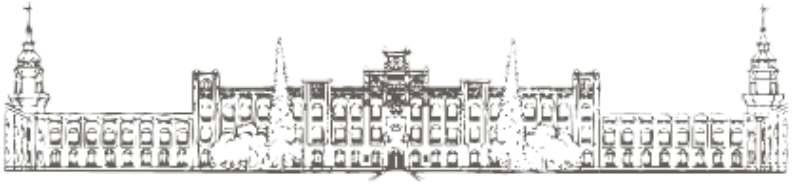

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
КАФЕДРА ІНФОРМАТИКИ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ

Курсова робота з освітнього компоненту
«Моделювання систем. Курсова робота»
Тема: Моделювання процесу передачі мовних пакетів у цифровій системі зв’язку. Формалізація систем масового обслуговування

Керівник:
Асистент А. Ю. Дифучин

«Допущено до захисту»
_____

«___» _____ 2024 р.

Захищено з оцінкою
_____

Члени комісії:

_____ Інна СТЕЦЕНКО

_____ Антон ДИФУЧИН

Виконавець:
Бабіч Денис Володимирович
студент групи __ІП-13__
залікова книжка № __1402__
_____

«17» грудня 2024 р.

Київ – 2024

ЗАВДАННЯ

Виконати дослідження для такої системи:

У системі передачі цифрової інформації передається мовлення в цифровому виді. Мовні пакети передаються через два транзитних канали, буферуючись у накопичувачах перед кожним каналом. Час передачі пакета по каналу складає 5 мс. Пакети надходять через 6±3 мс. Пакети, що передавалися більш 10 мс, на виході системи знищуються, тому що їхня поява у декодері значно знизить якість переданого мовлення. Знищення більш 30% пакетів неприпустимо. При досягненні такого рівня система за рахунок ресурсів прискорює передачу до 4 мс на канал. При зниженні рівня до прийнятного відбувається відключення ресурсів. Визначити частоту знищення пакетів і частоту підключення ресурсу.

Для вищезазначеної системи:

1) розробити опис концептуальної моделі системи;
2) виконати формалізацію опису об'єкта моделювання в термінах визначеного у завданні формалізму;
3) розробити алгоритм імітації моделі дискретно-подійної системи у відповідності до побудованого формального опису;
4) для доведення коректності побудованого алгоритму виконати верифікацію алгоритму імітації;
5) визначити статистичні оцінки заданих характеристик моделі, що є метою моделювання;
6) провести експериментальне дослідження моделі;
7) інтерпретувати результати моделювання та сфомулювати пропозиції щодо поліпшення функціонування системи;
8) зробити висновки щодо складності розробки моделі та алгоритму імітації на основі використаного формалізму, отриманих результатів моделювання та їх корисності.

АНОТАЦІЯ

Бабіч Денис Володимирович. Курсова робота на тему «Моделювання процесу передачі мовних пакетів у цифровій системі зв’язку. Формалізація систем масового обслуговування».

Метою даної курсової роботи є дослідження роботи системи передачі цифрового мовлення, а саме визначення частоти знищення пакетів і частоти підключення додаткових ресурсів для забезпечення стабільної роботи системи.

Текстова частина курсової роботи складається із вступу, розробки концептуальної моделі, формалізованої моделі системи, алгоритмізації та програмної реалізації імітаційної системи, експериментального дослідження моделі, інтерпретації результатів моделювання, висновків, списку використаних джерел, додатків А та Б. В тексті роботи оформлено 41 рисунків, 5 таблиць.

У розділі розробки концептуальної моделі описано модель системи, процес обробки пакетів, механізм регулювання та вхідні й вихідні змінні для експериментів.

У розділі розробки формалізованої моделі представлено формалізм мереж масового обслуговування, структуру системи, динаміку компонентів і методи визначення вихідних змінних для експериментів.

У розділі алгоритмізації та програмної реалізації імітаційної системи побудовано модель мережі, що враховує затримки та додаткові ресурси, верифікація підтвердила коректність роботи.

У розділі експериментального дослідження оцінено перехідний період, оптимальну кількість прогонів та проведено регресійний аналіз з перевіркою рівняння регресії.

У розділі інтерпретації результатів моделювання було проведено аналіз основних факторів, що впливають на ефективність роботи системи, та оцінено їхній вплив на ключові показники.

Ключові слова: ІМІТАЦІЙНА МОДЕЛЬ, МЕРЕЖА МАСОВОГО ОБСЛУГОВУВАННЯ, ДЕКОДЕР, РЕГРЕСІЯ, КРИТЕРІЙ ФІШЕРА, КРИТЕРІЙ СТЬЮДЕНТА.

ЗМІСТ

ПОСТАНОВКА ЗАДАЧІ

У системі передачі цифрової інформації передається мовлення в цифровому виді. Мовні пакети передаються через два транзитних канали, буферуючись у накопичувачах перед кожним каналом. Час передачі пакета по каналу складає 5 мс. Пакети надходять через 6±3 мс. Пакети, що передавалися більш 10 мс, на виході системи знищуються, тому що їхня поява у декодері значно знизить якість переданого мовлення. Знищення більш 30% пакетів неприпустимо. При досягненні такого рівня система за рахунок ресурсів прискорює передачу до 4 мс на канал. При зниженні рівня до прийнятного відбувається відключення ресурсів. Визначити частоту знищення пакетів і частоту підключення ресурсу.

ВСТУП

У сучасних умовах швидкого розвитку інформаційних технологій, системи передачі цифрової інформації відіграють ключову роль у забезпеченні ефективного зв’язку між користувачами. Цифрові системи, що використовуються для передачі голосу та аудіо-контенту загалом, мають працювати у реальному часі та забезпечувати мінімальні затримки, навіть умовах значного навантаження. Ефективність таких систем залежить від багатьох чинників, зокрема від часу передачі даних, частоти надходження пакетів, а також здатності системи динамічно адаптуватися до змін у навантаженні [1].

Метою даної курсової роботи є дослідження роботи системи передачі цифрового мовлення, а саме визначення частоти знищення пакетів і частоти підключення додаткових ресурсів для забезпечення стабільної роботи. Для цього проводиться моделювання процесу передачі мовних пакетів через два транзитні канали з урахуванням буферизації та динамічної адаптивності системи.

Для вирішення завдання використовується нотація мережі масового обслуговування, а також програмні засоби, розроблені мовою C#, які реалізують функціональність формалізму масового обслуговування для проведення експериментів. Результати досліджень дозволяють визначити залежності між параметрами роботи системи, зокрема частотою втрати пакетів та необхідністю залучення додаткових ресурсів для забезпечення якісної передачі мовлення.

Отримані результати можуть бути використані не лише для вдосконалення систем передачі цифрового мовлення, але й для побудови нових моделей комунікаційних систем, що працюють у реальному часі. Вони також сприятимуть підвищенню загальної надійності цифрових комунікаційних мереж та забезпеченню високої якості обслуговування користувачів за умов зростаючого навантаження та збільшення обсягів переданих даних [1].

1 РОЗРОБКА КОНЦЕПТУАЛЬНОЇ МОДЕЛІ

1.1. Загальний огляд та опис задачі

Метою задачі є аналіз ефективності роботи системи передачі цифрового мовлення, визначення частки втрачених пакетів через перевищення допустимого часу перебування в системі та оцінка частоти використання додаткових ресурсів для забезпечення якості передачі. Це дозволяє оцінити надійність системи та оптимізувати її роботу за рахунок регулювання часу передачі.

У системі цифрової передачі інформації здійснюється передача мовлення в цифровому вигляді. Передача даних відбувається у вигляді мовних пакетів, які проходять через два послідовних транзитних канали. Перед кожним каналом пакети накопичуються у буферних накопичувачах. Час передачі одного пакета через кожен канал за стандартних умов становить 5 мс. Інтервал надходження пакетів у систему є випадковою величиною, яка підпорядковується нормальному закону розподілу з середнім значенням 6 мс і стандартним відхиленням 3 мс. Пакети, які перебувають у системі більше ніж 10 мс, знищуються на моменті потрапляння у декодер, оскільки така затримка значно погіршує якість мовлення. Для забезпечення нормальної роботи системи втрати пакетів не повинні перевищувати 30%, а у випадку перевищення цього рівня, система автоматично активує додаткові ресурси, які прискорюють передачу пакетів через кожен канал до 4 мс. Після зниження втрат до прийнятного рівня, який становить ті ж 30%, додаткові ресурси відключаються.

Необхідно визначити два основні показники системи:

- Частоту знищення пакетів – частку пакетів, які були знищені через перевищення допустимого часу перебування в системі (більше 10 мс);
- Частоту підключення додаткових ресурсів – частоту активації прискореного режиму передачі через перевищення допустимого рівня втрат (30%).

1.2. Концептуальна модель

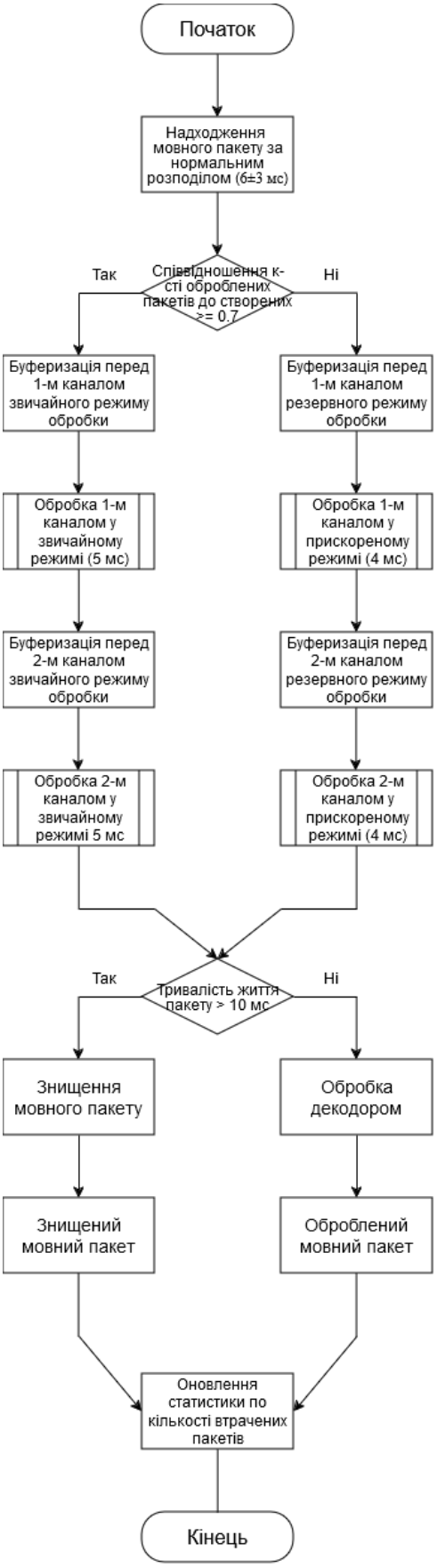

Рисунок 1.1 – Концептуальна модель обробки одного пакету у вигляді діаграми

У цій задачі основна особливість реалізації полягає в тому, що для оцінки роботи системи використовується частка успішно оброблених пакетів, яка має бути не меншою за 70%, що є оберненим значенням до частки знищених пакетів у 30%, яка згадується в умові завдання. Підставою для впровадження оберненої умови є спрощення процесу розрахунків, таким чином, що замість відстеження кількості втрачених даних система контролює, скільки пакетів вдалося доставити вчасно. Це значення вираховується як кількість пакетів, що пройшли через докер розділене на кількість оброблених пакетів на даний момент часу, тобто це сума тих, які вже були знищені, або успішно оброблені.

Передача мовних пакетів відбувається через два послідовних канали, перед якими стоять буфери, але при цьому існують 2 логічні маршрути обслуговування – стандартний та резеверний режими. У стандартному режимі кожен пакет проходить через канал за 5 мс, проте через випадковість інтервалів надходження пакетів (нормальний розподіл із середнім 6 мс і стандартним відхиленням 3 мс) можливі ситуації, коли частина пакетів затримується в системі понад 10 мс, що робить їх непридатними для передачі, що фактично є єдиним місцем у системі, де можуть виникати відмови, бо канали обробки мають буфери необмеженої довжини за умовою завдання.

Якщо частка успішно оброблених пакетів опускається нижче 70%, то система автоматично переходить у прискорений режим роботи. Під цим розуміється те, що наступний мовний пакет, який потрапив у систему автоматично відправляється на обробку у резервний маршрут. У цьому режимі додаткові ресурси скорочують час проходження пакетів через кожен канал до 4 мс, що дозволяє відновити співвідношення успішно оброблених пакетів до знищених. Коли частка успішних пакетів повертається до 70%, або більше, то система вимикає додаткові ресурси і повертається до стандартного режиму роботи. Такий підхід забезпечує стабільну роботу системи навіть за умов змінного навантаження.

1.3. Опис вхідних та вихідних змінних

Вхідними змінними є змінні з тексту умови задачі, їх перелік та опис можна побачити у таблиці 1.1.

Таблиця 1.1 – Перелік вхідних змінних

Позначення параметру	Значення-аргумент	Опис
$T_{\text{надх.}}$	6 ± 3 мс	Нормальний закон розподілу, за допомогою якого задана швидкість надходження нових пакетів у систему
W	1	Кількість пакетів, які надходять одночасно до системи
$t_{\text{макс. життя}}$	10 мс	Максимально допустима тривалість життя пакету у системі
$t_{\text{обр. стандарт.}}$	5 мс	Стандартний час обробки каналу
$t_{\text{обр. резерв.}}$	4 мс	Резервний час обробки каналу
$\eta_{\text{стандарт.}}$	>= 70%	Мінімально-допустимий відсоток успішності обробки пакетів
$\eta_{\text{резерв.}}$	< 70%	Максимальний відсоток при якому задіяні резервні потужності
L_i	∞	Максимальна довжина буферу-черги для кожного каналу обробки

Вихідними параметрами є ті значення, які отримуються у результаті експериментального дослідження побудованої моделі, їх список можна побачити у таблиці 1.2.

Таблиця 1.2 – Перелік вихідних змінних

Позначення параметру	Опис
$N_{\text{створених}}$	Кількість створених пакетів у системі за час моделювання
N_{i}	Кількість оброблених пакетів кожним каналом обробки
$N_{\text{знищених}}$	Кількість знищених пакетів за час моделювання
$n_{\text{знищення}}$	Частота знищення пакетів у системі
$N_{\text{активацій осн.}}$	Кількість активацій основних потужностей
$N_{\text{активацій резерв.}}$	Кількість активацій резервних потужностей
$N_{\text{деактивацій осн.}}$	Кількість деактивацій основних потужностей
$N_{\text{активацій резерв.}}$	Кількість деактивацій резервних потужностей
$T_{i\ \text{зайнятості}}$	Загальний час зайнятості маршруту системи
$n_{\text{підключення}}$	Частота використання резервного маршруту обробки
$B_{i\ \text{зайнятості}}$	Середня зайнятість кожного каналу обробки
$T_{\text{середній}}$	Середній час життя пакету у системі
$Q_{i\ \text{середня}}$	Середня довжина черги перед кожним процесом

1.4. Проміжні висновки до розділу

У розділі було розроблено концептуальну модель системи для загального її аналізу, було описано процес роботи, логіку обробки пакетів, а також логіку механізму регулювання роботи моделі за допомогою додаткових ресурсів для забезпечення оптимальної якості передачі. Також було розглянуто вхідні та вихідні змінні, що використовуватимуться надалі під час проведення експериментальних досліджень.

2 ФОРМАЛІЗОВАНА МОДЕЛЬ СИСТЕМИ

2.1. Загальний огляд формалізму мережі масового обслуговування

Мережі масового обслуговування є потужним інструментом для аналізу та моделювання систем, які здатні працювати у різних сферах застосування. Формалізм та нотація таких мереж дозволяють точно описувати структуру системи, її динаміку, а також взаємодію між її компонентами [2].

Однією з ключових складових формалізму є опис вузлів системи, які виступають місцями обслуговування завдань. Кожен такий вузол характеризується певними параметрами, зокрема, типом черги перед ним (наприклад, FIFO – першим прийшов, першим обслуговується, LIFO – останнім прийшов, першим обслуговується, або обслуговування за якимось іншими заданими параметрами), кількістю підпроцесів у вузлі та розподілом часу обслуговування. Ці та інші параметри визначають, як завдання будуть оброблятися в системі [2].

