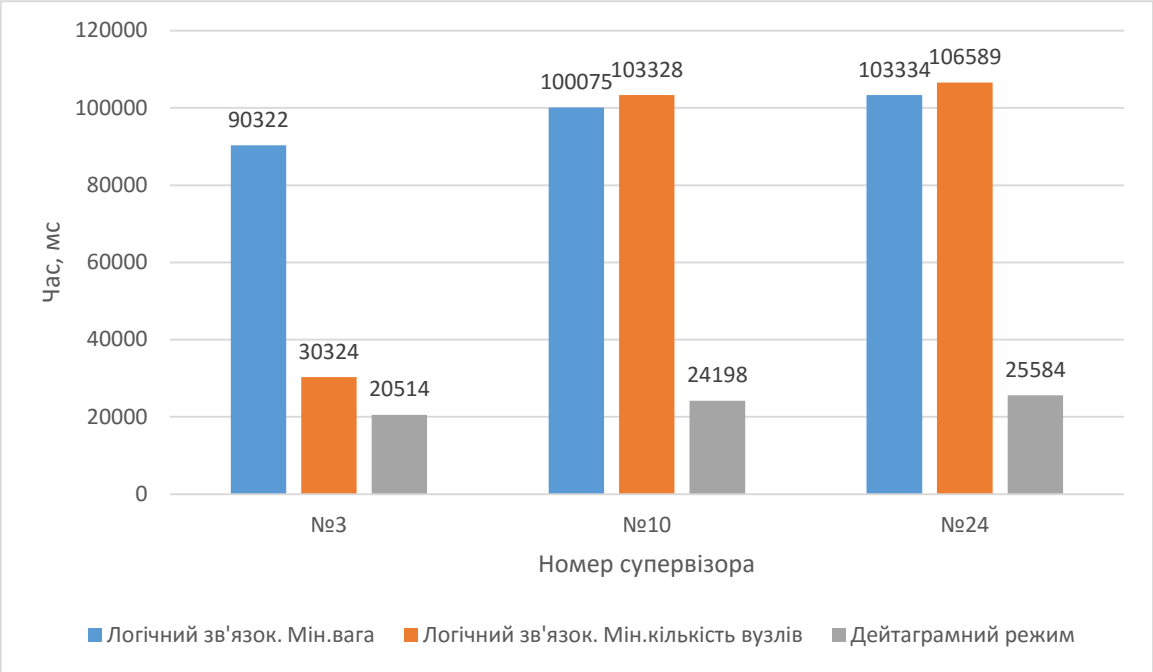


Рис. 10. Аналіз часу передачі даних за віддаленістю від супервізора. Розмір повідомлення 1000Кбайт, розмір інформаційних

2) Проаналізуємо час передачі даних в залежності від віддаленості вузлів, що приймають учать у передачі, один від одного (рис.11). Діаграма на рис. 11 побудована на основі тесту №4 (таблиця 5) і відображає залежність часу передачі даних від відстані вузла-відправника до вузла–адресата, незалежно від розміру повідомлення та кількості пакетів.

Зі збільшенням відстані між вузлами збільшується і час передачі даних, як у дейтаграмному режимі так і в режимі встановлення логічного з’єднання. Збільшення ваги ліній означає збільшення часу передачі даних у каналі, відповідно збільшується і загальний час передачі. Затримка в буфері вузла є незначною, відповідно пакет, хоч у проходить менше транзитних ділянок, проте довше затримується у каналах з великою вагою. Таким чином пошук мінімальному шляху за кількістю транзитних ділянок не є виправданим. Дейтаграмний режим дає значний виграш в часі, адже

пакети відправляються по різних каналах паралельно, а службові пакети мають менший розмір, ніж в режимі із встановленням логічного з'єднання.

