Зображення, що містить будівля, чорно-білий, панорама, замок

Автоматично згенерований опис

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»

Тема: Система передачі цифрової інформації. Формалізація мережею Петрі

|  |  |
| --- | --- |
| **Керівник**:  Дифучина О. Ю.  «Допущено до захисту»  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2023 р.  Захищено з оцінкою  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Члени комісії:  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | **Виконавець**:  Терешкович М.О  студент групи ІТ-02  залікова книжка № 10  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «26» грудня 2023 р.  Інна СТЕЦЕНКО  Олександра ДИФУЧИНА |

**Київ – 2023**

**ЗАВДАННЯ**

на курсову роботу студента

Терешкович Максим Олександрович

(прізвище, ім'я, по батькові)

1.Тема роботи: Імітаційна модель роботи транзитних каналів для передачі інформації на основі формалізму мережі Петрі

2. Термін здачі студентом закінченої роботи "2" січня 2024 р.

3. Вихідні дані до проекту: Завдання № 16 з Навчального Посібника

4. Зміст розрахунково-пояснювальної записки (перелік питань, що розробляються)

* Розробка концептуальної моделі
* Розробка формалізованої моделі
* Програмна реалізація моделі
* Проведення експериментів
* Інтерпретація результатів експериментів. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Графічного матеріалу не має.

6. Дата видачі завдання "12" вересня 2023 р.

[**РЕФЕРАТ** 4](#_Toc154017306)

[**ВСТУП** 5](#_Toc154017307)

[**ПОСТАНОВКА ЗАДАЧІ** 7](#_Toc154017308)

[**РОЗДІЛ 1. КОНЦЕПТУАЛЬНА МОДЕЛЬ** 8](#_Toc154017309)

[**1.1 Дослідження можливих видів моделювання** 8](#_Toc154017310)

[**1.2 Ціль моделювання** 9](#_Toc154017311)

[**1.3 Концептуальна модель** 10](#_Toc154017312)

[**1.4 Вхідні та вихідні дані, параметри моделі** 11](#_Toc154017313)

[**1.5 Обмеження** 12](#_Toc154017314)

[**1.6 Постановка задачі** 12](#_Toc154017315)

[**РОЗДІЛ 2. ФОРМАЛІЗОВАНА МОДЕЛЬ** 14](#_Toc154017316)

[**2.1 Побудова формалізованої моделі** 14](#_Toc154017317)

[**2.2 Побудова формалізованої моделі** 15](#_Toc154017318)

[**РОЗДІЛ 3 РЕАЛІЗАЦІЯ МОДЕЛІ** 19](#_Toc154017319)

[**РОЗДІЛ 4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ** 25](#_Toc154017320)

[**4.1 Постановка задачі** 25](#_Toc154017321)

[**4.2 Тактичне планування** 25](#_Toc154017322)

[**4.3 Стратегічне планування** 26](#_Toc154017323)

[**ВИСНОВКИ** 31](#_Toc154017324)

[**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ** 32](#_Toc154017325)

[**ДОДАТОК А** 33](#_Toc154017326)

**РЕФЕРАТ**

Курсова робота: 51с., 13 рис., 9 табл., 1 додаток, 2 джерела літератури.

Об'єкт дослідження – транзитний канал передачі даних.

Мета роботи – визначення частоту знищення пакетів і частоту підключення ресурсу. Метод дослідження – імітаційне моделювання роботи каналу.

ІМІТАЦІЙНА МОДЕЛЬ, МОВА МОДЕЛЮВАННЯ, ПЛАНУВАННЯ ЕКСПЕРИМЕНТІВ, ДИСПЕРСІЙНИЙ АНАЛІЗ, КАНАЛ ОБРОБКИ ІНФОРМАЦІЇ, МЕРЕЖА ПЕТРІ

**ВСТУП**

У даній курсовій роботі проводиться аналіз системи передачі цифрової інформації, яка використовується для передачі мовлення в цифровому форматі. Заснована на принципах ефективності та якості передачі, ця система вимагає детального вивчення її функціональності та оптимізації для досягнення оптимальних показників продуктивності.