Вузли пов’язані між собою маршрутами, які визначають порядок і правила пересування завдань між точками обслуговування. Для опису ймовірності такого пересування використовуються асоціативні значення, де кожен перехід пов’язаний з ймовірністю потрапляння до нього, наявністю, або відсутністю блокування з заданим предикатом, параметрами якого можуть бути поточний стан всієї системи, якогось конкретного вузла, або потоного завдання.

Ключовою перевагою такого формалізму є можливість аналітичного визначення характеристик системи, які можна отримати за допомогою комп'ютерних симуляцій, що особливо корисно в реальних системах із складною архітектурою та багатьма вхідними й вихідними змінними [2].

Таким чином, формалізм і нотація мереж масового обслуговування є універсальними інструментами, які дозволяють детально описувати складні системи та визначати їхню ефективність і знаходити способи оптимізації [2].

Всі можливі елементи нотації МО можна побачити на рисунку 2.1

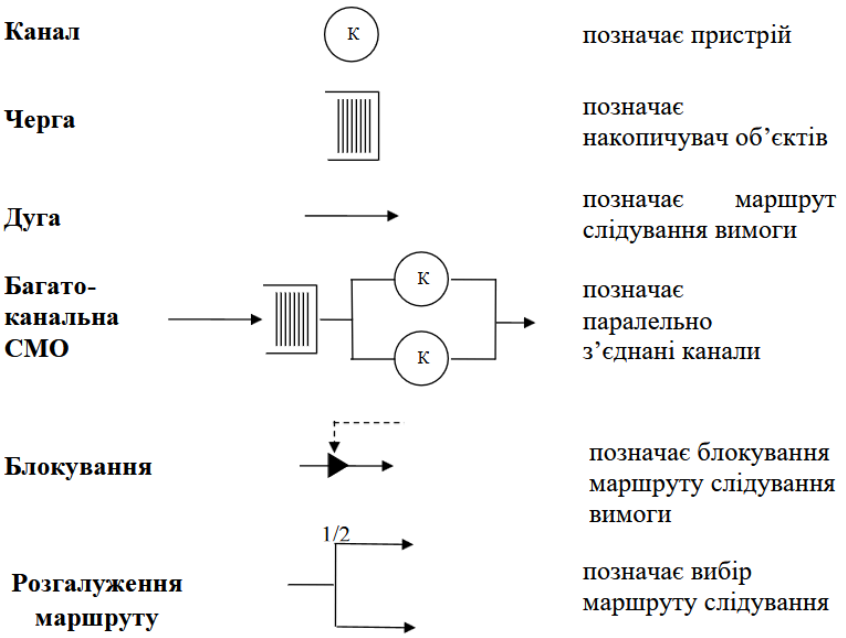

Рисунок 2.1 – Елементи мережі масового обслуговування [2]

2.2. Побудова формалізованої моделі

Формалізована модель мережі масового обслуговування наведена на рисунку 2.2.

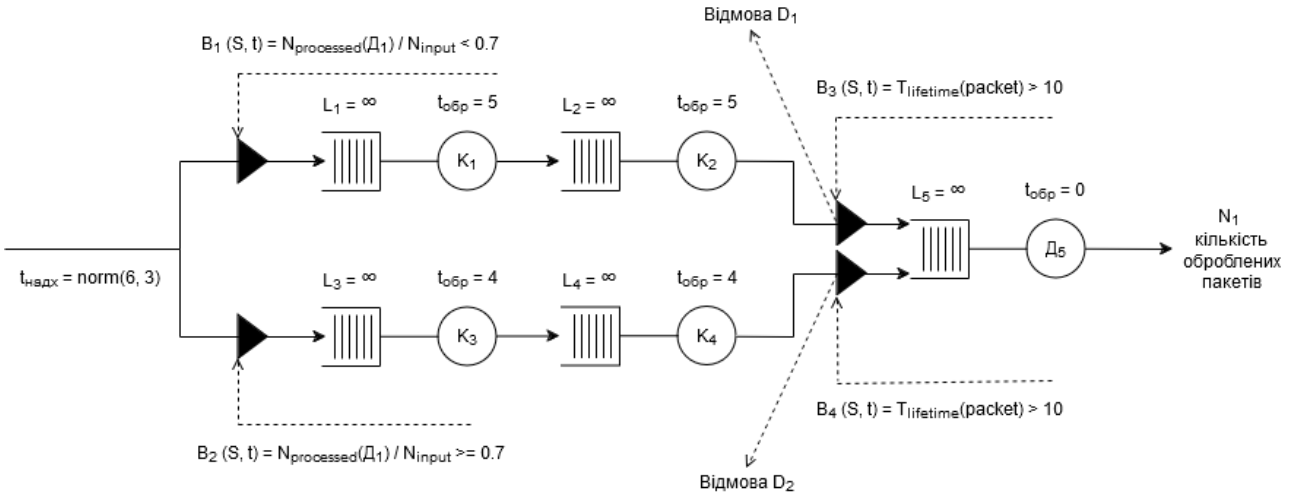

Рисунок 2.2 – Формалізована модель до завдання

Загалом, система складається з надходження нових пакетів із заданим нормальним законом розподілу з відповідними параметрами та має 2 маршрути обробки пакетів, на кожному з яких розташовується по 2 одноканальні системи масового обслуговування, які є каналами обробки пакетів, а також

одноканального декодеру, після якого відбувається вихід з системи та підрахунок кількості оброблених пакетів.

Система працює таким чином, що завдання потрапляють на обробку, але яким саме способом вони будуть оброблені залежить від стану системи, який визначається параметрами у заданих предикатах блокування перед кожним маршрутом. Умовами предикатів B_1 та B_2, які розташовуються перед звичайним та резервним режимами роботи відповідно, є загальний стан системи й застосовується логіка розрахунку відсотка успішно оброблених пакетів за допомогою таких параметрів, як кількість оброблених пакетів декодором на поточний момент часу та кількості створених пакетів станом на зараз. Як результат, рішення приймається на основі того, чи відсоток успішно оброблених завдань у системі перевищує поріг у 70%, що є зворотною умовою до зазначеної в завданні, де 30% визначено як максимально допустимий відсоток відмов, але при цьому зберігається та сама логіка перемикання ресурсів системи, бо ці умови є рівноцінними.

Далі пакет йде по маршруту, обробляється каналами $К_1$ та $К_2$, якщо ймовірність успіху у системі більше, або дорівнює 70%, або йде резервним маршрутом і проходить через канали $К_3$ й $К_4$. Для всіх каналів встановлені необмежені черги, оскільки, згідно до завдання, пакети можуть буферизуватися перед кожним каналом.

Після каналів, пакети відправляються до декодера $Д_1$, перед яким стоять блокування B_3 й B_4, які створюють відмови D_1 і D_2 у випадку, якщо тривалість життя повідомлення довше 10 мс. Наявність декодера у моделі пов’язана виключно з необхідністю встановлення блокування на шляху слідування пакетів, що можливо зробити лише між системами масового обслуговування, тому й тривалість обробки задано константним значенням 0.

Завершується обробка пакету у системі виходом у N_1, де відбувається підрахунок успішно оброблених пакетів.

2.3. Алгебраїчний опис вихідних змінних

Завданням моделювання є визначення таких статистичних величин, як частота знищення пакетів системі ($n_{\text{знищення}}$), загальний час зайнятості маршруту у системі ($T_{\text{зайнятості}}$) та частота підключення додаткового ресурсу.

Обчислення частоти знищених пакетів (n) виконується як для системи виключно з лінійними зв'язками та наведено у формулі 2.1 [3].

$$n_{\text{знищення}} = \frac{N_{\text{знищених}}}{N_{\text{загальна}}} \tag{2.1}$$

де $N_{\text{знищених}}$ – кількість знищених пакетів у системі, $N_{\text{загальна}}$ – загальна кількість пакетів, що дійшли до декодера за час моделювання.

Обчислення загального часу зайнятості ($T_{\text{зайнятості}}$) маршруту обробки наведено у формулі 2.2.

$$T_{\text{зайнятості}} = \sum_{1}^{n} t_i + \Delta t_{\text{системне}} \tag{2.2}$$

де t_i – стан часу моделі на поточний момент, $\Delta t_{\text{системне}}$ – різниця між поточним часом системи та часом активації маршруту.

Обчислення частоти підключення способу обробки пакету ($n_{\text{підключення}}$) маршруту обробки наведено у формулі 2.3.

$$n_{\text{підключення}} = \frac{N_{\text{активацій резерв.}}}{T_{\text{зайнятості}}} \tag{2.3}$$

де $N_{\text{активацій резерв.}}$ – кількість підключень резервного маршруту обробки, $T_{\text{зайнятості}}$ – час роботи маршруту у симуляції системи.

2.4. Проміжні висновки до розділу

У розділі було представлено формалізм мереж масового обслуговування, було розглянуто структуру системи, динаміку її компонентів, і логіку розподілу та обробки завдань. Також було описано алгебраїчні методи визначення вихідних змінних, що будуть використані для експериментальних досліджень.

3 АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ СИСТЕМИ

3.1. Опис алгоритму імітації мережі масового обслуговування

Стохастична мережа масового обслуговування (МО) складається з взаємопов'язаних систем масового обслуговування (СМО), у яких об'єкти (або вимоги), що обслуговуються, переходять з однієї СМО до іншої з певною ймовірністю. Кожна система масового обслуговування являє собою один або кілька обслуговуючих пристроїв з чергою. Кожна СМО має лише одну чергу й кількість місць у черзі може бути обмеженою або необмеженою. У випадках, коли відсутні обмеження на кількість місць у черзі в реальній системі, черга моделюється як необмежена, навіть якщо вона має кілька місць. Обмеження на кількість місць застосовуються, якщо необхідно моделювати ситуації, коли вимога не може бути прийнята в СМО через відсутність вільного місця [2].

У теорії систем масового обслуговування традиційно прийнято називати об'єкти, що обслуговуються, вимогами. У розімкнутій мережі МО вимоги надходять ззовні й після обробки покидають мережу. У замкнутій мережі МО фіксована кількість вимог постійно перебуває всередині, переміщуючись між СМО, але не залишаючи мережу. Алгоритм імітації дискретно-подійної системи має базуватися на принципах, які забезпечують високу ефективність, тому найкращим підходом є просування часу до найближчої події з переходом моделі від однієї події до іншої. Використання дискретного кроку часу в такому алгоритмі недоцільне, оскільки в стохастичних системах це може призводити до великої кількості непродуктивних обчислень та неточного моделювання. Крок часу має бути настільки малим, щоб за цей період відбулася не більше ніж одна подія. У випадку стохастичних часових затримок інтервали між подіями можуть бути як дуже малими, так і великими. Наприклад, для рівномірного розподілу існує найменше можливе значення інтервалу, а для експоненціального розподілу такої межі немає, тому округлення до фіксованого мінімального кроку призводить до втрати точності. Отже, просування до найближчої події є оптимальним підходом як по швидкодії, так і точності відтворення системи [2].

Функціонування мереж масового обслуговування підпорядковується простим правилам. Вимоги, що надходять на обслуговування, послідовно проходять через СМО відповідно до заданого маршруту. На вході кожної СМО вимога намагається зайняти один із вільних пристроїв, які працюють паралельно. Правило вибору пристрою може бути задане окремо, враховуючи специфіку реальної системи. Якщо всі пристрої однакові, вимога перевіряє їх по черзі, займаючи перший доступний. Якщо в момент надходження вимоги всі пристрої зайняті, вона намагається потрапити в чергу або покидає мережу, залишаючись необслуговуваною. У випадках, коли кілька подій відбуваються одночасно, необхідно враховувати пріоритети подій для правильного їх відтворення. Наприклад, якщо одночасно мають відбутися «надходження нового запиту» та «завершення обслуговування», може виникнути ситуація, коли новий запит отримує відмову через обмеження черги, хоча вже через мить пристрій звільниться. Якщо порядок виконання подій жорстко фіксований, це може призводити до значного збільшення кількості відмов порівняно з реальними умовами [2].

Псевдокод алгоритму імітації можна побачити на рисунку 3.1.

```
while timeCurrent < simulationTime:
    iterationsCount += 1

    for module in modules:
        module.updateStatistics(timeNext - timeCurrent)

    timeCurrent := timeNext

    for module in modules:
        module.updateTimeline(timeCurrent)

    nextModules := modules where module.timeNext == timeCurrent

    for module in nextModules:
        module.completeTask()

    timeNext := minimum timeNext across modules
endloop
```

Рисунок 3.1 – Псевдокод алгоритму імітації мережі масового обслуговування

3.2. Опис програмної реалізації мережі масового обслуговування

Програмна реалізація мережі масового обслуговування побудована за допомогою мови C# з використанням виключно стандартних бібліотек .NET.

Мережа масового обслуговування будується за допомогою відповідних компонентів. Діаграма базових класів компонентів наведена на рисунку 3.2.

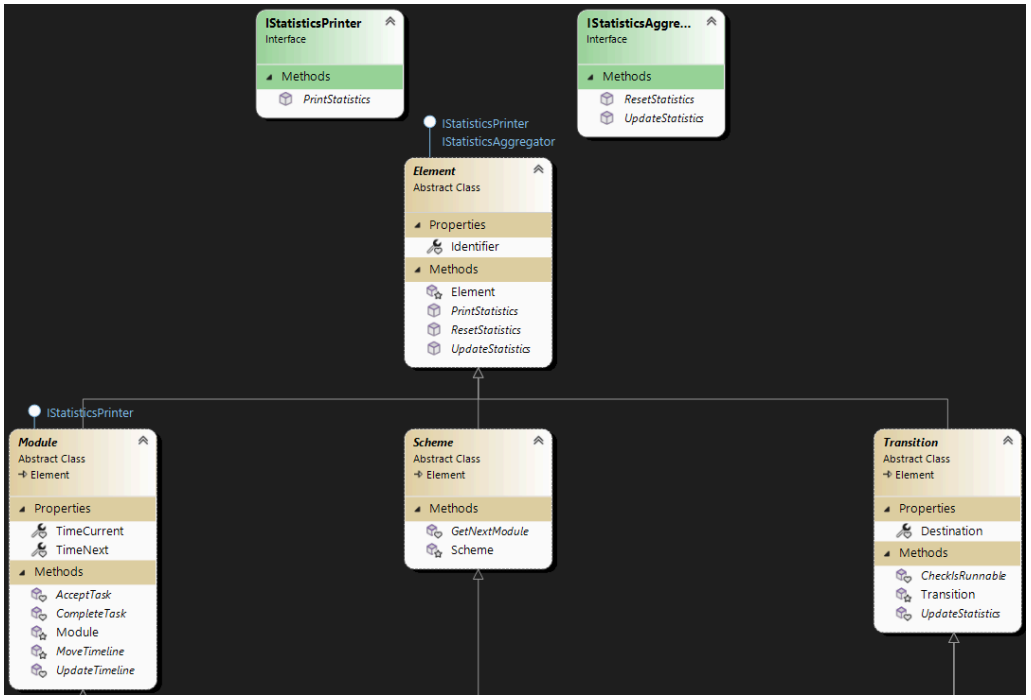

Рисунок 3.2 – Діаграма класів основних складових мережі масового обслуговування

Головним чином, було реалізовано 3 типи, які є нащадками класу Element:

– Module – базовий клас для модулів системи, таких як Create, Processor та Dispose;
– Scheme – базовий клас з’єднання модулів між собою за допомогою переходів;
– Transition – базовий клас для переходів, для виконання даної роботи було розроблено 3 переходи: між процесами без блокування, між процесами з блокуванням, для зв’язку з Dispose.

Опис ролі та призначення кожного базового класу у програмній реалізації можна побачити у таблиці 3.1.

Таблиця 3.1 – Опис призначення базових класів

Назва класу	Призначення
Element	Абстрактний базовий елемент, який містить ідентифікатор і реалізує метод для друку статистики, призначений для подальшого успадкування
Module	Абстрактний клас модуля, що успадковує від Element, містить властивості часу та методи для оновлення статистики, управління часом і обробки завдань
Scheme	Абстрактний клас схеми, що успадковує від Element, призначений для отримання наступного модуля на основі завдання
Transition	Абстрактний клас переходу, що успадковує від Element, призначений для перевірки можливості виконання та оновлення статистики переходу між модулями

Конкретні реалізації класу Module можна побачити у таблиці 3.2 й на рисунку 3.3.