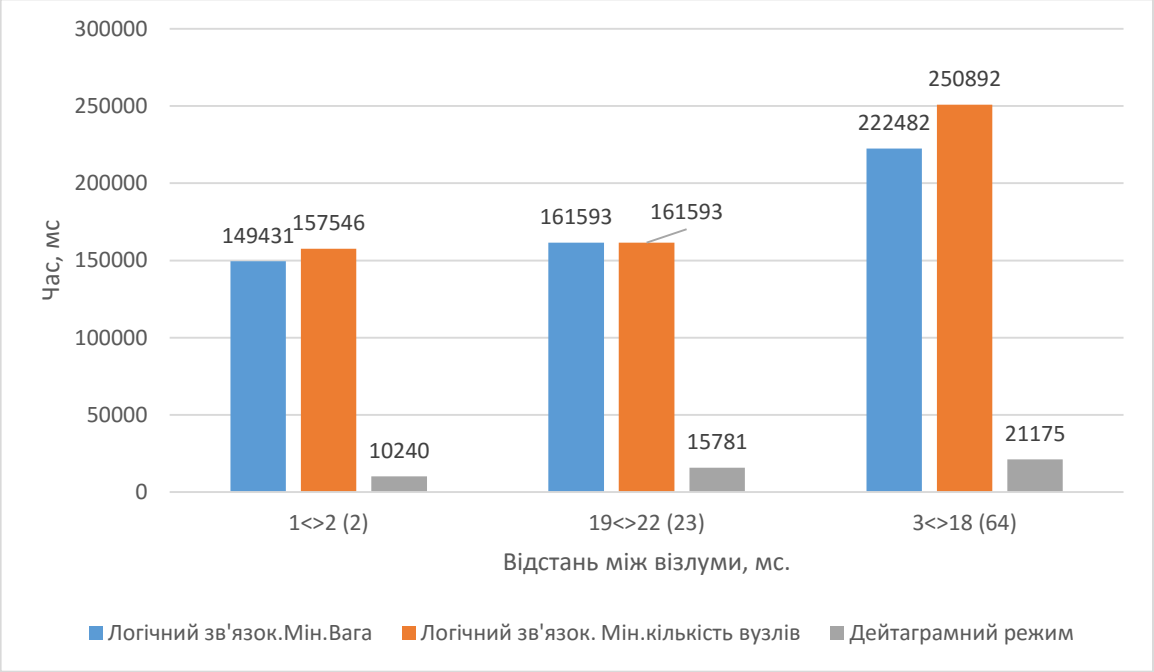


Рис. 11. Аналіз часу передачі даних за віддаленістю вузлів один від одного. Розмір повідомлення 1000Кбайт, розмір інформаційних пакетів - 200байт

3) Проаналізуємо час передачі даних в залежності від розміру інформаційних пакетів (рис. 12). Графік, забражений на рис. 12, побудований на основі тесту №5 (таблиця 6) і відображає час передачі даних в залежності від розміру інформаційних пакетів. Дані передаються від вузла 1 до вузла 5, відстань між якими 12 мс.

Зі збільшенням розміру інформаційних пакетів час передачі даних в режимі із встановленням логічного з'єднання суттєво зменшується, наближуючись до мінімума за даних параметрах передачі при досягненні MTU (576 байт). Це пояснюється зменшенням кількості службових пакетів. У дейтаграмному режимі навпаки: збільшення розміру інформаційних пакетів веде до збільшення часу їх передачі. Розмір службових пакетів у

дейтаграмному режимі менший, ніж у режимі із встановленням логічного з'єднання, тому вони незначно впливають на час передачі повідомлення.