Система передачі інформації складається з двох транзитних каналів, які використовуються для передачі мовлення у цифровому форматі. Пакети мовлення буферуються перед кожним каналом, дозволяючи оптимальне керування їхнім потоком. Однак, виникає проблема знищення пакетів, що перебувають в системі більше 10 мс, через ймовірне погіршення якості переданого мовлення. Завданням є визначення частоти знищення пакетів та частоти підключення додаткових ресурсів для оптимізації цієї системи.

В рамках вирішення даної проблеми буде використано метод імітаційного моделювання. Метод імітаційного моделювання дозволяє відтворити функціонал системи та вивчити її характеристики, надаючи можливість оптимізувати та покращити ефективність передачі інформації без прямого впливу на реальну систему.

Крім того, для моделювання використовується формалізм мереж Петрі. Мережі Петрі - це математичний та графічний метод для моделювання та аналізу паралельних та розподілених систем. Цей метод був розроблений німецьким математиком та інженером Карлом Адамом Петрі в 1962 році і знайшов широке застосування в інформаційних технологіях, виробництві, бізнес-процесах, а також у інших областях. Мережі Петрі[1], як математичний та графічний метод, нададуть можливість відобразити паралельні процеси та взаємодію компонентів системи. Вони складаються з двох основних елементів: позицій і переходів. Дуги з'єднують ці позиції і переходи, визначаючи потік ресурсів чи подій в системі. Кожен елемент системи має своє власне значення та роль в процесі моделювання. Застосування цього формалізму спростить аналіз та розуміння динаміки системи передачі цифрової інформації, а також допоможе у виявленні оптимальних стратегій в умовах змінних чинників та обмежень якості передачі мовлення.

**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.

**РОЗДІЛ 1. КОНЦЕПТУАЛЬНА МОДЕЛЬ**

## **1.1 Дослідження можливих видів моделювання**

Для аналізу характеристик функціонування будь-якої системи, важливо розробити математичну модель, яка встановлює відповідність між реальною системою та математичним об'єктом, відомим як математична модель.

Математичне моделювання процесів функціонування системи може мати різні методи, включаючи:

* Аналітичне;
* Імітаційне;
* Комбіноване.

Аналітичне моделювання характеризується записом процесів функціонування елементів системи у вигляді функціональних залежностей, таких як алгебраїчні, диференціальні або інтегральні рівняння. Використання аналітичних моделей включає аналіз систем засобами мереж масового обслуговування, вивчення властивостей систем, описаних мережами Петрі, та застосування методів самоорганізації моделей. Аналітичне моделювання дозволяє аналізувати розв'язки математичними методами, але може спрощувати реальні ситуації для отримання аналітичних розв'язків.

Імітаційне моделювання, натомість, використовується для відтворення процесів функціонування системи за допомогою комп'ютерного алгоритму. Цей метод дозволяє враховувати дискретні елементи, нелінійні характеристики та випадкові фактори, що може бути важко врахувати при аналітичних дослідженнях. Імітаційні моделі працюють як система, дозволяючи отримувати інформацію про властивості реальної системи через багаторазові прогони. Основна унікальність імітаційного моделювання полягає в тому, що воно дозволяє нам створювати та досліджувати алгоритми майже будь яких систем чи процесів, враховувати часову динаміку змінних і тд.

Вони використовуються навіть тоді, коли частина процесів досліджуваної системи взагалі не формалізована.

У більшості випадків поведінка системи описується за допомогою станів і переходів між ними. Стан системи в момент часу t у Мережах Петрі визначається розподілом маркерів по їхніх позначених місцях. Зміна цих маркерів призводить до зміни станів системи. Експерименти здійснюються шляхом прогонів програм на безлічі вхідних даних. У таких випадках говорять, що модель працює також як система.

Комбіноване (аналітичне-імітаційне) моделювання об'єднує переваги обох підходів, дозволяючи розділити процес функціонування системи на складові частини, для яких застосовуються аналітичні та імітаційні моделі відповідно. Цей метод дозволяє досліджувати нові класи систем, які можуть бути складні для дослідження окремо.

Вибір методу моделювання залежить від завдань дослідження, наявних ресурсів та обмежень часу, що визначає дослідник.