Таблиця 3.2 – Опис призначення класів-нащадків Module

Назва класу	Призначення
CreateModule	Клас, що представляє конкретний модуль для створення завдань, відповідає за генерацію нових завдань, управління часом та передачу завдань наступним модулям у схемі
DisposeModule	Клас, що представляє модуль для утилізації завдань, який приймає завдання та фіксує кількість утилізованих завдань
ProcessorModule	Клас, що представляє модуль обробки завдань, який обробляє завдання, управляє чергою, відстежує статистику успішних та неуспішних завдань, а також фіксує статистику

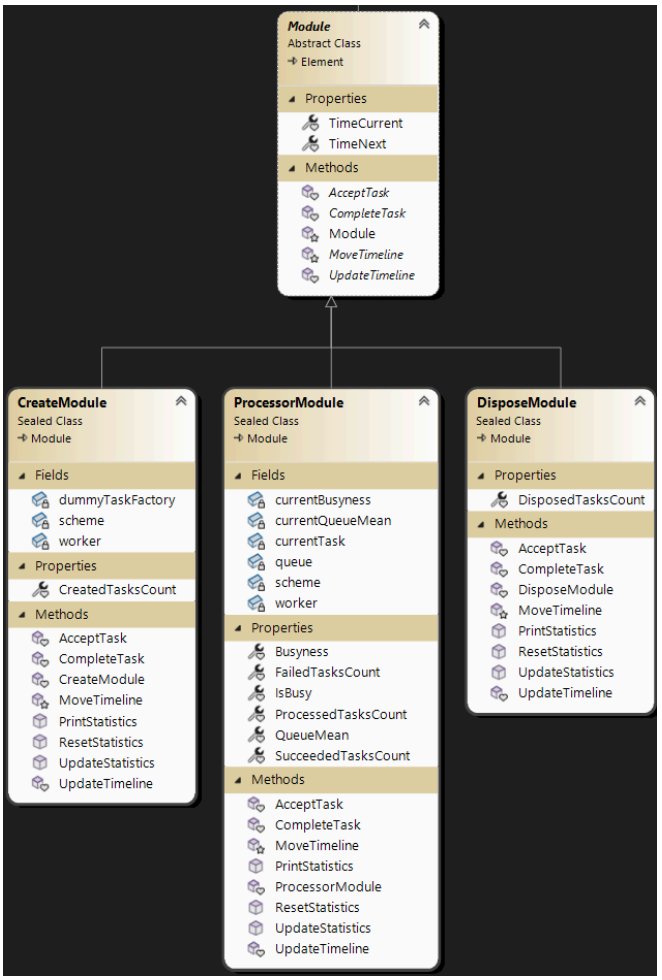

Рисунок 3.3 – Діаграма класів-реалізації конкретних модулів

Окремо можна відмітити логіку виконання завдання модулем-процесом. Оскільки канал може накопичувати завдання у черзі, то має бути реалізований і механізм отримання задач з неї. Відповідний псевдокод наведено на рисунках 3.4 та 3.5.

```
if isBusy:
        if not queue.isFull:
                queue.enqueue(task)
        else:
                failedTasksCount += 1
else:
        isBusy := true
        currentTask := task
        moveTimeline(worker.delay)
endif
```

Рисунок 3.4 – Псевдокод прийому нового завдання

```
succeededTasksCount += 1
finishedTask := currentTask

if queue.isEmpty:
        isBusy := false
        currentTask := null
        timeNext := float.MaxValue
else:
        moveTimeline(worker.delay)
        currentTask := queue.dequeue()
endif

nextModule := scheme.getNextModule(finishedTask)
nextModule.acceptTask(finishedTask)
```

Рисунок 3.5 – Псевдокод виконання завдання з просування часу

Для визначення затримок надходження завдань та часу їх обробки використовуються генератори затримок. Оскільки у завданні потрібні лише нормальний розподіл та константне значення часу роботи процесу, то реалізовані лише 2 відповідні конкретні класи. Діаграму класів затримок можна побачити на рисунку 3.6.

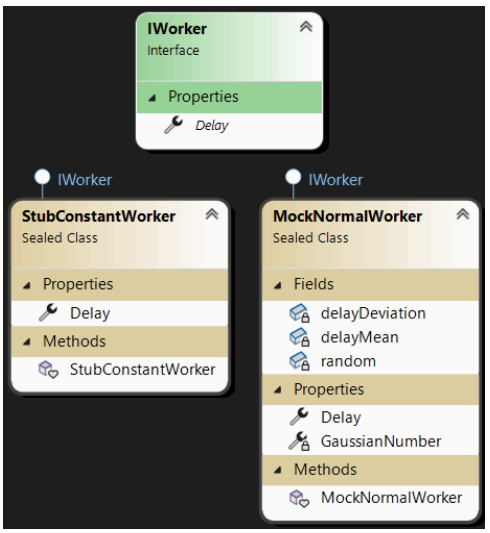

Рисунок 3.6 – Діаграма класів, які реалізують корисне навантаження

За умовою завдання, пакети мовлення, які проходять обробку у системі, можуть буферзиуватися перед каналим обробки, тому був створений клас звичайної черги, яка працює за правилом FIFO (англ. first in, first out – перший прийшов, перший вийшов). Діаграму класів та його члени можна побачити на рисунку 3.7.

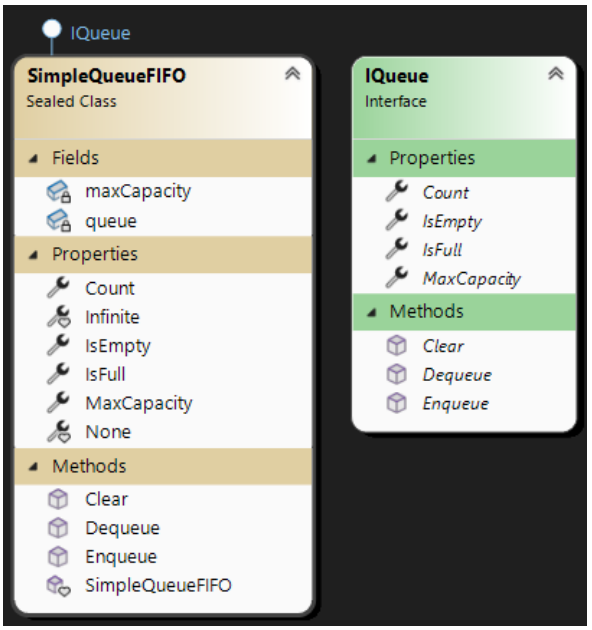

Рисунок 3.7 – Діаграма класу черги

Ще одним важливим елементом системи є схеми з’єднання модулів між собою, функціонал яких реалізований за допомогою класів Scheme та SimpleScheme. Діаграму цих класів разом з їх членами можна побачити на рисунку 3.8.

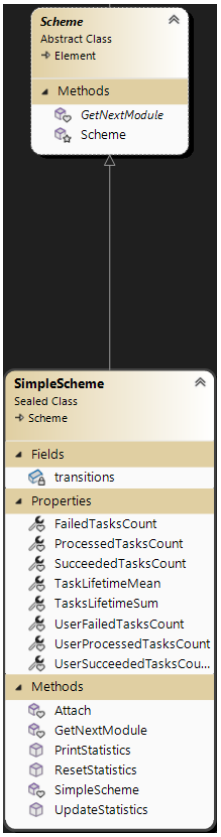

Рисунок 3.8 – Діаграма класу схеми з’єднання модулів

Оскільки схема представляє собою зв’язок виключно на основі детермінованій основі й не має зв’язків на основі ймовірності переходів, то додавання модулів до схеми відбувається без додаткових аргументів. Особливий інтерес викликає метод GetNextModule, який інкапсулює у собі код отримання модуля, до якого поточний елемент має передати пакет. Головна його ідея полягає у тому, щоб почергово перевіряти чи можливо виконати перехід до наступного модуля за допомогою методу checkIsRunnable і якщо такий перехід можливий та кількість потенційних переходів не перебільшує 1, то повертається посилання на цей модуль. Псевдокод методу GetNextModule можна побачити на рисунку 3.9.

```
for transition in transitions:
        if transition.checkIsRunnable(task):
                if nextModule != null:
                        throw exception

                nextModule := transition.destination
                transition.updateStatistics(TransitionStatus.Active, task)
                continue
        endif

        transition.updateStatistics(TransitionStatus.Inactive, task)
endloop

if nextModule == null:
        failedTasksCount += 1
else:
        succeededTasksCount += 1
endif

tasksLifetimeSum += task.lifetime

return nextModule
```

Рисунок 3.9 – Псевдокод методу отримання наступного модуля.

Ітерація у методі GetNextModule відбувається по об’єктах типу Transition, кожен з яких зберігає посилання на модуль, який пов’язаний з цим переходом. Сімейство типів яких можна побачити на рисунку 3.10 у вигляді діаграми класів.

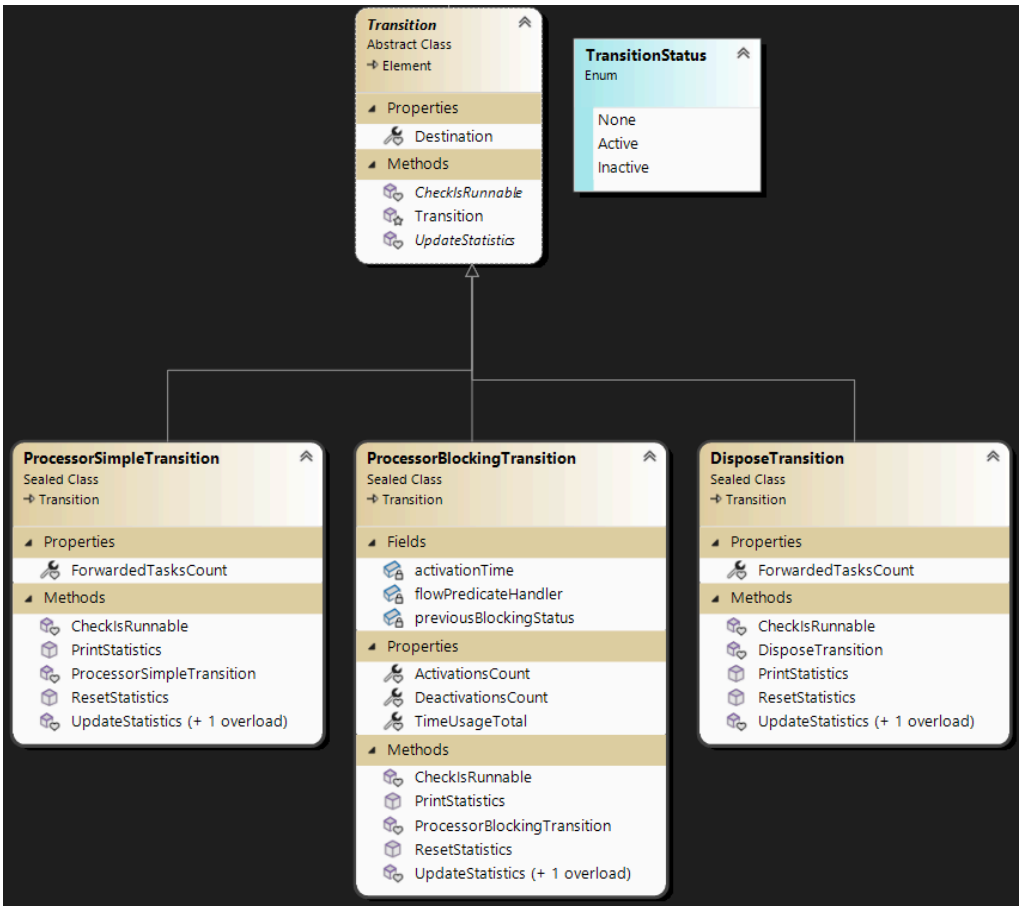

Рисунок 3.10 – Діаграма класів переходів

Варто звернути увагу на метод з ідентифікатором CheckIsRunnable, який визначає чи є активними перехід. Для ProcessorSimpleTransition та DisposeTransition не виконується жодних додаткових перевірок, вони завжди є активними, а ProcessorBlockingTransition має поле предикату, яке відповідає умові блокування шляху та яке викликається всередині методу CheckIsRunnable.

Завданнями, які циркулюють у системі є об’єкти типів, які є нащадками класу DummyTask, яке зберігає важливі системні поля, які відповідають за зберігання поточного статусу об’єкту, а саме тривалість життя пакету та час його створення.

Сам процес створення об’єктів відбувається у модулі CreateModule та відбувається завдяки об’єкту типу PacketDummyTaskFactory, який є реалізацією абстрактної фабрики для об’єктів типу PacketDummyTask.

Відповідні типи, їх поля та зв’язки можна побачити на рисунку 3.11.

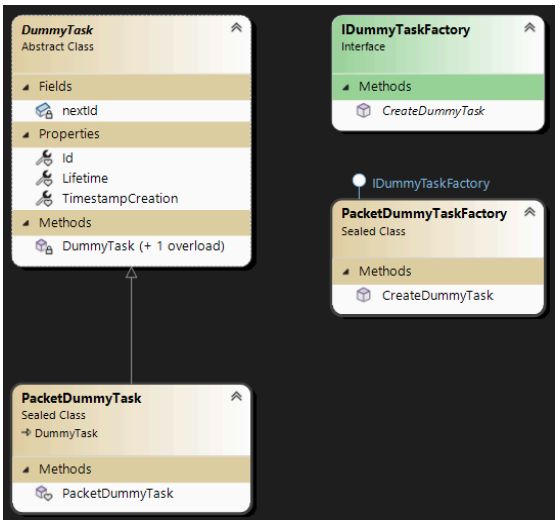

Рисунок 3.11 – Діаграма класів, пов’язаних з DummyTask

Загальна діаграма класів наведена на рисунку 3.12.

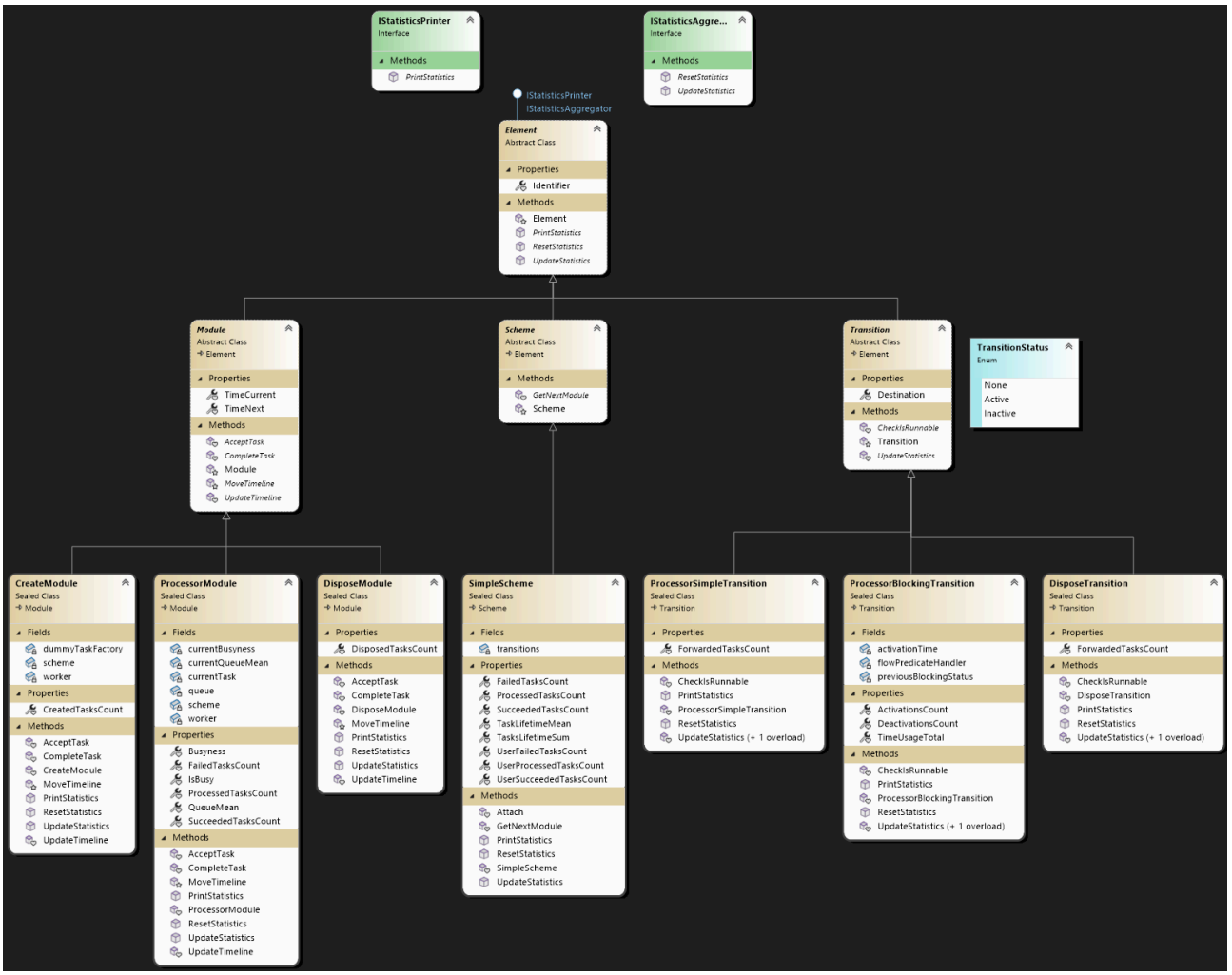

Рисунок 3.12 – Загальна діаграма класів компонентів моделі

3.3. Валідація побудованої моделі

У цифровій системі передачі інформації мовні пакети створюються генератором із середнім інтервалом 6 мс і розподілом ± 3 мс. Вони проходять обробку через два транзитні канали, кожен з яких має два етапи. Перший маршрут (K1, K2) і другий (K3, K4) забезпечують передачу пакетів із часом обробки 5 та 4 відповідно. Перед кожним каналом пакети можуть буферизуватися, тобто вставати у чергу.