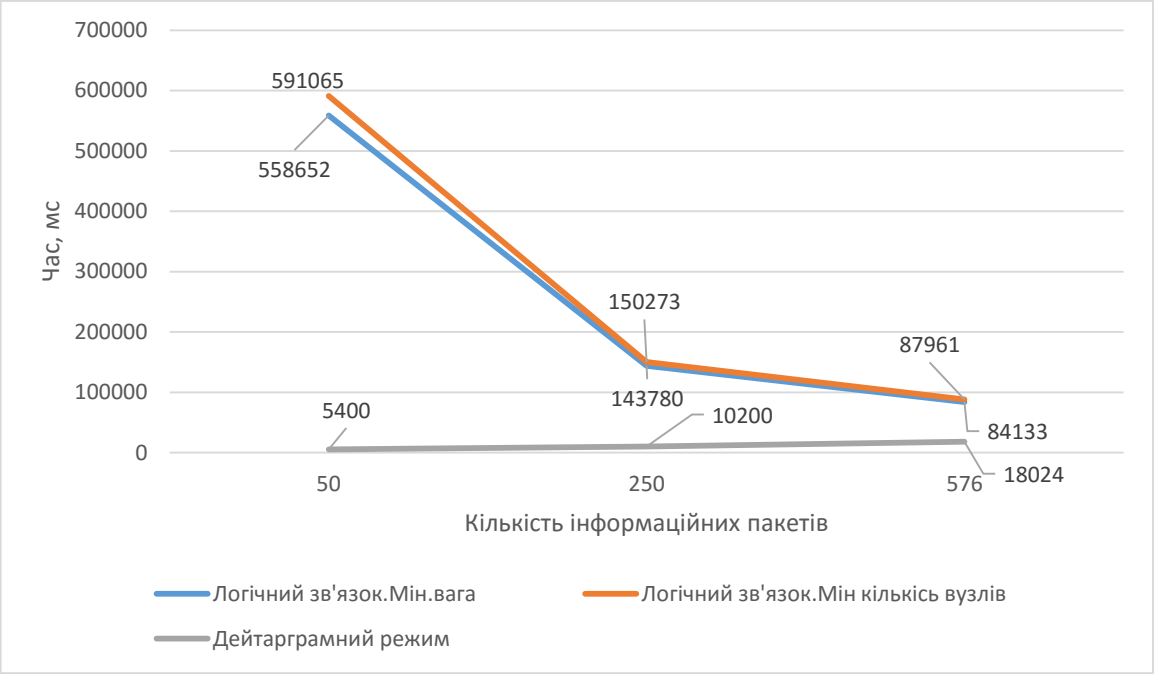


Рис. 12. Аналіз часу передачі даних за розміром інформаційних пакетів. Розмір повідомлення 1000Кбайт

4) На рис.13 відображений аналіз співвідношення інформаційних і службових пакетів у дейтаграмному режимі.

Діаграма на рис. 13 побудована на основі тесту №4 (таблиця 5) і відображає кількість службових та інформаційних пакетів в залежності ваги маршруту між кінцевими вузлами. Як бачимо, зі збільшенням відстані між вузлами збільшується і кількість службових пакетів. Це пояснюється тим, що у дейтаграмному режимі кожен наступний вузол маршруту передачі пакету відправляє попередньому службовий пакет-підтвердження прийому. Кількість інформаційних пакетів є незмінною, адже вона залежить не від ваги маршруту, а від розміру повідомлення та інформаційних пакетів.

В режимі із встановленням логічного з'єднання кількість службових пакетів завжди більша за кількість інформаційних на 6 одиниць: 2 пакети

на отримання маршруту від супервізору, 2 пакети на встановлення та підтвердження з'єднання, 2 пакети на роз'єднання.

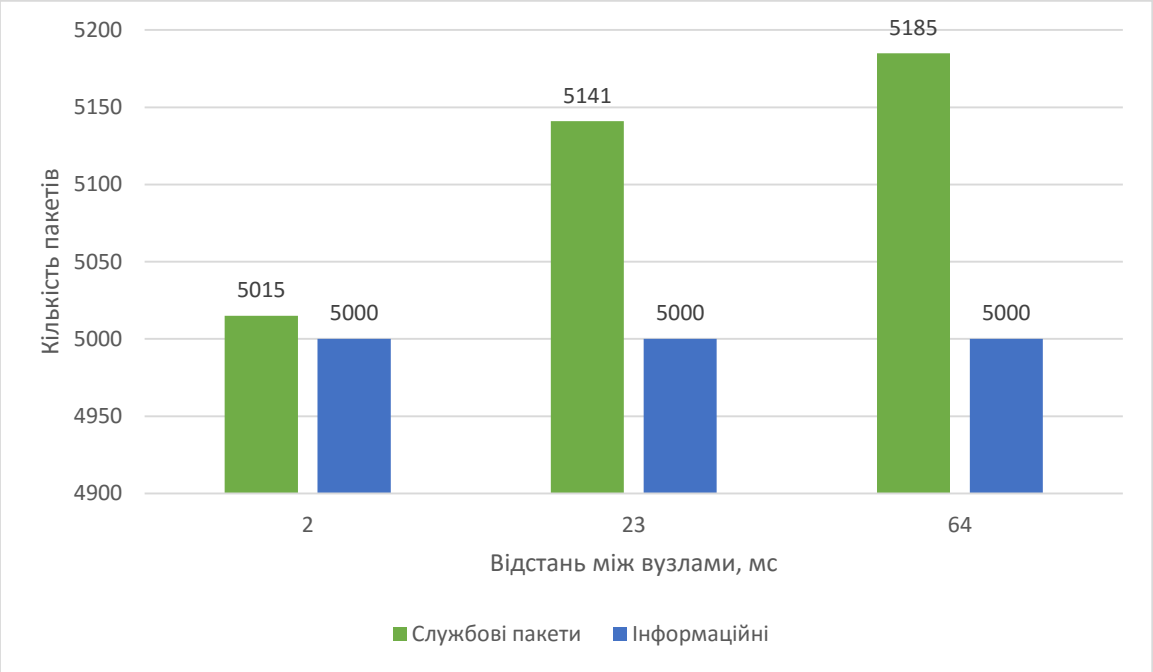


Рис. 13. Аналіз співвідношення службових та інформаційних пакетів в дейтагрмному режимі