## **1.2 Ціль моделювання**

Однією з ключових цілей нашого дослідження є розкриття ефективних та оптимальних стратегій функціонування системи передачі цифрової інформації для забезпечення максимальної якості переданого мовлення. Конкретно, ми прагнемо досягти наступних цілей:

* *Оцінка Якості Передачі*: Провести комплексний аналіз та оцінку якості передачі мовлення в системі, зокрема, враховуючи вплив часу передачі пакетів та ймовірність їхнього знищення на якість передачі.
* *Визначення Частоти Знищення Пакетів*: Встановити частоту знищення пакетів (відношення кількості знищених пакетів до часу симуляції), щоб уникнути погіршення якості мовлення та забезпечити ефективну передачу інформації.
* *Оптимізація Часу Передачі*: У випадку перевищення порогового рівня знищення пакетів, впровадження автоматичного регулювання часу передачі на канал для прискорення передачі і збереження якості мовлення.
* *Дослідження Частоти Підключення Ресурсів*: Визначення частоти підключення додаткових ресурсів (відношення кількості підключення ресурсу до загального часу симуляції) для забезпечення сталої якості мовлення та уникнення перевищення порогового рівня знищення пакетів.

**1.3 Концептуальна модель**

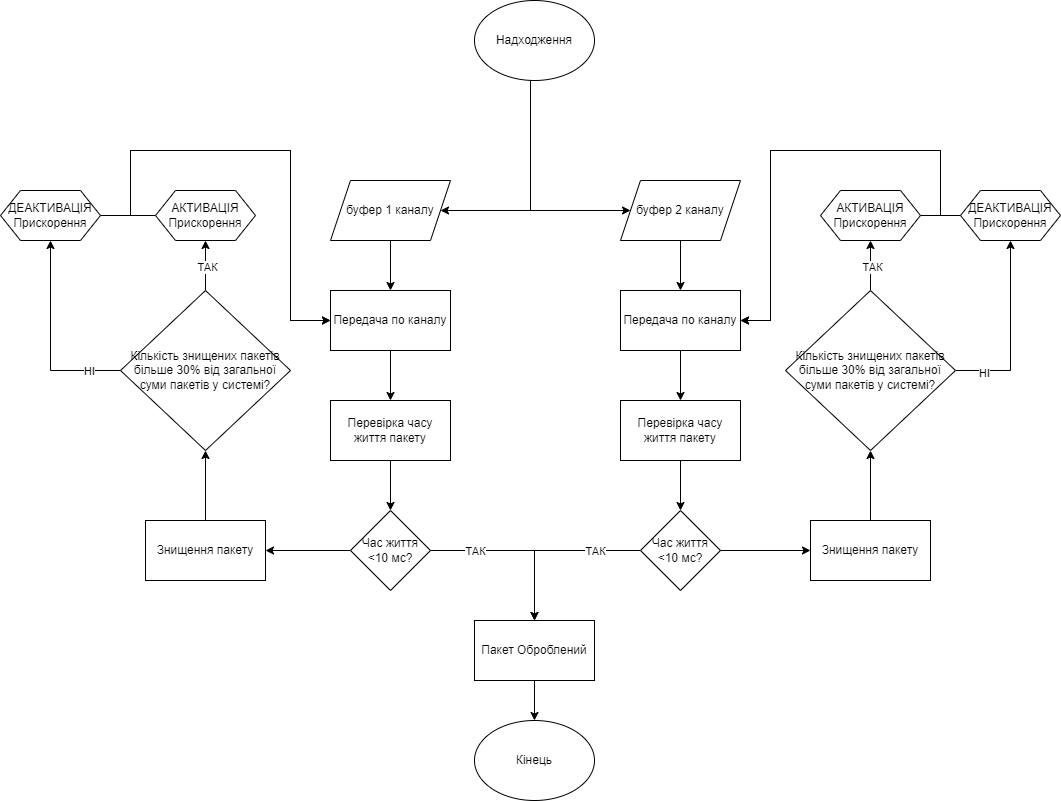


Рисунок 1.3 – Концептуальна модель роботи системи

На рисунку 1.3 показана концептуальна модель поведінки транзитних каналів. Спочатку йде надходження пакетів у буфер перед каналом, після чого пакет передається по каналу. Далі йде перевірка життя пакету, якщо його тривалість життя <10 мс, то все окей він рахується обробленим і йде в кінець, якщо ж ні то пакет рахується знищеним. Після чого йде перевірка на кількість знищених пакетів (все це робиться в продовж роботи програми) і якщо їх кількість по відношенню до всіх створених пакетів буде перевищувати 30% відсотків, то активується прискорення, що прискорює швидкість передачі пакетів у системі на 1 мс. З 5 до 4. Все це відбувається доти доки не вийде зазначений час моделювання системи.