Основний транзитний канал використовується, якщо співвідношення успішно оброблених пакетів до створених нижче порогу 0.7 (PIPELINE_BALANCE_THRESHOLD). У разі перевищення цього порогу навантаження перекидається на резервний канал.Система контролює рівень втрат пакетів. Якщо понад 30% пакетів знищуються, активується прискорений режим, який скорочує час обробки до 4 мс на етап. Коли втрати повертаються до прийнятного рівня, система вимикає додаткові ресурси і повертається до звичайного режиму.

Система знищує пакети, якщо їхній загальний час передачі перевищує 10, оскільки такі пакети знижують якість мовлення. Видалення таких пакетів реалізується завдяки механізму відмов, які виникають завдяки встановленому предикату на шляхах виходів з каналів обробки.

Остаточний отриманий код моделі наведений на рисунку 3.13.

```
const float MAX_PACKET_LIFETIME = 10.0f;
const float PIPELINE_BALANCE_THRESHOLD = 0.7f;

DisposeModule dispose = new DisposeModule("DISPOSE");

SimpleScheme processorD5Scheme = new SimpleScheme("D5_SCHEME", new DisposeTransition("D5 ⇒ DISPOSE", dispose));
ProcessorModule processorD5 = new ProcessorModule("PROCESSOR_D5", processorD5Scheme, new StubConstantWorker(0), SimpleQueueFIFO.Infinite);

SimpleScheme processorK2Scheme = new SimpleScheme("K2_SCHEME", new ProcessorBlockingTransition("K2 ⇒ D5", processorD5, (task) ⇒ task.Lifetime > MAX_PACKET_LIFETIME));
ProcessorModule processorK2 = new ProcessorModule("PROCESSOR_K2", processorK2Scheme, new StubConstantWorker(5), SimpleQueueFIFO.Infinite);

SimpleScheme processorK1Scheme = new SimpleScheme("K1_SCHEME", new ProcessorSimpleTransition("K1 ⇒ K2", processorK2));
ProcessorModule processorK1 = new ProcessorModule("PROCESSOR_K1", processorK1Scheme, new StubConstantWorker(5), SimpleQueueFIFO.Infinite);

SimpleScheme processorK4Scheme = new SimpleScheme("K4_SCHEME", new ProcessorBlockingTransition("K4 ⇒ D5", processorD5, (task) ⇒ task.Lifetime > MAX_PACKET_LIFETIME));
ProcessorModule processorK4 = new ProcessorModule("PROCESSOR_K4", processorK4Scheme, new StubConstantWorker(4), SimpleQueueFIFO.Infinite);

SimpleScheme processorK3Scheme = new SimpleScheme("K3_SCHEME", new ProcessorSimpleTransition("K3 ⇒ K4", processorK4));
ProcessorModule processorK3 = new ProcessorModule("PROCESSOR_K3", processorK3Scheme, new StubConstantWorker(4), SimpleQueueFIFO.Infinite);

SimpleScheme createScheme = new SimpleScheme("CRETE_SCHEME");
CreateModule create = new CreateModule("CREATE", createScheme, new MockNormalWorker(6.0f, 3.0f), new PacketDummyTaskFactory());
var defaultPipeline = new ProcessorBlockingTransition("CREATE ⇒ K1", processorK1, (_) ⇒ ((float)processorD5.ProcessedTasksCount / create.CreatedTasksCount) < PIPELINE_BALANCE_THRESHOLD);
var reservePipeline = new ProcessorBlockingTransition("CREATE ⇒ K1", processorK3, (_) ⇒ ((float)processorD5.ProcessedTasksCount / create.CreatedTasksCount) ⩾ PIPELINE_BALANCE_THRESHOLD);
createScheme.Attach(defaultPipeline);
createScheme.Attach(reservePipeline);
```

Рисунок 3.13 – Код побудови моделі

Код запуску симуляції моделі тривалістю 1000 одиниць часу наведено на рисунку 3.14.

```
SystemModelController.Instance.Build(new Module[] { create, processorK1, processorK2, processorK3, processorK4, processorD5, dispose });
SystemModelController.Instance.RunSimulation(SIMULATION_TIME);
```

Рисунок 3.14 – Код побудови та запуску симуляції моделі

Код обрахунку та виводу вихідних показників результатів експериментів наведено на рисунку 3.15.

```
int totalFailuresCount = processorK2Scheme.FailedTasksCount + processorK4Scheme.FailedTasksCount;
int totalProcessedCount = processorK2Scheme.ProcessedTasksCount + processorK4Scheme.ProcessedTasksCount;
Console.WriteLine($"|REPORT| (RESULTS) Packets loss frequency: {(float)totalFailuresCount / totalProcessedCount}");
Console.WriteLine($"|REPORT| (RESULTS) Reserve pipeline connection frequency: {reservePipeline.ActivationsCount / reservePipeline.TimeUsageTotal}");
```

Рисунок 3.15 – Вивід частоти знищення пакетів та частоти підключення додаткового ресурсу

На рисунку 3.16 можна побачити виведені логи системи, де видно, що завдання з ID 20, 21 та 23 були відкинуті на етапі переходу до декордеру, бо час їх життя перевищує встановлене обмеження у 10 одиниць часу. В той час, як пакет з ID 22 відправляється до декодеру, бо відповідає умові.

```
|LOG| (TRACE) [PROCESSOR_K2] sends [#20 (14.965286)] to the []
|LOG| (TRACE) [CREATE] sends [#22 (0)] to the [PROCESSOR_K1]
|LOG| (TRACE) [CREATE] sends [#23 (0)] to the [PROCESSOR_K1]
|LOG| (TRACE) [PROCESSOR_K2] sends [#21 (10.899261)] to the []
|LOG| (TRACE) [PROCESSOR_K1] sends [#22 (5)] to the [PROCESSOR_K2]
|LOG| (TRACE) [CREATE] sends [#24 (0)] to the [PROCESSOR_K1]
|LOG| (TRACE) [PROCESSOR_K1] sends [#23 (9.2425995)] to the [PROCESSOR_K2]
|LOG| (TRACE) [PROCESSOR_K2] sends [#22 (10)] to the [PROCESSOR_D5]
|LOG| (TRACE) [PROCESSOR_D5] sends [#22 (10)] to the [DISPOSE]
|LOG| (TRACE) [CREATE] sends [#25 (0)] to the [PROCESSOR_K1]
|LOG| (TRACE) [PROCESSOR_K1] sends [#24 (8.236649)] to the [PROCESSOR_K2]
|LOG| (TRACE) [PROCESSOR_K2] sends [#23 (14.2425995)] to the []
|LOG| (TRACE) [CREATE] sends [#26 (0)] to the [PROCESSOR_K3]
|LOG| (TRACE) [PROCESSOR_K1] sends [#25 (7.689514)] to the [PROCESSOR_K2]
```

Рисунок 3.16 – Фрагмент з логів системи для перевірки коректності роботи

На рисунку 3.17 демонструється статистика системи після прогону симуляції, де можна побачити кількість активацій та деактивацій заблокованих переходів, час використання кожного маршруту, кількість успішно оброблених пакетів кожним каналом, кількість відмов на маршрутах та деякі інші показники.

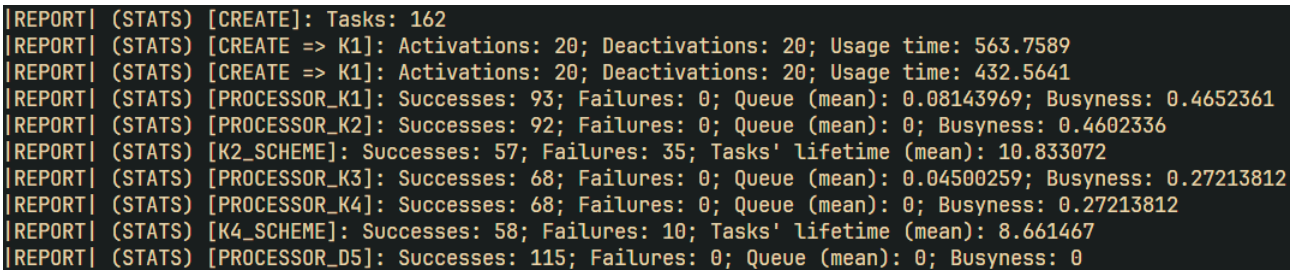

Рисунок 3.17 – Вивід статистики системи після прогону симуляції

Шукані значення статистики, а саме частоту знищення пакетів та частота підключення ресурсів наведена на рисунку 3.18.

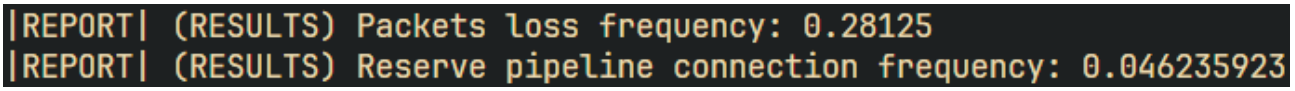

Рисунок 3.18 – Вивід значень шуканих статистичних значень прогону системи

Проаналізувавши отримані значення, можна зробити висновок про коректність роботи побудованої моделі. Оскільки значення частоти величини для лінійних систем співпадає з часткою відмов за математичним означенням, то можна переконатися у тому, що значення втрачених мовних пакетів залишається у межах норми відповідності умови завдання та не перевищує 0.3.

Значення частоти активації резервного маршруту обслуговування теж не перевищує можливий максимальний ліміт та сумарний час роботи обох маршрутів обробки є валідним значенням, яке не перевищує значення загального часу моделювання.

3.4. Верифікація побудованої моделі

Верифікація моделі здійснювалася на основі зміни вхідних параметрів моделі та оцінці отриманих вихідних результатів. Перевірка коректності отриманих значень відбувалася на основі аналітичних міркувань, логічного аналізу та порівняння отриманих даних з стандартними результатами моделювання. Верифікація виконувалася завдяки прогонами системи з часом моделювання у 1000 одиниць та для значення 100 прогонів. Відповідну верифікаційну таблицю можна побачити на рисунку 3.19.

	Вхідні дані							Вихідні дані										
Прогон №	Математичне сподівання	Середньоквадратичне відхилення	Час передачі каналом у звичайному режимі	Час передачі каналом у резервному режимі	Максимальний час життя пакету	Максимальний відсоток знищених пакетів	Мінімальний відсоток успішних пакетів	Кількість вхідних пакетів	Кількість успішно оброблених пакетів	Кількість знищених пакетів (відмов)	Середня тривалість життя пакету при стандартній обробці	Середня тривалість життя пакету при резервній обробці	Середня завантаженість каналів обробки	Середня черга каналів обробки	Кількість залучень резервного ресурсу	Час використання резервного ресурсу	Частота знищення пакетів	Частота підключення ресурсу
1	6	3	5	4	10	30	30	166	117	48	11	8.88	0.38	0.039	16	400	0.29	0.042
2	7	3	5	4	10	30	30	142	103	38	10.68	8.36	0.344	0.023	8	143	0.27	0.06
3	6	2	5	4	10	30	30	166	118	47	10.6	8.22	0.4	0.021	13	213	0.28	0.06
4	6	3	4	4	10	30	30	166	137	28	8.9	8.8	0.33	0.037	3	98	0.17	0.03
5	6	3	5	3	10	30	30	166	117	47	11	6.3	0.37	0.034	21	242	0.29	0.087
6	6	3	5	4	11	30	30	166	118	46	11.37	9	0.39	0.05	13	253	0.28	0.05
7	6	3	5	4	10	25	75	166	124	40	10.77	8.9	0.37	0.035	17	507	0.24	0.03
8	6	3	5	4	10	35	65	166	109	55	11.18	8.9	0.39	0.05	15	309	0.34	0.03

Рисунок 3.19 – Верифікація моделі

Починаючи з другого прогону моделі відбувалися зміни вхідних параметрів моделі для того, щоб переконатися у коректності роботи моделі.

Таким чином, на другій ітерації було збільшено значення математичного сподівання надходження пакетів до системи, що головним чином вплинуло на загальну кількість завдань, які надійшли до системи, а саме кількість мовних пакетів зменшилася, що є цілком очікуваним.

На третій ітерації було зменшено середньоквадратичне відхилення, що впливає на розкид значень і як результат, зменшився час залучення ресурсу, оскільки пакети почали поступати через більш рівномірні проміжки часу.

Метою зміни часу обробки звичайном каналом на чертверій ітерації стала перевірка коректності логіки залучення додаткових ресурсів. Можна констатувати, що перенаправлення ресурсів працює коректно, бо час їх залучення впав до 10%, а частота знищення пакетів впала майже вдвічі. Цей результат пов’язаний з тим, що пакети швидше проходять по системі та менше накопичуються у чергах, що і призводить до зростання часу перебування у ній.

П’ята ітерація цікава тим, що час передачі каналом у резервному режимі було зменшено до 3, що вплинуло на кількість активацій та частоту переходу до маршруту обробки резервними потужностями. Отримані дані є передбачуваними, бо зі зменшенням часу обробки на резервному маршруті, зменшується і час роботи цього резервного маршруту, а отже більша кількість завдань може встигнути пройти ним за той самий проміжок часу.

Збільшення максимального часу життя пакету на шостій ітерації цікава тим, що підтверджує коректність логіки виконання переходів між маршрутами обробки. Згідно до отриманих результатів, час використання резервного ресурсу зменшився, але кількість залучень ресурсу, а отже й частота залишилися майже незмінними у порівнянні зі стандартими параметрами, що вказує на меншу кількість відмов у моменті, бо їх відсоток накопичується повільніше.

Ітерація за номерами 7 та 8 доводять валідність роботи переходів між маршрутами обробки у системі. Так при меншому максимальному відсотку знищених пакетів, час роботи резервного маршруту став більшим. А при більшому значенню цього ж параметра, система має більшу частоту знищених пакетів, бо порогове значення блокування маршрутів було змінене.

3.5. Проміжні висновки до розділу

Побудована модель мережі масового обслуговування успішно відтворює роботу системи з обробки пакетів у заданих умовах, зокрема, враховуючи затримки, черги та використання додаткових ресурсів. Алгоритм імітації, заснований на дискретно-подійному підході, показав високу точність та ефективність, забезпечуючи правильне управління подіями та оптимальне просування часу. Верифікація та зміна вхідних параметрів моделі підтвердили коректність роботи, а отримані результати співпали з очікуваними теоретичними значеннями, що підтверджує правильність реалізації логіки переходів і управління ресурсами.

4 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ

4.1. Визначення перехідного періоду моделі

Визначення перехідного періоду моделі має на меті досягнення стабільної роботи системи. Для статистичних спостережень використовуються лише ті значення відгуку моделі, які відповідають сталому стану системи. Якщо середнє значення відгуку моделі з часом наближається до постійного значення, це свідчить про наявність сталого стану в системі. Час, необхідний для переходу системи до сталого стану, називається перехідним періодом. Його тривалість можна визначити шляхом проведення пробних експериментів з моделлю, для чого будується графік залежності відгуку моделі від часу прогону [2].