5) Проаналізуємо час передачі даних в залежності від розміру повідомлення (рис. 14). Графік на рис. 14 побудований на основі тесту №6 (таблиця 7) і відображає час передачі даних в залежності від розміру повідомлення. Як бачимо, чим більший розмір повідомлення, тим повільніше передаються дані при постійному розмірі інформаційних пакетів у режимі із встановленням логічного зв'язку. Більший розмір повідомлення означає і більшу кількість пакетів. Відповідно затримка в транзитних вузлах збільшується, що і є основним чинником повільної передачі повідомлення. Враховуючи те, що в дейтаграмному режимі пакети передаються паралельно по різним каналам, затримки на транзитних ділянках є незначними.

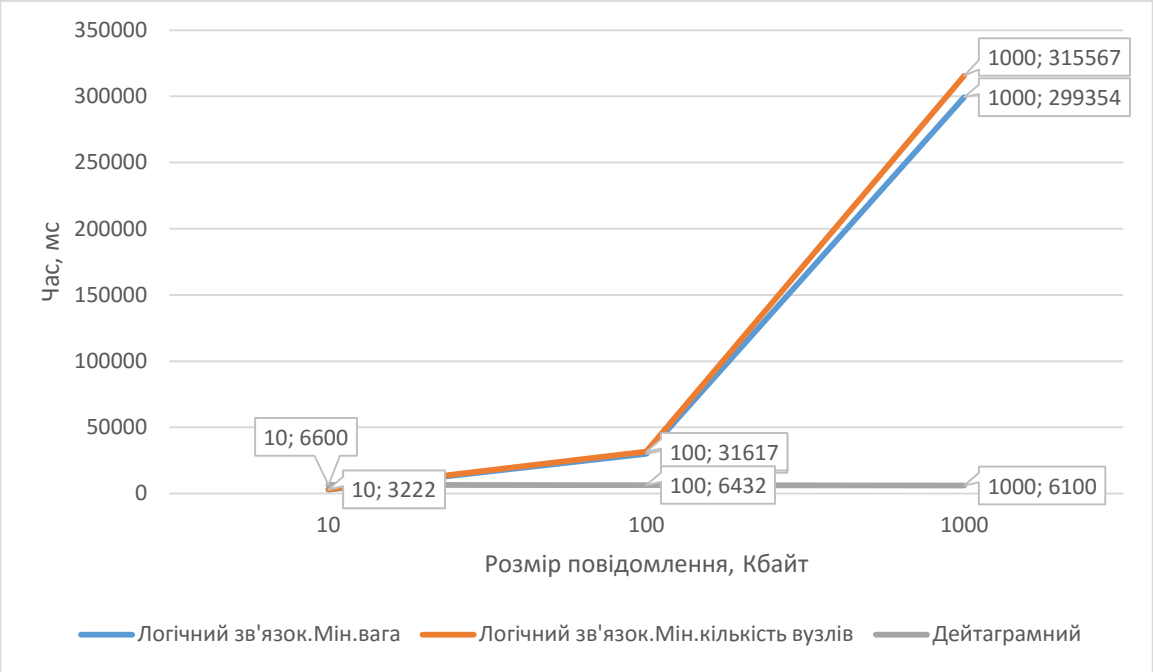


Рис. 14. Аналіз часу передачі даних за розміром повідомлення. Розмір інформаційних пакетів 100 байт

6) Проаналізуємо час доставки повідомлення при несправності каналу від вузла №1 до вузла №4 (рис. 15). Діаграма на рис. 15 побудована на основі тестів №1 (таблиця 2) та №7 (таблиця 8) і відображає зміну в часі передачі даних при несправності каналу 1<>4. Несправність каналу веде до зміни таблиці маршрутизації супервізора. Перераховуються мінімальні маршрути, їх довжина. Відповідно час передачі дещо збільшився в тому випадку, якщо канал 1<>4 входив до маршруту до виходу з ладу. Якщо канал 1<>4 до маршруту не входив, час передачі повідомлення не змінився