**1.4 Вхідні та вихідні дані, параметри моделі**

Таблиця 1.4 – Вхідні параметри моделі

|  |  |  |
| --- | --- | --- |
| **Параметри** | **Значення** | **Опис** |
|  | 1 | Кількість пакетів які надходять за раз |
|  | 6±3 мс. | Швидкість надходження одного пакету |
|  | 5 мс. | Стандартний час передачі пакету по каналу |
|  | 4 мс. | Прискорений час передачі пакету по каналу |
|  | 30% | Поріг активація прискорення при заданому відсотку знищення пакетів від загальної кількості |

Таблиця 1.4 – Вхідні параметри моделі

|  |  |
| --- | --- |
| **Параметри** | **Опис** |
|  | Кількість згенерованих пакетів |
|  | Кількість знищених пакетів |
|  | Кількість оброблених пакетів |
| V | Частота знищення пакетів |
|  | Кількість активації прискорення першого каналу |
|  | Кількість деактивації прискорення першого каналу |
|  | Кількість активації прискорення другого каналу |
|  | Кількість деактивації прискорення другого каналу |

## **1.5 Обмеження**

Для системи існують наступні обмеження:

* + - N1>0
    - T2> 0
    - T3> 0
    - D1> 0

## **1.6 Постановка задачі**

Основа функція моделювання системи передачі цифрової інформації це оптимізація якості передачі пакетів, частоту їх знищення та ефективного використання ресурсів. Цільова функція може бути визначена наступним чином:

* *Мінімізація Частоти Знищення Пакетів:*

Забезпечення ефективного функціонування системи шляхом мінімізації частоти знищення пакетів. Це допомагає уникнути втрат інформації та покращити загальну якість передачі.

* *Регулювання Часу Передачі:*

Автоматичне регулювання часу передачі пакетів для оптимізації якості передачі та уникнення знищення пакетів.

Оптимізація цих параметрів дозволить системі ефективно адаптуватися до змінних умов та забезпечити стабільну та надійну передачу цифрової інформації.

**РОЗДІЛ 2. ФОРМАЛІЗОВАНА МОДЕЛЬ**

**2.1 Побудова формалізованої моделі**

Мережі Петрі – це математичний і формальний інструмент, що використовується для моделювання, аналізу та перевірки паралельних та розподілених систем. Вони дозволяють представити конкурентну активність та взаємодію різних компонентів системи у зручній формі. Основними складовими мереж Петрі є позиції, переходи та дуги, які визначають структуру моделі.

Позиції відображають стани системи і мають певну ємність для маркерів, які представляють одиниці інформації чи ресурсів. Кількість маркерів у позиціях відображає стани системи.

Переходи представляють події чи дії, які можуть траплятися в системі та впливати на її стан. Активація переходу відбувається, коли всі його вхідні позиції мають достатню кількість токенів, і маркери переміщуються через вихідні дуги до наступних позицій.

Дуги визначають напрямок переміщення маркерів між позиціями і переходами. Вони можуть бути вхідними (від позиції до переходу) або вихідними (від переходу до позиції).

A diagram of a diagram

Description automatically generated with medium confidence

Рисунок 2.1 – Елементи мережі[2]

На рисунку 2.1 зображені елементи формалізації мережі Петрі.

**2.2 Побудова формалізованої моделі**

Для побудови формалізованої мережі Петрі для даного завдання розглянемо процес передачі цифрової інформації та його ключові етапи:

*Надходження Пакетів:*

Позначимо позицію "Надходження Пакетів", яка буде відображати стани системи, коли нові мовлення підготовлені до передачі. Маркери в цій позиції відповідатимуть пакетам, які надходять до системи з заданою інтенсивністю за рівномірним законом розподілу.

*Транзитні Канали:*

Використовуючи переходи та дуги, позначимо транзитні канали, де рух пакетів відбувається з позиції "Надходження Пакетів" до позицій, які відображають буфери перед кожним транзитним каналом.