Таким чином, час симуляції визначатиметься для моделі зі стандартними вхідними параметрами з умови завдання та часом моделювання від 1000 та до 100000 умовних одиниць. Відповідні графіки оцінки перехідного стану моделі можна побачити на рисунках 4.1 – 4.4.

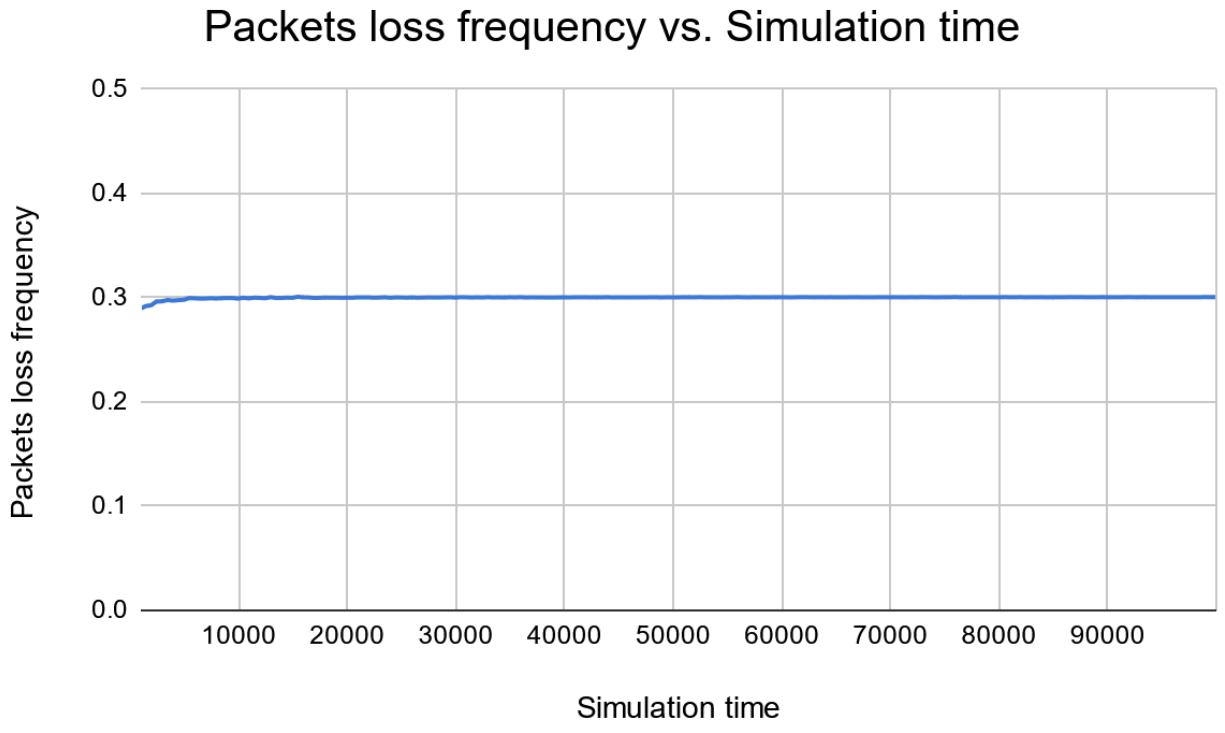

Рисунок 4.1 – Залежність величини частоти знищення пакетів від часу моделювання

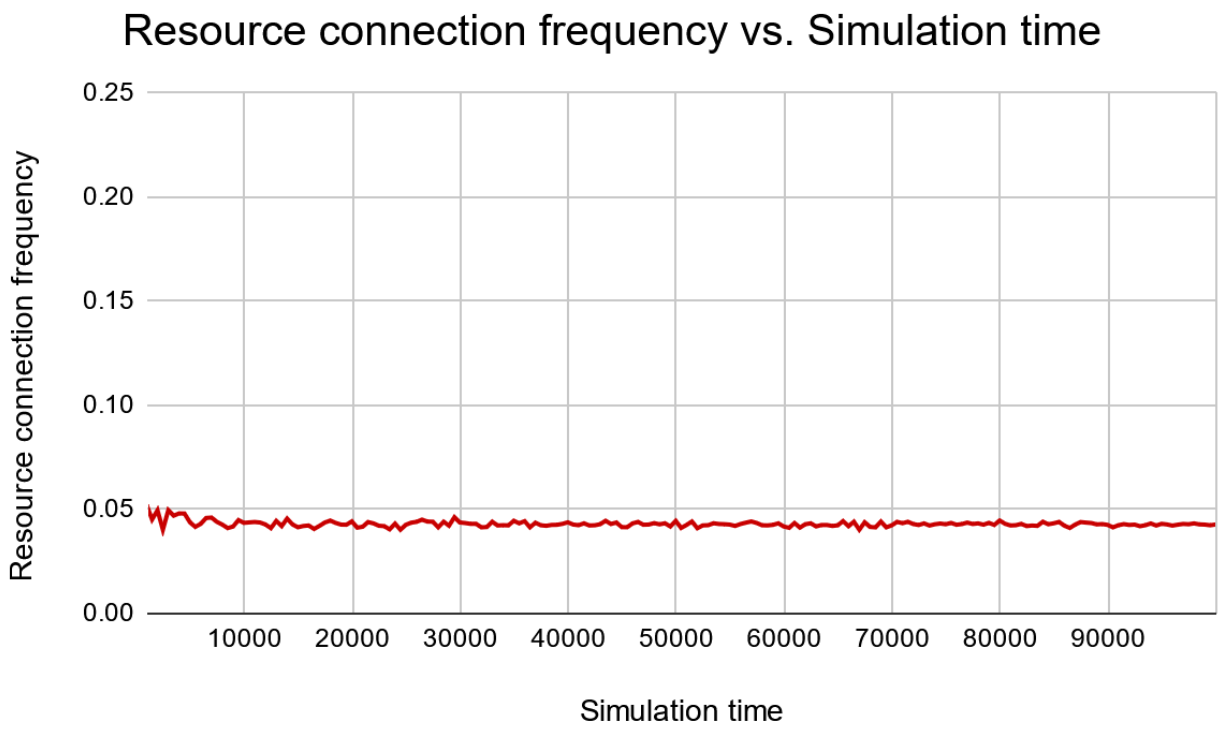

Рисунок 4.2 – Залежність величини частоти підключення резервних ресурсів від часу моделювання

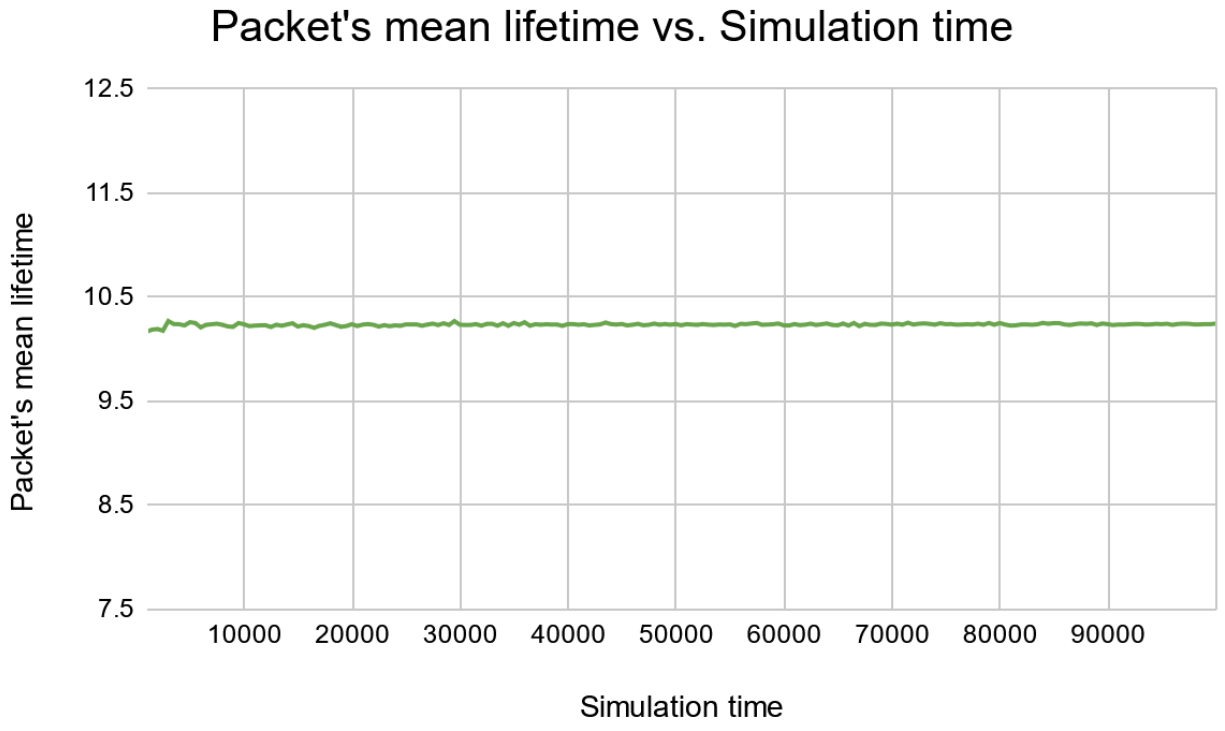

Рисунок 4.3 – Залежність величини тривалості життя пакету від часу моделювання

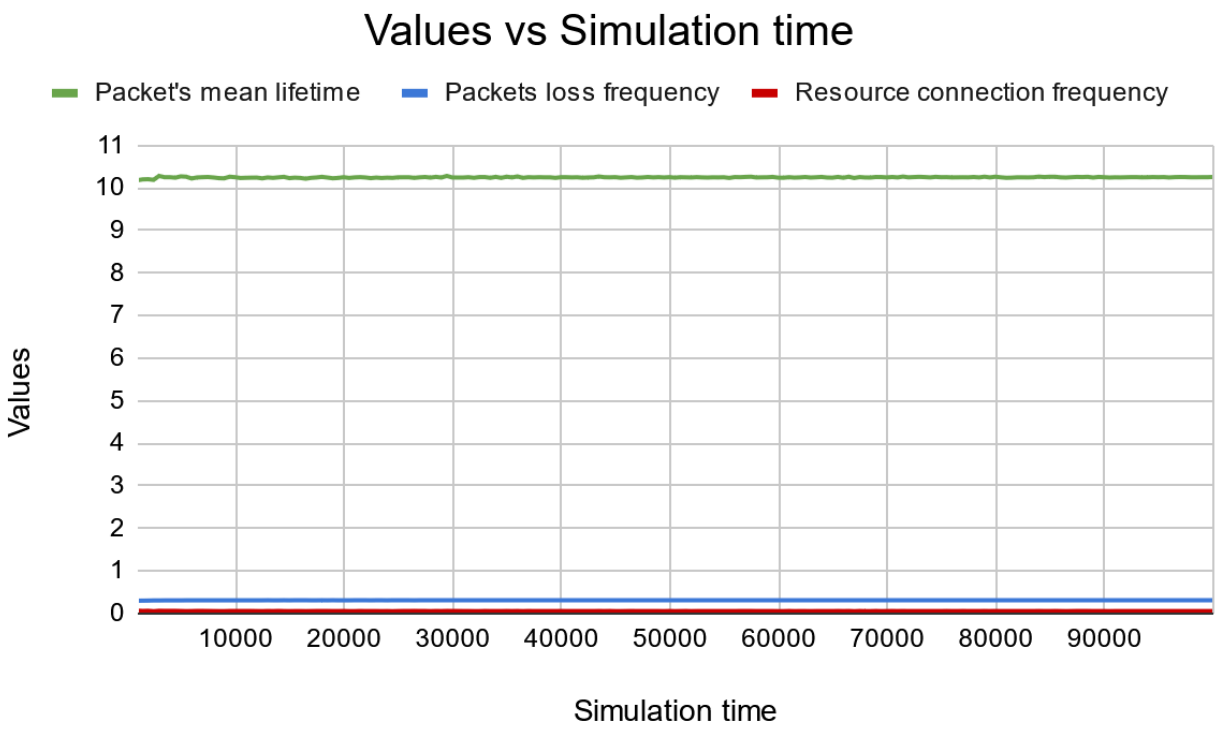

Рисунок 4.4 – Порівняння залежності значень величин від часу моделювання

Проаналізувавши отримані графіки, можна зробити висновок про те, що значення залежності частоти знищення пакетів стабілізується приблизно часі симуляції у 10000 умовних одиниць. А значення частоти залучення додаткових ресурсів стабілізується на позначці у 50000 умовних одиниць, тому для експериментування буде взято значення у 90000 умовних одиниць часу.

Загалом, оскільки точно визначити проміжок часу з якого значення залежності зміни величини від часу стає незначним не вдалося, то час тривалості експерименту буде встановлений у 100000 одиниць часу.

Окремо варто сказати про недоцільність повного обнулення статистики у контексті даного завдання, оскільки сама поведінка моделі переважним чином спирається на її глобальний стан та стан її елементів. Це пов’язано з логікою вибору маршруту обробки пакетів, яка залежить від статистики декодера, який є повноцінною СМО та елементом моделі. Тому для визначення необхідних статистичних, були використані копії даних та буде обраний довгий час роботи системи (100000) після тривалості роботи у 100000 одиниць часу, щоб мінімізувати негативний ефект від початкового розігріву симуляції [2].

4.2. Визначення оптимальної кількості прогонів моделі

Оскільки про закон розподілу відгуку невідомо, то визначення оптимальної кількості необхідних прогонів для точних експериментів над моделлю оцінюється за допомогою нерівності Чебишева без використання аргументу функції Лапласа, яка наведена у формулі 4.1 [2].

$$p = \frac{\sigma^2}{\varepsilon^2(1-\beta)} \tag{4.1}$$

де p – необхідна кількість прогонів, σ^2 – дисперсія відгуку моделі, ε – точність вимірювання моделі, β – довірча ймовірність.

Для проведення аналізу буде використаний відгук моделі – частота втрата пакетів. Для розрахунку необхідної кількості прогонів моделі буде виконано запуск тестової моделі на час симуляції у 1000 одиниць часу у кількості 100 ітерацій для визначення середнього значення та середньоквадратичного відхилення. Код оцінки цих метрик та результат можна побачити на рисунках 4.5 – 4.6.

```python
import numpy as np
import pandas as pd

data = pd.read_csv("coursework/docs/papers/iterations.csv")
values = data['Loss']

mean_value = np.mean(values)
std_deviation = np.std(values)

print(f"{mean_value:.5f} ± {std_deviation:.5f}")
```

Рисунок 4.5 – Оцінка середньоквадратичного відхилення збірки

```
0.28850 ± 0.00300
```

Рисунок 4.6 – Отримані значення середнього арифметичного та середньоквадратичного відхилення

За значення довірчої ймовірності (β) буде взято значення 0.95 й враховуючи те, що точність вимірювання моделі (ε) буде прийнята за доволі високу, тому її можна прирівняти до значення дорівнює стандартному

відхиленню, тому отримаємо наступний вираз обчислення кількості прогонів моделі, який продемонстрований у формулі 4.2 [2].

$$p = \frac{\sigma^2}{\varepsilon^2(1-\beta)} = \frac{\sigma^2}{\sigma^2(1-0.95)} = 20 \quad (4.2)$$

Відповідний розподіл значень для різних варіантів співвідношень величин дисперсії відгуку моделі та точності вимірювання можна побачити на рисунку 4.7.

ε	$\frac{\sigma^2}{\varepsilon^2(1-0.95)}$	$\frac{\sigma^2 * 1.96^2}{\varepsilon^2}$
σ	20	4
$\sigma/2$	80	15
$\sigma/4$	320	61
$\sigma/6$	720	138
$\sigma/8$	1280	246
$\sigma/10$	2000	384

Рисунок 4.7 – Оцінка кількості прогонів за нерівністю Чебишева та за центральною граничною теоремою [2]

4.3. Проведення регресійного аналізу

Метою регресійного аналізу є кількісна оцінка впливу факторів. Регресійний аналіз дозволяє визначити, який з факторів має найбільший чи найменший вплив на результат. Також він дає змогу встановити, яким чином необхідно змінювати значення факторів, щоб досягти збільшення або зменшення значення відгуку моделі на задану величину [2].

Метою проведення регресійного аналізу у контексті даної роботи є оцінка залежності частоти знищених пакетів від вхідних параметрів системи. Формули розрахунку значень вхідних змінних наведені у формулах 4.3 – 4.5 [2].

$$x_i = \frac{X_i - X_{i0}}{\Delta i} \quad (4.3)$$

$$x_{i0} = \frac{X_{i\,max} + X_{i\,min}}{2} \quad (4.4)$$

$$\Delta i = \frac{X_{i\,max} - X_{i\,min}}{2} \quad (4.5)$$

де x_i – вхідна змінна, яка змінюється у межах ($X_{i\,min}$, $X_{i\,max}$).