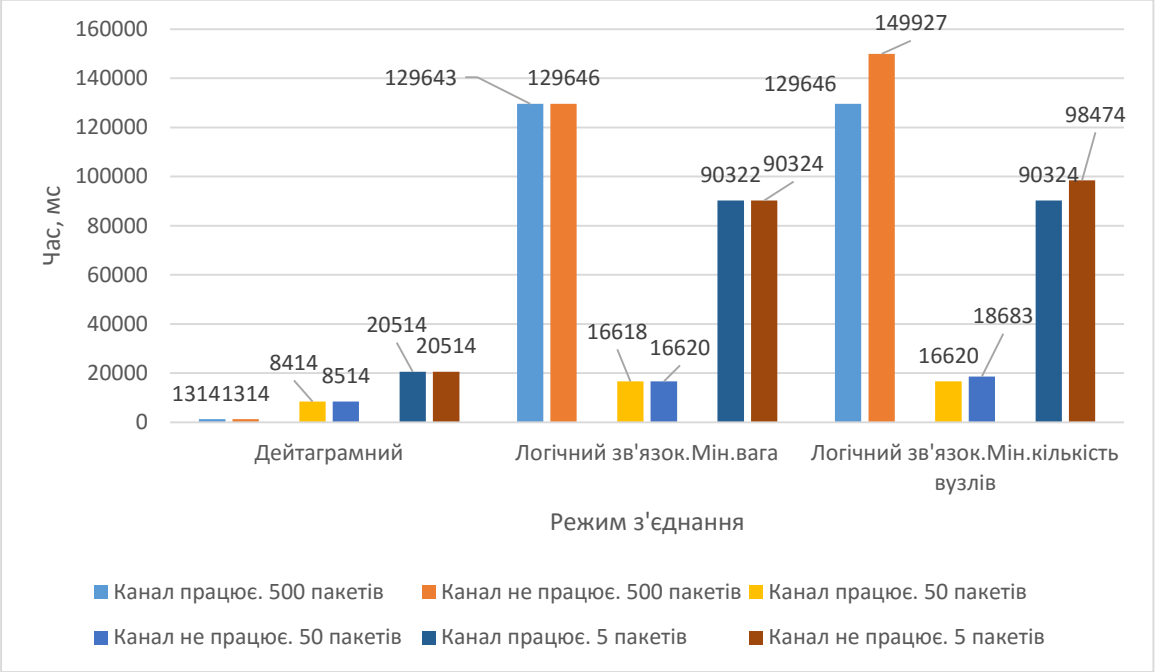


Рис. 15. Аналіз часу передачі даних при несправному каналі між вузлами №1 та №4.

7) Розрахуємо інтенсивність запитів від кінцевих вузлів, при якій час очікування його обслуговування більше заданого часу T на основі даних тесту №5 (таблиця 6).

a. Дейтаграмний режим.

Повідомлення розміром 1000 Кбайт поділяється на інформаційні пакети, розміром 576 байт і передається за 18024 мс. Пакети надсилалися за п’ятьма маршрутами, час передачі окремих пакетів по цих маршрутах відповідно дорівнює 9221.0 мс , 9230.0 мс, 13830.0 мс, 13844.0 мс та 13832.0 мс. Тобто коли останній пакет на першому маршруті дійшов до вузла-адресата, відправляється друге повідомлення. Час очікування другого повідомлення = 18024 - 9221 = 8803 (мс). Таким чином, якщо задане T = 9000 мс, то інтенсивність запитів від кінцевих вузлів має бути 2,05 повідомлення в секунду. При T = 72000 мс, інтенсивність – 8,18 повідомлень в секунду.

Графік залежності інтенсивності повідомлені від заданого часу очікування зображений на рис. 16