*Буфери Каналів:*

Введемо позиції для буферів перед кожним транзитним каналом, де маркери (пакети) можуть тимчасово накопичуватися перед подальшою передачею.

*Система Знищення Пакетів:*

Визначимо переходи та дуги, які вказують на систему знищення пакетів, де пакети, що перебувають у системі більше 10 мс, можуть бути знищені.

*Система Регулювання Часу Передачі:*

Введемо переходи та дуги для системи регулювання часу передачі. Якщо знищення пакетів перевищує 30%, система автоматично прискорює час передачі до 4 мс на канал.

*Система Підключення Ресурсів:*

Додамо переходи та дуги, що регулюють систему підключення додаткових ресурсів. Якщо частота знищення пакетів досягає прийнятного рівня, система може підключити додаткові ресурси для покращення якості передачі.

*Система Відключення Ресурсів:*

Введемо переходи та дуги, що вказують на можливість відключення додаткових ресурсів, коли рівень частоти знищення пакетів знову стає прийнятним.

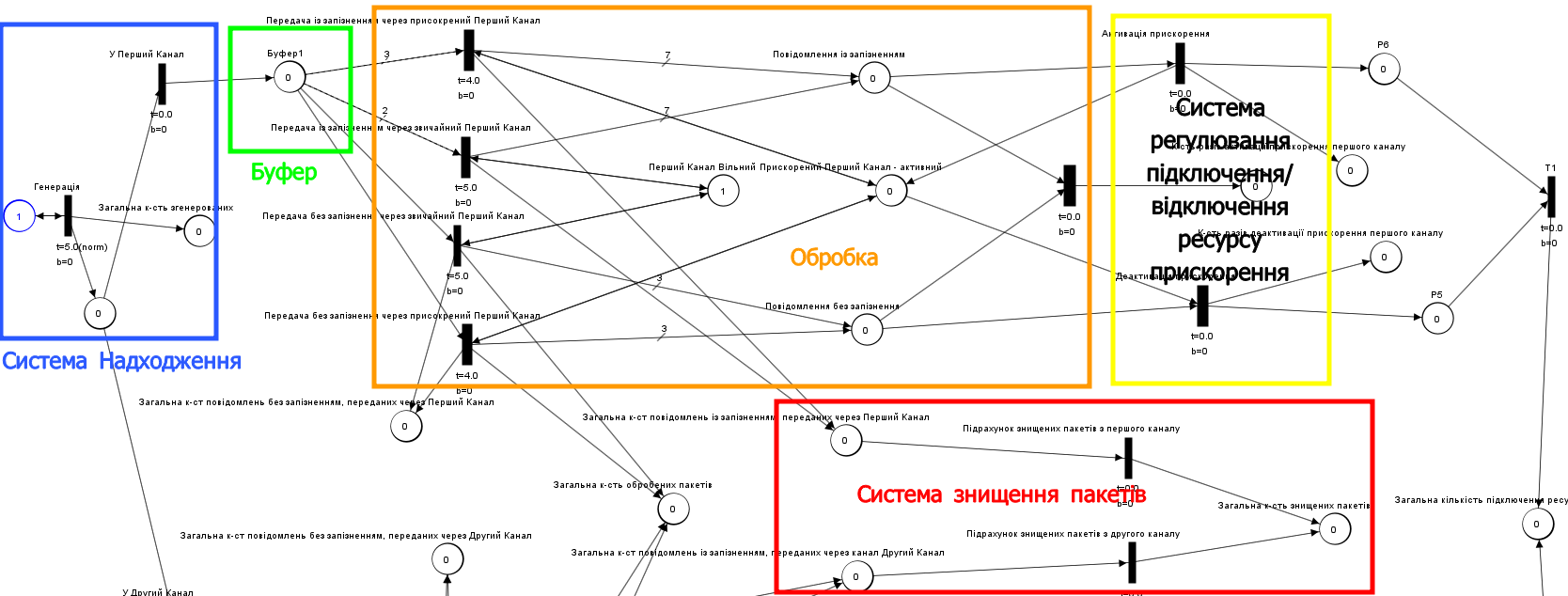


Рисунок 2.2 – Системи використані в мережі

Система в загальному працює за таким принципом. Я опишу тільки перший канал, бо другий канал працює ідентично:   
Пакет надходить в систему з інтервалом 6±3 мс розподіленим за нормальним законом розподілу , після чого потрапляє в буфер каналу звідки він потрапить на обробку. За раз можна пропустити тільки один пакет через канал, швидкість передачі пакету по каналу становить 5 мс. Взагалі даний функціонал обробки життя пакетів, підрахунку відсотків знищення і тд не підтримується програмним забезпеченням PetriObjModel, але я знайшов вихід (звісно через костилі і інтерпретації, як ми робили в лабараторній роботі з автобусом і підрахунком прибутку). Основна задумка полягає в інтерпретації кожного повідомлення у вигляді очків.

Якщо отримане повідомлення з запізненням, його "кратність" буде 10 очок. У той час, як повідомлення, яке приходить вчасно, має 3 очки. Потім запускається процес, де частина цих "очок" балансується між собою. Наприклад, якщо після обробки повідомлень з запізненням залишається певна кількість очок (у випадку вище - 7 очок), це може вказувати на необхідність активації певного механізму для покращення обробки. У нашому прикладі, використовується поняття "Активація прискорення", повідомляючи, що треба виконати прискорення передачі з 5 мс до 4 мс. Тут потрібно більш детально пояснити, як працює цей механізм та його нюанси. В декого може виникнути питання, а що буде якщо маркери, які із запізненням не буде з чим порівнювати маркерів без запізнення – невже тоді всі 7 маркерів підуть у активацію