Обчислення значення вхідних змінних наведено у таблиці 4.1. Змінна x_1 відповідає за тривалість обробки каналами у звичайному режимі роботи, x_2 позначає тривалість обробки у резервному режимі, x_3 – максимально допустима тривалість життя пакету.

Змінна	Xi min	Xi max	Xi0	Δi
x1	5	10	7.5	2.5
x2	1	4	2.5	1.5
x3	10	20	15	5

Рисунок 4.8 – Обчислення нормалізованих значень вхідних змінних

Далі буде складено матрицю планування для двох факторів, де кожен рядок відповідає окремому експерименту. Матриця планування, за якою буде проводитися серія експериментів, для двох факторів змінних наведена у таблиці 4.1. За значення величини y – було прийнято частоту знищення пакетів.

Таблиця 4.1 – Матриця планування експерименту

2^3	x_0	x_1	x_2	x_3	x_1x_2	x_1x_3	x_2x_3	$x_1x_2x_3$	y
1	+	+	+	+	+	+	+	+	y_1
2	+	-	+	+	-	-	+	-	y_2
3	+	+	-	+	-	+	-	-	y_3
4	+	-	-	+	+	-	-	+	y_4
5	+	+	+	-	+	-	-	-	y_5

Продовження таблиці 4.2

6	+	-	+	-	-	+	-	+	y_6
7	+	+	-	-	-	-	+	+	y_7
8	+	-	-	-	+	+	+	-	y_8

Загальне рівняння регресії можна знайти у формулі 4.6 [2].

$$y = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + b_4x_1x_2 + b_5x_1x_3 + b_6x_2x_3 + b_7x_1x_2x_3 \quad (4.6)$$

Результат проведення експериментів можна побачити на рисунку 4.9.

Default	Reserve	Lifetime	Succeeded	Failed	Loss
10	4	20	11656	4992	0.29987
5	4	20	15448	1202	0.07219
10	1	20	11656	4993	0.29989
5	1	20	15471	1198	0.07191
10	4	10	11672	5002	0.29999
5	4	10	11664	4997	0.29991
10	1	10	11671	5001	0.29993
5	1	10	11657	4993	0.29987

Рисунок 4.9 – Результати проведення експериментів

Загальний вираз обчислення коефіцієнтів b наведено у формулі 4.7 [2].

$$b_i = \frac{\sum_{i=1}^{n} y_i x_{ik}}{N}, \; k = 0, \ldots, N-1 \quad (4.7)$$

Знайдені значення коефіцієнтів b_i можна побачити на рисунку 4.10.

x0	b0	0.242945
x1	b1	0.056975
x2	b2	0.000045
x3	b3	-0.05698
x1x2	b4	-0.000035
x1x3	b5	0.05694
x2x3	b6	0.00002
x1x2x3	b7	-0.00004

Рисунок 4.10 – Отримані значення коефіцієнтів рівняння b_i

Верифікацію значень b_i можна провести шляхом підставки [-1; 1] у отримане рівняння регресії. Валідація рівняння наведена на рисунку 4.11, де тестове значення для вхідних аргументів x_1, x_2 та x_3 дорівнює значенню y_1.

x1	1
x2	1
x3	1
TEST	0.29987

Рисунок 4.11 – Верифікація рівняння регресії

Статистична обробка результатів факторних експериментів розпочинається з оцінки відтворюваності експерименту за критерієм Кохрана. Для цього був підготовлений датасет на 20 прогонів моделі з відповідними параметрами експерименту. Рівняння обчислення дисперсії наведено у формулі 4.8 [2].

$$D_i = \frac{\sum_{i=1}^{20} (y_{ij} - y_j)^2}{19} \tag{4.8}$$

Отримані вихідні значення моделювання зображені на рисунку 4.12.

0	1	10	11	12	13	14	15	16	17	18	19	2	3	4	5	6	7	8	9
0.29995	0.29993	0.29994	0.29992	0.29991	0.29994	0.29993	0.29993	0.29992	0.29993	0.29995	0.29991	0.29993	0.29996	0.29994	0.29996	0.29991	0.29991	0.29993	0.29991
0.0675	0.06581	0.07536	0.0722	0.07081	0.07322	0.07538	0.07394	0.07652	0.06838	0.0775	0.08051	0.06788	0.07493	0.08263	0.07805	0.07201	0.06405	0.07736	0.07744
0.29993	0.29993	0.29994	0.29993	0.29992	0.29995	0.29993	0.29995	0.29996	0.29995	0.29996	0.29996	0.29992	0.29994	0.29993	0.29993	0.29995	0.29993	0.29997	0.29994
0.08623	0.07957	0.08438	0.07092	0.08023	0.08905	0.06793	0.07097	0.07351	0.06963	0.073	0.0764	0.07838	0.06835	0.06161	0.08315	0.07743	0.08106	0.0706	0.06789
0.29996	0.3	0.29998	0.30004	0.29998	0.29998	0.29997	0.29997	0.29995	0.3001	0.29995	0.29997	0.29997	0.3	0.29998	0.29998	0.30003	0.29998	0.29999	0.29997
0.29992	0.29998	0.29999	0.29995	0.29996	0.2999	0.30015	0.29997	0.30004	0.29995	0.29996	0.29996	0.29992	0.2999	0.29992	0.29995	0.29993	0.29994	0.29997	0.30006
0.29994	0.29998	0.29997	0.29998	0.29999	0.29997	0.29996	0.29995	0.29996	0.29998	0.29998	0.29998	0.29998	0.29998	0.29998	0.29998	0.29996	0.29997	0.29996	0.29996
0.29992	0.29995	0.29994	0.29994	0.29994	0.29996	0.29992	0.29992	0.29992	0.29992	0.29997	0.29996	0.29994	0.29991	0.29994	0.29995	0.29994	0.29993	0.29994	0.29993

Рисунок 4.12 – Агреговані результати запуску прогонів

Код обчислення середнього значення та дисперсії наведено наведено на рисунку 4.13.

```
def calculate_dispersion(row: pd.Series):
    row_sum: float = 0
    values_count: int = len(row)
    sum_squared_diff: float = 0.0

    for value in row:
        row_sum += value

    row_mean: float = row_sum / values_count

    for value in row:
        sum_squared_diff += (value - row_mean) ** 2

    return sum_squared_diff / (values_count - 1)
```

Рисунок 4.13 – Обчислення значення дисперсії для кожного рядку даних

Критичне значення Кохрена було обрано для 20 прогонів з набором параметрів рівним 3 з таблиці, яка наведена на рисунку 4.14.

	Upper 0.05 significance level critical values												
df	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*17*	*37*	*145*
2	.9985	.9750	.9392	.9057	.8772	.8534	.8332	.8159	.8010	.7880	.7341	.6602	.5813
3	.9669	.8709	.7977	.7457	.7071	.6771	.6530	.6333	.6167	.6025	.5466	.4748	.4031
4	.9065	.7679	.6841	.6287	.5895	.5598	.5365	.5175	.5017	.4884	.4366	.3720	.3093
5	.8412	.6838	.5981	.5441	.5065	.4783	.4564	.4387	.4241	.4118	.3645	.3066	.2513
6	.7808	.6161	.5321	.4803	.4447	.4184	.3980	.3817	.3682	.3568	.3135	.2612	.2119
7	.7271	.5612	.4800	.4307	.3974	.3726	.3535	.3384	.3259	.3154	.2756	.2278	.1833
8	.6798	.5157	.4377	.3910	.3595	.3362	.3185	.3043	.2926	.2862	.2462	.2022	.1616
9	.6385	.4775	.4027	.3584	.3286	.3067	.2901	.2768	.2659	.2568	.2226	.1820	.1446
10	.6020	.4450	.3733	.3311	.3029	.2823	.2666	.2541	.2439	.2353	.2032	.1655	.1308
12	.5410	.3924	.3264	.2880	.2624	.2439	.2299	.2187	.2098	.2020	.1737	.1403	.1100
15	.4709	.3346	.2758	.2419	.2195	.2034	.1911	.1815	.1736	.1671	.1429	.1144	.0889
20	.3894	.2705	.2205	.1921	.1735	.1602	.1501	.1422	.1357	.1303	.1108	.0879	.0675
24	.3434	.2354	.1907	.1656	.1493	.1374	.1286	.1216	.1160	.1113	.0942	.0743	.0567
30	.2929	.1980	.1593	.1377	.1237	.1137	.1061	.1002	.0958	.0921	.0771	.0604	.0457
40	.2370	.1576	.1259	.1082	.0968	.0887	.0827	.0780	.0745	.0713	.0595	.0462	.0347
60	.1737	.1131	.0895	.0765	.0682	.0623	.0583	.0552	.0520	.0497	.0411	.0316	.0234
120	.0998	.0632	.0495	.0419	.0371	.0337	.0312	.0292	.0279	.0266	.0218	.0165	.0120
∞	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000	.0000

Рисунок 4.14 – Критичні значення Кохрена для ступеня значущості 0.05 [4]

Код перевірки отриманого значення Кохрена та порівняння до критичного наведено на рисунку 4.15.

```python
max_dispersion: float = df['Dispersion'].max()
sum_dispersion: float = df['Dispersion'].sum()
cochran_value: float = max_dispersion / sum_dispersion

print(f"dispersion: {max_dispersion:.5f}")

threshold: float = 0.24

if cochran_value ⩽ threshold:
    print("Cochran value is within acceptable limits.")
else:
    print("Cochran value exceeds acceptable limits.")
```

Рисунок 4.15 – Перевірка значення Кохрена

Вивід результату перевірки критичного значення наведено на рисунку 4.16, відповідно можна зробити висновок про відтворюваність експерименту.

```
dispersion: 0.00397
Cochran value is within acceptable limits.
```

Рисунок 4.16 – Підтвердження перевірки критичного значення Кохрена

Отримані коефіцієнти рівняння регресії наведені у формулі 4.9.

$$y^{\text{рег}} \approx 0.24 + 0.057x_1 + 4.5E - 5x_2 - 0.057x_3 + 3.5E - 5x_1x_2 + \\ + 0.057x_1x_3 + 2E - 5x_2x_3 - 4E - 5x_1x_2x_3 \quad (4.9)$$

Для тесту Ст'юдента був обраний ступінь значущості $a = 0.005$ та число ступенів свободи $N(p - 1) = 20 - 1 = 19$, був обраний $t_{\text{кр}} = 2.09$ [5].

Перевірка значень відбувається за нерівністю загального вигляду, яке наведено у формулі 4.10.

$$t_i = b_i \cdot \sqrt{\frac{8 \cdot 20}{D}} > 2.09 \quad (4.10)$$

Результати обчислень наведені на рисунку 4.17.

t0	15.04873274	TRUE
t1	3.529200221	TRUE
t2	0.002787433259	FALSE
t3	-3.529509935	FALSE
t4	-0.002168003646	FALSE
t5	3.527032217	TRUE
t6	0.001238859226	FALSE
t7	-0.002477718452	FALSE

Рисунок 1.17 – Отримані результати перевірки за критерієм Ст'юдента

Спрощене рівняння регресії наведено у формулі 4.11.

$$y^{\text{рег}} = 0.24 + 0.056975x_1 + 0.05694x_1x_3 \quad (4.11)$$

Оскільки були відкинуті деякі члени рівняння, то необхідно провести оцінку адекватності побудованої моделі за критерієм Фішера. Значення, отримані за спрощеним рівнянням регресії наведено на рисунку 1.18. Обчислення значення адекватності дисперсії адекватності наведено у формулі 4.12 [2].

0.299885
0.242945
0.299885
0.242945
0.29992
0.242945
0.29992
0.242945

Рисунок 1.18 – Значення, які отримані за спрощеним рівнянням регресії

$$D_{\text{ад}} = \frac{\sum_{i=1}^{8}(y_i - y_i^{\text{per}})^2}{N-L} = \frac{0.0648957}{8-3} \approx 3.27 \qquad (4.12)$$

Критичне значення критерію Фішера було обрано для таких значень кількості ступенів вільності як $8 \cdot (3-1) = 16$ та $3-1 = 2$, саме значення 19.43. Таблиця критичних значень Фішера наведена на рисунку 1.19 [6].

Рівень значимості $\alpha = 0,05$								
$f_2 \backslash f_1$	4	7	10	16	24	40	100	∞
1	225,0	237,0	242,0	246,0	249,0	251,0	253,0	254,0
2	19,25	19,36	19,39	19,43	19,45	19,47	19,49	19,50
3	9,12	8,88	8,78	8,69	8,64	8,60	8,56	8,53
4	6,39	6,09	5,96	5,84	5,77	5,71	5,66	5,63
5	5,19	4,88	4,74	4,60	4,53	4,46	4,40	4,36
6	4,53	4,21	4,06	3,92	3,84	3,77	3,71	3,67
7	4,12	3,79	3,63	3,49	3,41	3,34	3,28	3,23
8	3,84	3,50	3,34	3,20	3,12	3,05	2,98	2,93
9	3,63	3,29	3,13	2,98	2,90	2,82	2,76	2,71
10	3,48	3,14	2,97	2,82	2,74	2,67	2,59	2,54
12	3,26	2,92	2,76	2,60	2,50	2,42	2,35	2,30
14	3,11	2,77	2,60	2,44	2,35	2,27	2,19	2,13
16	3,01	2,66	2,49	2,33	2,24	2,16	2,07	2,01
18	2,93	2,58	2,41	2,25	2,15	2,07	1,98	1,92

Рисунок 1.19 – Критичні значення Фішера для ступеня значущості 0.05 [6]

Оскільки отримане значення критерію Фішера у 3.27 менше табличного значення у 19.43, то, то отримане спрощене рівняння регресії визнається адекватним результатам факторного експерименту.

4.4. Проміжні висновки до розділу

У даному розділі виконано оцінку перехідного періоду моделі, визначено оптимальну кількість прогонів для точних експериментів та проведено регресійний аналіз. Аналіз перехідних графіків дозволив встановити тривалість симуляції в 100000 умовних одиниць для мінімізації впливу початкових ефектів. За допомогою нерівності Чебишева визначено необхідну кількість прогонів, що забезпечують надійність результатів. Побудовано рівняння регресії, проведено його верифікацію та оцінку адекватності за критерієм Фішера, що підтвердило його відповідність експериментальним даним. У результаті отримано спрощене рівняння регресії, яке дозволяє ефективно оцінювати вплив вхідних факторів на віжгук моделі.

5 ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ СИСТЕМИ

5.1. Загальний огляд результатів моделювання

У результаті проведення моделювання були отримані наступні відгуки:

- Частота знищення пакетів не перевищує 0.3 при стандартних факторах системи;
- Частота підключення резервних ресурсів залежить від частоти знищення пакетів, чим вище частота знищення, тим довше працює допоміжні потужності без упину, й відповідно менша частота їх підключення.

5.2. Огляд процесу проведення експериментів

На початковому етапі були побудовані графіки, що відображають залежність частоти знищення пакетів, частоти підключення резервних ресурсів та середньої тривалості життя пакету від часу моделювання. Це дозволило визначити перехідний період моделі, який є важливим для забезпечення стабільної роботи системи, оскільки для статистичних спостережень враховуються лише дані, що відповідають сталому стану системи [2].

Перехідний період визначається як час, необхідний для досягнення системою сталого стану. Його тривалість встановлюється експериментально шляхом побудови графіків залежності відгуку моделі від часу. Аналіз показав, що перехідний період завершується на 90,000 одиниць часу для всіх параметрів. Відтак статистика обнуляється з цього моменту, а тривалість моделювання становить 100,000 одиниць часу [2].

Для аналізу було обрано метод регресії, який дозволяє кількісно оцінити вплив факторів та визначити, які з них найбільше чи найменше впливають на відгук моделі. Цей метод також допомагає встановити, як змінювати значення факторів для досягнення потрібного результату. У рамках роботи регресійний аналіз застосовано для оцінки залежності частоти знищення пакетів від вхідних параметрів системи [2].

5.3. Огляд результатів проведення експериментів

Провівши всі розрахунки, склавши рівняння системи можна зробити наступні висновки:

- Найбільший вплив на частоту знищених пакетів має час обробки основним каналом обробки та максимальна тривалість життя пакету;
- Логічним також є той факт, що зі збільшенням фактору максимальної тривалості життя пакету, зменшується й частота їх знищення у системі;
- Параметр тривалості обробки резервним каналом має низький вплив на відгук моделі. Можна припустити, що це є правдою лише у контексті розглянутих межах вхідних параметрів, які були обрані для проведеного експерименту і якщо взяти більший проміжок, то картина буде іншою.