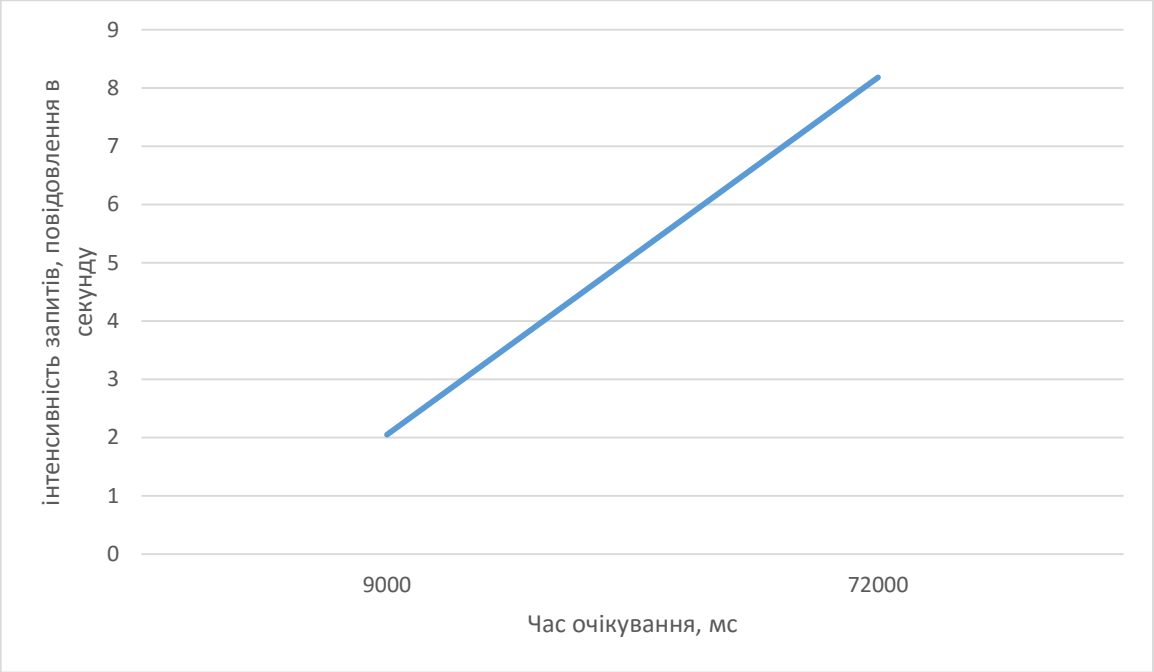


Рис. 16. Аналіз інтенсивності запитів від кінцевих вузлів, при якій час очікування його обслуговування більше заданого часу T в дейтаграмному режимі

b. Режим із встановленням логічного з'єднання

Повідомлення розміром 1000Кбайт поділяється на пакети, розміром 576 байт і передається за 85133 мс. Наступне повідомлення буде відправлено лише по закінченню цього часу. Тому, якщо час очікування T = 90 000 мс, то кількість повідомлень в секунду = 2,11мс, якщо T = 360с, то інтенсивність = 8,45 повідомлень в секунду.

Графік залежності інтенсивності повідомлені від заданого часу очікування зображений на рис. 17.

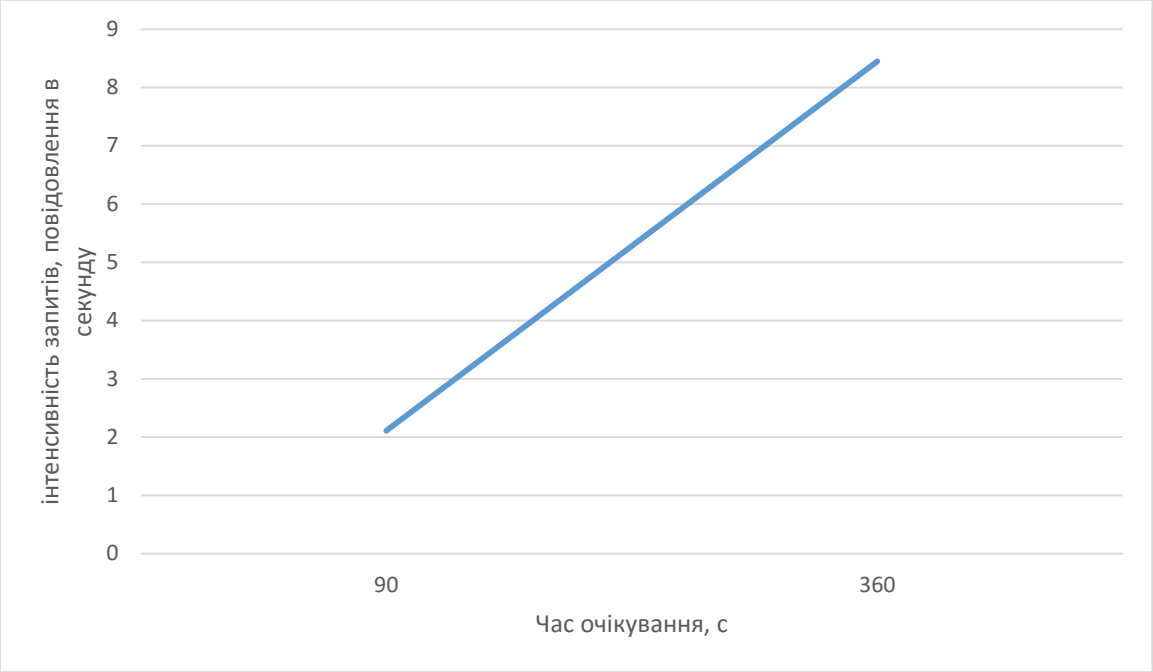


Рис. 17. Аналіз інтенсивності запитів від кінцевих вузлів, при якій час очікування його обслуговування більше заданого часу T в режимі із встановленням логічного з’єднання

ВИСНОВКИ

Моделювання мережі передачі даних дозволяє проаналізувати параметри мережі, що лише проектується, та проаналізувати передачу даних в різних режимах та вибрати той, що дає більшу ефективність.

Наведені результати моделювання системи передачі даних певної конфігурації, а саме: залежність часу передачі даних від розміру повідомлення, розміру інформаційних пакетів, режиму передачі даних, типу пошуку мінімального шляху.

Наведені дослідження показали, що співвідношення інформаційних і службових пакетів в мережі передачі даних майже завжди становить 1:1, для режиму із встановленням логічного з'єднання кількість службових пакетів залежить від кількісті інформаційних, у дейтаграмному режимі – ще й від кількості транзитних ділянок маршрутів передачі повідомлення. Але порівнювати інформаційні та службові пакети некоректно, адже їх розмір відрізняється на порядок.

З проведених досліджень випливає, шо майже завжди дейтаграмний режим є швидшим, ніж режим із встановленням логічного з'єднання, на 50%. У випадку передачі великої кількості пакетів невеликого розміру дейтаграмний режим працює швидше на порядок. Лише коли великі повідомлення розбиваються на невелику кількість інформаційних пакетів, використання дейтаграмного режиму для зменшення часу передачі є недоцільним. При несправності одного з каналів втрати часу є незначними.

Слабке місце алгоритму Tymnet – його вразливість до виходу з ладу супервізора. Для підвищення надійності в мережі Tymnet використовуються вузли, які дублюють роботу супервізора.

Вибір режиму передачі також залежить від призначення передачі даних. У випадку, коли важливо надійно переслати дані, необхідно попередньо встановлювати з'єднання та відправляти повідомлення-підтвердження з'єднання і отримання даних. А у випадках, коли швидкість передачі важливіша за якість, більш ефективним буде дейтаграмний режим.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Комп’ютерні мережі: навчальний посібник / Азаров О.Д., Захарченко С.М., Кадук О.В., Орлова М.М., Тарасенко В.П.. – Вінниця: ВНТУ. – 2013. – 371 с.
2. Конспект лекцій з дисципліни «Комп’ютерні мережі» / Орлова М.М.
3. Алгоритм Дейкстры. Поиск оптимальных маршрутов на графе [Електронний ресурс]. - Режим доступу - http://habrahabr.ru/post/111361/
4. Моя Библиотека - ТОМ 2: Маршрутизации сети Tymnet [Електронний ресурс]. - Режим доступу - http://mybiblioteka.su/tom2/3-169658.html
5. Terminal-Oriented Computer-Communication Networks / M. Schwartz and R. R. Boorstyn are with the Polytechnic Institute of Brooklyn, Brooklyn, N. Y. 11201. R. L. Pickholtz is with George Washington University, Washington, D.C. [Електронний ресурс]. - Режим доступу - http://rogerdmoore.ca/PS/TONET/TON.html#TYMNET
6. Системи та мережі передавання даних (Частина II) / Навчальний посібник. – Вінниця: ВНТУ, 2007. – 101 с. [уклад. О. М. Бевз, С. Г. Кривогубченко, А. Я. Кулик]. [Електронний ресурс]. – Режим доступу - http://obevz.vk.vntu.edu.ua/file/fa839b8597647d614dbdb6f3d476b077.pdf