5.4. Формулювання пропозицій по покращенню роботи системи

Розглянувши висновки про роботу системи, можна надати наступні рекомендації по зменшенню частоти знищених пакетів:

- Переглянути політику максимальної тривалості життя пакету у системі в бік збільшення цього значення, що дозволить зменшити завантаженість каналів обробки й частоту залучення додаткових ресурсів;
- Якщо тривалість життя пакету має залишатися сталим значенням, то потрібно задуматися про пришвидшення часу обробки стандартним каналом, що допоможе економити ресурси, які необхідні для підключення, відключення та роботи каналів у резервному режимі.

5.5. Проміжні висновки до розділу

Моделювання показало, що ключовими факторами впливу на частоту знищення пакетів є час обробки основним каналом і максимальна тривалість життя пакету. Збільшення тривалості життя пакету зменшує частоту їх знищення, а прискорення обробки основним каналом дозволяє знизити навантаження на систему.

ВИСНОВКИ

Під час виконання курсової роботи було розроблено концептуальну модель системи, яка включає аналіз структури, логіки роботи та механізму регулювання, спрямованого на оптимізацію якості передачі мовлених пакетів. Було детально описано процес обробки пакетів, динаміку роботи системи та використання додаткових ресурсів, що дозволило сформулювати комплексне уявлення про досліджувану систему. Особливу увагу приділено визначенню вхідних та вихідних змінних, що стануть основою для подальших експериментальних досліджень.

У межах роботи було використано формалізм мереж масового обслуговування, що дало змогу розглянути систему як сукупність компонентів із визначеною структурою та динамікою. На основі алгебраїчних підходів було отримано математичні моделі для визначення вихідних змінних, що використовувалися під час експериментів.

Алгоритм імітації, розроблений у межах дискретно-подійного підходу, показав високу ефективність і точність. Механізм управління подіями та часовими інтервалами забезпечує коректне функціонування моделі, а результати її верифікації підтвердили відповідність теоретичним та аналітичним прогнозам. Дослідження виявило, що модель адекватно реагує на зміну вхідних параметрів, а її логіка забезпечує правильність переходів між станами.

У рамках експериментального дослідження моделі було проведено регресійний аналіз частоти знищення пакетів, побудовано рівняння регресії та оцінено його коректність та адекватність. Експерименти показали, що основними факторами, які впливають на ефективність роботи системи, є тривалість обробки пакетів основним каналом та максимальна тривалість їхнього життя. Оптимізація цих параметрів дозволяє суттєво знизити частоту знищення пакетів і покращити використання системних ресурсів. Зокрема, збільшення тривалості життя пакетів зменшує кількість втрат, тоді як прискорення обробки основним каналом знижує навантаження на систему.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Digital audio compression algorithms and implementations, Jon Rowlands [Електронний ресурс] – Режим доступу: https://www.winlab.rutgers.edu/~andrej/research/papers/ac3.pdf
2. Моделювання систем, навчальний посібник, І. В. Стеценко [Електронний ресурс] – Режим доступу: https://do.ipo.kpi.ua/pluginfile.php/112577/mod_resource/content/1/%D0%9C%D0%BE%D0%B4%D0%B5%D0%BB%D1%8E%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%9D%D0%B0%D0%B2%D1%87%D0%9F%D0%BE%D1%81%D1%96%D0%B1%D0%BD%D0%B8%D0%BA_2011.pdf
3. Моделювання систем, курсова робота, І. В. Стеценко, Дифучина О. Ю., Дифучин А. Ю. [Електронний ресурс] – Режим доступу: https://ela.kpi.ua/items/907b5d1f-ca48-4165-a67e-e790ac6fbec5
4. Biostatistics: A Methodology for the Health Sciences (2nd Edition), IGerald van Belle, Lloyd Fisher, Patrick J. Heagerty, Thomas Lumley [Електронний ресурс] – Режим доступу: https://faculty.washington.edu/heagerty/Books/Biostatistics/TABLES/Cochran.pdf
5. Application of Student's t-test, Analysis of Variance and Covariance, Prabhaker Mishra, Uttam Singh, Chandra M Pandey, Priyadarshni Mishra, Gaurav Pandey [Електронний ресурс] – Режим доступу: https://pmc.ncbi.nlm.nih.gov/articles/PMC6813708/
6. Upper Critical Values of the F Distribution, US National Institute of Standards and Technology [Електронний ресурс] – Режим доступу: https://www.itl.nist.gov/div898/handbook/eda/section3/eda3673.htm

ДОДАТКИ

Посилання на GitHub-репозиторій: https://github.com/NgeNXQ/System-Modelling

Додаток А. Код модулів системи масового обслуговування

Клас SimpleScheme

```
using System;
using System.Collections.Generic;
using Coursework.Framework.Core.Services;
using Coursework.Framework.Components.Tasks.Common;
using Coursework.Framework.Components.Modules.Common;
using Coursework.Framework.Components.Blueprints.Schemes.Common;
using Coursework.Framework.Components.Blueprints.Transitions.Common;

namespace
Coursework.Framework.Components.Blueprints.Schemes.Concrete;

internal sealed class SimpleScheme : Scheme
{
   private readonly ICollection<Transition> transitions;

     internal SimpleScheme(string identifier, params Transition[] transitions) :
base(identifier)
     {
       this.transitions = new LinkedList<Transition>();

       foreach (Transition transition in transitions)
                                   this.transitions.Add(transition ?? throw new
ArgumentNullException($"{nameof(transition)} cannot be null."));
     }
```

```
        internal int FailedTasksCount { get; private set; }

        internal int UserFailedTasksCount { get; private set; }

        internal float TasksLifetimeSum { get; private set; }

        internal int SucceededTasksCount { get; private set; }

        internal int UserSucceededTasksCount { get; private set; }

            internal float TaskLifetimeMean => this.TasksLifetimeSum / this.ProcessedTasksCount;

            internal int ProcessedTasksCount => this.SucceededTasksCount + this.FailedTasksCount;

            internal int UserProcessedTasksCount => this.UserSucceededTasksCount + this.UserFailedTasksCount;

        internal override sealed Module? GetNextModule(DummyTask task)
        {
            if (task == null)
                throw new ArgumentNullException($"{nameof(task)} cannot be null.");

            Module? nextModule = null;

            foreach (Transition transition in this.transitions)
            {
                if (transition.CheckIsRunnable(task))
```

```
            {
                if (nextModule != null)
                    throw new InvalidOperationException("Invalid scheme. More than
1 runnable transition has been detected.");

                nextModule = transition.Destination;
                transition.UpdateStatistics(TransitionStatus.Active, task);
                continue;
            }

            transition.UpdateStatistics(TransitionStatus.Inactive, task);
        }

        if (nextModule == null)
        {
            ++this.FailedTasksCount;
            ++this.UserFailedTasksCount;
        }
        else
        {
            ++this.SucceededTasksCount;
            ++this.UserSucceededTasksCount;
        }

        this.TasksLifetimeSum += task.Lifetime;

        return nextModule;
    }

    internal void Attach(Transition transition)
```

```
        {
                                this.transitions.Add(transition ?? throw new
ArgumentNullException($"{nameof(transition)} cannot be null."));
        }

        public override sealed void PrintStatistics()
        {
                string formattedLogMessage = $"[{base.Identifier}]: Successes:
{this.SucceededTasksCount}; Failures: {this.FailedTasksCount}; Tasks' lifetime
(mean): {this.TaskLifetimeMean}";
                SystemLoggerService.Instance.AppendLogEntry(this, "REPORT",
"STATS", formattedLogMessage);

            foreach (Transition transition in this.transitions)
                transition.PrintStatistics();
        }

        public override sealed void ResetStatistics()
        {
            this.UserFailedTasksCount = 0;
            this.UserSucceededTasksCount = 0;
        }

        public override sealed void UpdateStatistics(float deltaTime)
        {
        }
    }
```

Клас DisposeTransition

```
    using System;
```

```
using Coursework.Framework.Core.Services;
using Coursework.Framework.Components.Tasks.Common;
using Coursework.Framework.Components.Modules.Concrete;
using Coursework.Framework.Components.Blueprints.Transitions.Common;

namespace Coursework.Framework.Components.Blueprints.Transitions.Concrete;

internal sealed class DisposeTransition : Transition
{
    internal DisposeTransition(string identifier, DisposeModule dispose) : base(identifier, dispose)
    {
    }

    internal int ForwardedTasksCount { get; private set; }

    internal override sealed bool CheckIsRunnable(DummyTask task)
    {
        return true;
    }

    internal sealed override void UpdateStatistics(TransitionStatus status, DummyTask task)
    {
        if (task == null)
            throw new ArgumentNullException($"{nameof(task)} cannot be null.");

        if (status == TransitionStatus.Active)
```

```
            {
                ++this.ForwardedTasksCount;
                        SystemLoggerService.Instance.AppendLogEntry(this,  "LOG",
"TRACE", $"[{base.Identifier}] resends task #{task.Id} (lifetime: {task.Lifetime})");
            }

        }

        public override sealed void PrintStatistics()
        {
            SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "STATS",
$"[{base.Identifier}]: Tasks (forwarded) {this.ForwardedTasksCount}");
        }

        public override sealed void ResetStatistics()
        {
            this.ForwardedTasksCount = 0;
        }

        public override sealed void UpdateStatistics(float deltaTime)
        {
        }
    }
```

Клас ProcessorBlockingTransition

```
    using System;
    using Coursework.Framework.Core.Services;
    using Coursework.Framework.Core.Controllers;
    using Coursework.Framework.Components.Tasks.Common;
    using Coursework.Framework.Components.Modules.Concrete;
```

```
using Coursework.Framework.Components.Blueprints.Transitions.Common;

namespace
Coursework.Framework.Components.Blueprints.Transitions.Concrete;

internal sealed class ProcessorBlockingTransition : Transition
{
   private readonly Func<DummyTask, bool> flowPredicateHandler;

   private float activationTime;
   private TransitionStatus previousBlockingStatus;

      internal ProcessorBlockingTransition(string identifier, ProcessorModule
processor, Func<DummyTask, bool> flowPredicateHandler) : base(identifier,
processor)
      {
         this.previousBlockingStatus = TransitionStatus.None;
               this.flowPredicateHandler = flowPredicateHandler ?? throw new
ArgumentNullException($"{nameof(flowPredicateHandler)} cannot be null.");
      }

      internal float TimeUsageTotal { get; private set; }

      internal int ActivationsCount { get; private set; }

      internal int DeactivationsCount { get; private set; }

      internal sealed override void UpdateStatistics(TransitionStatus status,
DummyTask task)
      {
```

```
            if (status != this.previousBlockingStatus)
            {
                if (status == TransitionStatus.Active)
                {
                    ++this.ActivationsCount;
                    this.activationTime = SystemModelController.Instance.TimeCurrent;
                }
                else if (status == TransitionStatus.Inactive)
                {
                    ++this.DeactivationsCount;
                }
            }

            if (this.previousBlockingStatus != TransitionStatus.Inactive)
            {
                 this.TimeUsageTotal += SystemModelController.Instance.TimeCurrent
- this.activationTime;
                this.activationTime = SystemModelController.Instance.TimeCurrent;
            }

            this.previousBlockingStatus = status;
        }

        internal override sealed bool CheckIsRunnable(DummyTask task)
        {
                    return !this.flowPredicateHandler.Invoke(task ?? throw new
ArgumentNullException($"{nameof(task)} cannot be null."));
        }

        public override sealed void PrintStatistics()
```

```
        {
                string formattedLogMessage = $"[{base.Identifier}]: Activations: {this.ActivationsCount}; Deactivations: {this.DeactivationsCount}; Usage time: {this.TimeUsageTotal}";
                SystemLoggerService.Instance.AppendLogEntry(this, "REPORT", "STATS", formattedLogMessage);
        }

        public override sealed void ResetStatistics()
        {
            this.TimeUsageTotal = 0;
            this.ActivationsCount = 0;
            this.DeactivationsCount = 0;
        }

        public override sealed void UpdateStatistics(float deltaTime)
        {
        }
    }
```

Клас ProcessorSimpleTransition

```
    using System;
    using Coursework.Framework.Core.Services;
    using Coursework.Framework.Components.Tasks.Common;
    using Coursework.Framework.Components.Modules.Concrete;
    using Coursework.Framework.Components.Blueprints.Transitions.Common;

    namespace Coursework.Framework.Components.Blueprints.Transitions.Concrete;
```

```
internal sealed class ProcessorSimpleTransition : Transition
{
    internal ProcessorSimpleTransition(string identifier, ProcessorModule processor) : base(identifier, processor)
    {
    }

    internal int ForwardedTasksCount { get; private set; }

    internal override sealed bool CheckIsRunnable(DummyTask task)
    {
      return true;
    }

    internal sealed override void UpdateStatistics(TransitionStatus status, DummyTask task)
    {
      if (task == null)
            throw new ArgumentNullException($"{nameof(task)} cannot be null.");

      if (status == TransitionStatus.Active)
      {
        ++this.ForwardedTasksCount;
            SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "TRACE", $"[{base.Identifier}] resends task #{task.Id} (lifetime: {task.Lifetime})");
      }
    }

    public override sealed void PrintStatistics()
```

```
        {
            SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "STATS",
$"[{base.Identifier}]: Tasks (forwarded) {this.ForwardedTasksCount}");
        }

        public override sealed void ResetStatistics()
        {
            this.ForwardedTasksCount = 0;
        }

        public override sealed void UpdateStatistics(float deltaTime)
        {
        }
    }
```

Клас CreateModule

```
    using System;
    using System.Collections.Generic;
    using Coursework.Framework.Core.Services;
    using Coursework.Framework.Components.Tasks.Common;
    using Coursework.Framework.Components.Workers.Common;
    using Coursework.Framework.Components.Modules.Common;
    using Coursework.Framework.Components.Blueprints.Schemes.Common;
    using Coursework.Framework.Components.Tasks.Utilities.Factories.Common;

    namespace Coursework.Framework.Components.Modules.Concrete;

    internal sealed class CreateModule : Module
    {
        private readonly Scheme scheme;
```

```
        private readonly IWorker worker;
        private readonly IDummyTaskFactory dummyTaskFactory;

         internal CreateModule(string identifier, Scheme scheme, IWorker worker,
IDummyTaskFactory dummyTaskFactory) : base(identifier)
        {
                                    this.scheme = scheme ?? throw new
ArgumentNullException($"{nameof(scheme)} cannot be null.");
                                     this.worker = worker ?? throw new
ArgumentNullException($"{nameof(worker)} cannot be null.");
                    this.dummyTaskFactory = dummyTaskFactory ?? throw new
ArgumentNullException($"{nameof(dummyTaskFactory)} cannot be null.");

           this.MoveTimeline(this.worker.Delay);
        }

        internal int CreatedTasksCount { get; private set; }

        internal override sealed void CompleteTask()
        {
           ++this.CreatedTasksCount;

                                                DummyTask newTask =
this.dummyTaskFactory.CreateDummyTask(this.TimeCurrent);
           Module nextModule = this.scheme.GetNextModule(newTask)!;
           this.MoveTimeline(this.worker.Delay);
           nextModule.AcceptTask(newTask);

            SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "TRACE",
$"[{base.Identifier}] sends [{newTask}] to the [{nextModule?.Identifier}]");
```

```
        }

        internal override sealed void AcceptTask(DummyTask task)
        {
                        throw new InvalidOperationException($"{base.GetType()}
{base.Identifier} is not able to accept tasks.");
        }

        internal override sealed void UpdateTimeline(float currentTime)
        {
            base.TimeCurrent = currentTime;
        }

        private protected override sealed void MoveTimeline(float deltaTime)
        {
            base.TimeNext = this.TimeCurrent + deltaTime;
        }

        public override sealed void PrintStatistics()
        {
                SystemLoggerService.Instance.AppendLogEntry(this, "REPORT",
"STATS", $"[{base.Identifier}]: Tasks: {this.CreatedTasksCount}");
            this.scheme.PrintStatistics();
        }

        public override sealed void ResetStatistics()
        {
            this.CreatedTasksCount = 0;
        }
```

```
    public override sealed void UpdateStatistics(float deltaTime)
    {
    }
}
```

Клас DisposeModule

```
using System;
using Coursework.Framework.Core.Services;
using Coursework.Framework.Components.Tasks.Common;
using Coursework.Framework.Components.Modules.Common;

namespace Coursework.Framework.Components.Modules.Concrete;

internal sealed class DisposeModule : Module
{
    internal DisposeModule(string identifier) : base(identifier)
    {
    }

    internal int DisposedTasksCount { get; private set; }

    internal override sealed void CompleteTask()
    {
                    throw new InvalidOperationException($"{base.Identifier}
({this.GetType()}) is not able to complete tasks.");
    }

    internal override sealed void AcceptTask(DummyTask task)
    {
      ++this.DisposedTasksCount;
```

```
            SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "TRACE",
$"[{base.Identifier}] disposes [{task}]");
        }

        internal override sealed void UpdateTimeline(float timeCurrent)
        {
            base.TimeCurrent = timeCurrent;
        }

        private protected override sealed void MoveTimeline(float deltaTime)
        {
            base.TimeNext = this.TimeCurrent + deltaTime;
        }

        public override sealed void PrintStatistics()
        {
                SystemLoggerService.Instance.AppendLogEntry(this,  "REPORT",
"STATS", $"[{base.Identifier}]: Tasks (disposed) {this.DisposedTasksCount}");
        }

        public override sealed void ResetStatistics()
        {
            this.DisposedTasksCount = 0;
        }

        public override sealed void UpdateStatistics(float deltaTime)
        {
        }
    }
```

Клс ProcessorModule

```
using System;
using Coursework.Framework.Core.Services;
using Coursework.Framework.Core.Controllers;
using Coursework.Framework.Components.Tasks.Common;
using Coursework.Framework.Components.Queues.Common;
using Coursework.Framework.Components.Workers.Common;
using Coursework.Framework.Components.Modules.Common;
using Coursework.Framework.Components.Blueprints.Schemes.Common;

namespace Coursework.Framework.Components.Modules.Concrete;

internal sealed class ProcessorModule : Module
{
    private readonly IQueue queue;
    private readonly Scheme scheme;
    private readonly IWorker worker;

    private DummyTask? currentTask;

    private float currentBusyness;
    private float currentQueueMean;

    internal ProcessorModule(string identifier, Scheme scheme, IWorker worker,
IQueue queue) : base(identifier)
    {
                                this.queue = queue ?? throw new
ArgumentNullException($"{nameof(queue)} cannot be null.");
```

```
                this.scheme = scheme ?? throw new ArgumentNullException($"{nameof(scheme)} cannot be null.");
                this.worker = worker ?? throw new ArgumentNullException($"{nameof(worker)} cannot be null.");
        }

        internal bool IsBusy { get; private set; }

        internal int FailedTasksCount { get; private set; }

        internal int SucceededTasksCount { get; private set; }

            internal int ProcessedTasksCount => this.FailedTasksCount + this.SucceededTasksCount;

                internal float Busyness => this.currentBusyness / SystemModelController.Instance.TimeCurrent;

                internal float QueueMean => this.currentQueueMean / SystemModelController.Instance.TimeCurrent;

        internal override sealed void CompleteTask()
        {
            ++this.SucceededTasksCount;
            DummyTask finishedTask = this.currentTask!;

            if (this.queue.IsEmpty)
            {
                this.IsBusy = false;
                this.currentTask = null;
```

```
            this.TimeNext = Single.MaxValue;
        }
        else
        {
            this.MoveTimeline(this.worker.Delay);
            this.currentTask = this.queue.Dequeue();
        }

        Module? nextModule = this.scheme.GetNextModule(finishedTask);
         SystemLoggerService.Instance.AppendLogEntry(this, "LOG", "TRACE",
$"[{base.Identifier}] sends [{finishedTask}] to the [{nextModule?.Identifier}]");
        nextModule?.AcceptTask(finishedTask);
    }

    internal override sealed void AcceptTask(DummyTask task)
    {
        if (task == null)
                throw new ArgumentNullException($"{nameof(task)} cannot be
null.");

        if (this.IsBusy)
        {
            if (!this.queue.IsFull)
                this.queue.Enqueue(task);
            else
                ++this.FailedTasksCount;
        }
        else
        {
            this.IsBusy = true;
```

```
            this.currentTask = task;
            this.MoveTimeline(this.worker.Delay);
        }
    }

    internal override sealed void UpdateTimeline(float currentTime)
    {
        base.TimeCurrent = currentTime;
    }

    private protected override sealed void MoveTimeline(float deltaTime)
    {
        base.TimeNext = this.TimeCurrent + deltaTime;
    }

    public override sealed void PrintStatistics()
    {
                string formattedLogMessage = $"[{base.Identifier}]: Successes: {this.SucceededTasksCount}; Failures: {this.FailedTasksCount}; Queue (mean): {this.QueueMean}; Busyness: {this.Busyness}";
                SystemLoggerService.Instance.AppendLogEntry(this, "REPORT", "STATS", formattedLogMessage);
        this.scheme.PrintStatistics();
    }

    public override sealed void ResetStatistics()
    {
        this.FailedTasksCount = 0;
        this.SucceededTasksCount = 0;
    }
```

```
    public override sealed void UpdateStatistics(float deltaTime)
    {
        this.currentQueueMean += this.queue.Count * deltaTime;
        this.currentBusyness += (this.currentTask != null ? 1 : 0) * deltaTime;
    }
}
```

Клас SimpleQueueFIFO

```
using System;
using System.Collections.Generic;
using Coursework.Framework.Components.Tasks.Common;
using Coursework.Framework.Components.Queues.Common;

namespace Coursework.Framework.Components.Queues.Concrete
{
    internal sealed class SimpleQueueFIFO : IQueue
    {
        private readonly int maxCapacity;
        private readonly Queue<DummyTask> queue;

        internal SimpleQueueFIFO(int maxCapacity)
        {
            if (maxCapacity < 0)
                throw new ArgumentException($"{nameof(maxCapacity)} cannot be
less than 0.");

            this.maxCapacity = maxCapacity;
            this.queue = new Queue<DummyTask>();
        }
```

```
        internal static SimpleQueueFIFO None => new SimpleQueueFIFO(0);

                        internal static SimpleQueueFIFO Infinite => new
SimpleQueueFIFO(Int32.MaxValue);

        public int Count => this.queue.Count;

        public int MaxCapacity => this.maxCapacity;

        public bool IsEmpty => this.queue.Count == 0;

        public bool IsFull => this.queue.Count >= this.maxCapacity;

        public void Clear()
        {
           this.queue.Clear();
        }

        public DummyTask Dequeue()
        {
           if (this.IsEmpty)
              throw new InvalidOperationException("Queue is empty.");

           return this.queue.Dequeue();
        }

        public void Enqueue(DummyTask task)
        {
           if (this.IsFull)
```

```
                throw new InvalidOperationException("Queue is full.");

            if (task == null)
                throw new ArgumentNullException($"{nameof(task)} cannot be
null.");

            this.queue.Enqueue(task);
        }
    }
}
```

Клас SystemModuleController

```
using System;
using System.Linq;
using System.Collections.Generic;
using Coursework.Framework.Common;
using Coursework.Framework.Components.Modules.Common;

namespace Coursework.Framework.Core.Controllers;

internal sealed class SystemModelController : IStatisticsPrinter
{
    private static readonly SystemModelController instance;

    private readonly IList<Module> modules;

    static SystemModelController()
    {
        SystemModelController.instance = new SystemModelController();
    }
```

```
        private SystemModelController()
        {
            this.modules = new List<Module>();
        }

                        internal static SystemModelController Instance =>
SystemModelController.instance;

        internal float TimeNext { get; private set; }

        internal float TimeCurrent { get; private set; }

        internal float IterationsCount { get; private set; }

        internal void Build(IEnumerable<Module> modules)
        {
            this.modules.Clear();

            foreach (Module module in modules)
                                        this.modules.Add(module ?? throw new
ArgumentNullException($"{nameof(module)} cannot be null."));
        }

        internal void RunSimulation(float simulationTime)
        {
            this.TimeNext = 0.0f;
            this.TimeCurrent = 0.0f;
            IList<Module> nextModules;
```

```
            this.TimeNext = this.modules.Min(module => module.TimeNext);

            while (TimeNext < simulationTime)
            {
               ++this.IterationsCount;

               foreach (Module module in this.modules)
                  module.UpdateStatistics(this.TimeNext - this.TimeCurrent);

               this.TimeCurrent = this.TimeNext;

               foreach (Module module in this.modules)
                  module.UpdateTimeline(this.TimeCurrent);

                 nextModules = this.modules.Where(module => module.TimeNext ==
this.TimeCurrent).ToList();

               foreach (Module module in nextModules)
                  module.CompleteTask();

               this.TimeNext = this.modules.Min(module => module.TimeNext);
            }

            this.PrintStatistics();
         }

         internal void RunBenchmarkSimulation(float simulationTime, Action<float>
customLogicHandler)
         {
            this.TimeNext = 0.0f;
```

```
            this.TimeCurrent = 0.0f;
            IList<Module> nextModules;

            this.TimeNext = this.modules.Min(module => module.TimeNext);

            while (TimeNext < simulationTime)
            {
               ++this.IterationsCount;

               foreach (Module module in this.modules)
                  module.UpdateStatistics(this.TimeNext - this.TimeCurrent);

               this.TimeCurrent = this.TimeNext;

               foreach (Module module in this.modules)
                  module.UpdateTimeline(this.TimeCurrent);

                 nextModules = this.modules.Where(module => module.TimeNext ==
this.TimeCurrent).ToList();

               foreach (Module module in nextModules)
                  module.CompleteTask();

               this.TimeNext = this.modules.Min(module => module.TimeNext);

               customLogicHandler.Invoke(this.TimeCurrent);
            }

            this.PrintStatistics();
         }
```

```
    public void PrintStatistics()
    {
        foreach (Module module in this.modules)
            module.PrintStatistics();
    }
}
```

Клас PacketDummyTask

```
using Coursework.Framework.Components.Tasks.Common;

namespace Coursework.Framework.Components.Tasks.Concrete;

internal sealed class PacketDummyTask : DummyTask
{
    internal PacketDummyTask(float timeCreation) : base(timeCreation)
    {
    }

    public sealed override string ToString()
    {
        return $"#{this.Id} ({this.Lifetime})";
    }
}
```

Клас SystemLoggerService

```
using System;
using System.Text;
using System.Collections.Generic;
using Coursework.Framework.Components.Common;
```

```
namespace Coursework.Framework.Core.Services;

internal sealed class SystemLoggerService : IDisposable
{
    private const char LOG_ENTRY_DELIMITER = ' ';
    private const char LOG_ENTRY_TERMINATOR = '\n';

    private const string LOG_ENTRY_TYPE_WRAPPERS = "()";
    private const string LOG_ENTRY_PREFIX_WRAPPERS = "||";

    private const int LOG_ENTRY_WRAPPER_INDEX_START = 0;
    private const int LOG_ENTRY_WRAPPER_INDEX_FINISH = 1;

    private static readonly SystemLoggerService instance;

    private readonly StringBuilder buffer;
    private readonly HashSet<Element> senders;

    private bool isInitialized;

    static SystemLoggerService()
    {
        SystemLoggerService.instance = new SystemLoggerService();
    }

    private SystemLoggerService()
    {
        this.buffer = new StringBuilder();
        this.senders = new HashSet<Element>();
```

```
        }

                    internal static SystemLoggerService Instance =>
SystemLoggerService.instance;

        public void Dispose()
        {
            this.isInitialized = false;
        }

        internal void Initialize()
        {
            this.isInitialized = true;
        }

        internal void RegisterSender(Element element)
        {
            if (element == null)
                throw new ArgumentNullException($"{nameof(element)} cannot be
null.");

            if (this.senders.Contains(element))
                    throw new ArgumentException($"{base.GetType()} is already
subscribed to notifications from the {nameof(element)}.");

            this.senders.Add(element);
        }

        internal void AppendLogEntry(Element sender, string logPrefix, string
logType, string logMessage)
```

```
        {
            if (sender == null)
                throw new ArgumentNullException($"{nameof(sender)} cannot be
null.");

            if (!this.isInitialized)
                return;

            if (!this.senders.Contains(sender))
                return;


this.buffer.Append(SystemLoggerService.LOG_ENTRY_PREFIX_WRAPPERS[Sys
temLoggerService.LOG_ENTRY_WRAPPER_INDEX_START]);
                                this.buffer.Append(logPrefix ?? throw new
ArgumentNullException($"{nameof(logPrefix)} cannot be null."));

this.buffer.Append(SystemLoggerService.LOG_ENTRY_PREFIX_WRAPPERS[Sys
temLoggerService.LOG_ENTRY_WRAPPER_INDEX_FINISH]);

            this.buffer.Append(SystemLoggerService.LOG_ENTRY_DELIMITER);


this.buffer.Append(SystemLoggerService.LOG_ENTRY_TYPE_WRAPPERS[Syste
mLoggerService.LOG_ENTRY_WRAPPER_INDEX_START]);
                                this.buffer.Append(logType ?? throw new
ArgumentNullException($"{nameof(logType)} cannot be null."));

this.buffer.Append(SystemLoggerService.LOG_ENTRY_TYPE_WRAPPERS[Syste
mLoggerService.LOG_ENTRY_WRAPPER_INDEX_FINISH]);
```

```
            this.buffer.Append(SystemLoggerService.LOG_ENTRY_DELIMITER);

                                    this.buffer.Append(logMessage ?? throw new
ArgumentNullException($"{nameof(logMessage)} cannot be null."));


this.buffer.Append(SystemLoggerService.LOG_ENTRY_TERMINATOR);

            Console.Write(this.buffer.ToString());
            this.buffer.Clear();
        }
    }
```

Додаток Б. Код математичних розрахунків

script_deviation.py

```
import numpy as np
import pandas as pd

data = pd.read_csv("coursework/docs/papers/iterations.csv")
values = data['Loss']

mean_value = np.mean(values)
std_deviation = np.std(values)

print(f"{mean_value:.5f} ± {std_deviation:.5f}")
```

script_cochran_summary.py

```
import pandas as pd

df: pd.DataFrame = pd.read_csv("coursework/docs/papers/cochran_default.csv")

row_means: list[float] = list()
dispersions: list[float] = list()

for index, row in df.iterrows():
    row_sum: float = 0
    values_count: int = len(row)
    sum_squared_diff: float = 0.0

    for value in row:
        row_sum += value

    row_means.append(row_sum / values_count)
```

```
    for value in row:
        sum_squared_diff += (value - row_means[-1]) ** 2

    dispersions.append(sum_squared_diff / (values_count - 1))

df['Mean'] = row_means
df['Dispersion'] = dispersions
df.to_csv("coursework/docs/papers/cochran_default.csv", index = False)

max_dispersion: float = df['Dispersion'].max()
sum_dispersion: float = df['Dispersion'].sum()
cochran_value: float = max_dispersion / sum_dispersion

print(f"dispersion: {df['Dispersion'].mean():.5f}")

threshold: float = 0.24

if cochran_value <= threshold:
    print("Cochran value is within acceptable limits.")
else:
    print("Cochran value exceeds acceptable limits.")
```

util_cochran_aggregator.py

```
import os
import pandas as pd

directory = "coursework/docs/papers/"

aggregated_default_df = pd.DataFrame()
```

```
    for filename in os.listdir(directory):
                                    if filename.startswith('cochran_') and
filename[len('cochran_'):].split('.')[0].isdigit() and filename.endswith('.csv'):
        file_path = os.path.join(directory, filename)

        df = pd.read_csv(file_path)

        column_name = filename.split('_')[1].split('.')[0]
        aggregated_default_df[column_name] = df['Loss']
    aggregated_default_df.to_csv("coursework/docs/papers/cochran_default.csv",
index = False)
    aggregated_verbose_df = pd.DataFrame()
    for filename in os.listdir(directory):
    if                  filename.startswith('cochran_')                  and
filename[len('cochran_'):].split('.')[0].isdigit() and filename.endswith('.csv'):
        file_path = os.path.join(directory, filename)

        df = pd.read_csv(file_path)
        column_name = filename.split('_')[1].split('.')[0]

        if aggregated_verbose_df.empty:
            aggregated_verbose_df = df.copy()
                aggregated_verbose_df.rename(columns = {"Loss": column_name},
inplace = True)
        else:
            aggregated_verbose_df[column_name] = df['Loss']

    aggregated_verbose_df.to_csv("coursework/docs/papers/cochran_verbose.csv",
index = False)
```