Додаток 2

Реалізація алгоритму Дейкстри

```
import graph.Edge;
import graph.Graph;
import graph.Vertex;

import java.util.*;

/**
 * Created by Eugenia on 08.12.2015.
 */
public class DijkstraAlgorithm {
    private final List<Vertex> nodes;
    private List<Edge> edges;
    private Set<Vertex> settledNodes;
    private Set<Vertex> unSettledNodes;
    private Map<Vertex, Vertex> predecessors;
    private Map<Vertex, Integer> distance;
    final int INF = Integer.MAX_VALUE / 2;
    private final List<Integer> backUPedges = new ArrayList<>();


    public DijkstraAlgorithm(Graph graph) {
        // create a copy of the array so that we can operate on this array
        this.nodes = new ArrayList<Vertex>(graph.getVertexes());
        this.edges = new ArrayList<Edge>(graph.getEdges());
        setBackUPedges();
    }

    private void setBackUPedges() {
```

```
        for (Edge edge : edges) {
            backUPedges.add(edge.getWeight());
        }
    }

    public void execute(Vertex source, boolean protocol) {
        settledNodes = new HashSet<Vertex>();
        unSettledNodes = new HashSet<Vertex>();
        distance = new HashMap<Vertex, Integer>();
        predecessors = new HashMap<Vertex, Vertex>();
        distance.put(source, 0);
        for (Vertex v : nodes)
            unSettledNodes.add(v);

        while (unSettledNodes.size() > 0) {

            if (protocol) {
                Vertex node = getMinimum(unSettledNodes);
                settledNodes.add(node);
                unSettledNodes.remove(node);
                findMinimalDistances(node);
            } else {
                Vertex node = getMinimum(unSettledNodes);
                settledNodes.add(node);
                unSettledNodes.remove(node);
                findMinimalDistances(node);
            }
        }
    }
```

```
    public ArrayList<LinkedList<Vertex>> executeDatagram(Vertex source,
Vertex destination, boolean protocol) {
        ArrayList<LinkedList<Vertex>> tmp = new ArrayList<>();
        Set<LinkedList<Vertex>> paths = new HashSet<>();
        while (true) {

            int length = paths.size();
            execute(source, false);
            LinkedList<Vertex> minPath = getPath(destination);
            paths.add(minPath);
            if (length == paths.size())
                break;

            setINFEdges(minPath);

        }
        for (int i = 0; i < edges.size(); i++) {
            edges.get(i).setWeight(backUPedges.get(i));
        }
        tmp.addAll(paths);
        return tmp;
    }

    void setINFEdges(LinkedList<Vertex> path) {
        for (int i = 0; i < edges.size(); i++) {
            for (int j = 0; j < path.size() - 1; j++) {
                if ((edges.get(i).getSource() == path.get(j) &&
edges.get(i).getDestination() == path.get(j + 1)) || (edges.get(i).getDestination() ==
path.get(j) && edges.get(i).getSource() == path.get(j + 1))) {
```

```
                edges.get(i).setWeight(INF);
            }
        }
    }
}


private void findMinimalDistances(Vertex node) {
    List<Vertex> adjacentNodes = getNeighbors(node);
    for (Vertex target : adjacentNodes) {
        if (getShortestDistance(target) > getShortestDistance(node)
                + getDistance(node, target)) {
            distance.put(target, getShortestDistance(node)
                    + getDistance(node, target));
            predecessors.put(target, node);
            unSettledNodes.add(target);
        }
    }

}


private int getDistance(Vertex node, Vertex target) {
    for (Edge edge : edges) {
        if ((edge.getSource().equals(node)
                && edge.getDestination().equals(target))
                || (edge.getDestination().equals(node)
                && edge.getSource().equals(target))) {

            return edge.getWeight();
```

```
        }
    }
    throw new RuntimeException("Should not happen");
}

private List<Vertex> getNeighbors(Vertex node) {
    List<Vertex> neighbors = new ArrayList<Vertex>();
    for (Edge edge : edges) {
        if (edge.getSource().equals(node)
                && !isSettled(edge.getDestination())) {
            neighbors.add(edge.getDestination());
        }
        if (edge.getDestination().equals(node)
                && !isSettled(edge.getSource())) {
            neighbors.add(edge.getSource());
        }
    }
    return neighbors;
}


private Vertex getMinimum(Set<Vertex> vertexes) {
    Vertex minimum = null;
    for (Vertex vertex : vertexes) {
        if (minimum == null) {
            minimum = vertex;
        } else {
            if (getShortestDistance(vertex) < getShortestDistance(minimum)) {
                minimum = vertex;
            }
```

```
        }
    }
    return minimum;
}


private boolean isSettled(Vertex vertex) {
    return settledNodes.contains(vertex);
}

private int getShortestDistance(Vertex destination) {
    Integer d = distance.get(destination);
    if (d == null) {
        return Integer.MAX_VALUE;
    } else {
        return d;
    }
}

/*
 * This method returns the path from the source to the selected target and
 * NULL if no path exists
 */
public LinkedList<Vertex> getPath(Vertex target) {
    LinkedList<Vertex> path = new LinkedList<Vertex>();
    Vertex step = target;

    // check if a path exists
    if (predecessors.get(step) == null) {
        return path;
```

```
        }
        path.add(step);
        while (predecessors.get(step) != null) {
            step = predecessors.get(step);
            path.add(step);



        }
        // Put it into the correct order
        Collections.reverse(path);

        return path;
    }

    public Map<Vertex, Integer> getDistance() {
        return distance;
    }
}
```