каналу? Хочу запевнити всіх що таке не можливо, причина цьому те, що маркери із запізненням з’являються лише тоді, коли в нас є черга на обробку в 2 маркери (для звичайного каналу) і в 3 маркери (для каналу із прискоренням). Тобто в нас завжди буде з чим порівнювати маркери між собою – бо не можливо позначити маркери, як ті що запізнилися, не маючи маркерів, які відповідають за повідомлення надійшовши вчасно. Простими словами – пакет не може запізнитися, якщо звичайний канал обробки пустий («вільний»).

За цим же принципом, якщо повідомлення надходять вчасно, то система вже працює оптимально, і немає потреби в активації додаткових ресурсів. Тобто якщо в нас є хоча б одна фішка в позиції «Повідомлення без запізнення», то в активації прискорення немає ніякого сенсу.

Таким чином, система вирішується динамічно в залежності від ваги та кількості повідомлень, що надходять вчасно чи з запізненням, що визначає подальші дії для оптимізації її функціонування.

Як результат роботи системи підсумуємо: В нас генерується пакет, який приходить в наш канал, звідки він одразу потрапляє в обробник, де в нас цілком вільний канал, і відбувається інтерпретація цього маркера як 3 очки, записується в оброблені системою і тд. Далі система чекає надходження наступних пакетів, бо деактивація ресурсу може відбутися тільки за умови активації прискорення. Ми чекаємо коли в нас будуть маркери для порівняння, щоб подивитися чому пакет запізнився і чи потрібно нам прискорення в такому випадку. Відбувається це тому що в нас стоїть найвищий пріоритет в позиції порівняння очок. Як результат виконується це до того моменту поки система не згенерує пакети частіше ніж може обробити наш канал. Умовно він обробив вчасно 2 пакети, які записалися у вигляді 6 очок. А тепер прийшло одразу два пакети(що є сигналом для активації переходу), вони вже йдуть у «Передача із запізненням через звичайний канал» де записуються у вигляді 7 очок. Після чого відбувається порівняння: 7-6=1. В результаті залишився 1 маркер, який йде у активацію прискорення, символізуючи системі що треба пришвидшити обробку.

Звісно система працює складніше і загалом це досить костильний варіант але вже, яку умову маємо, таку і робимо. Також завдання сформульовано так, що його можна зрозуміти у двох різних варіантах:   
1)Тарнзитні – «ті через, які проходить щось транзитом», без затримок, напряму клієнту. Тобто вони теоретично не можуть бути один за одним.

2)Або два послідовних канали, тоді вони вже не транзитні, бо в них можуть виникати затримки, збої і пакет вже йде не напряму, а через два послідовні канали.

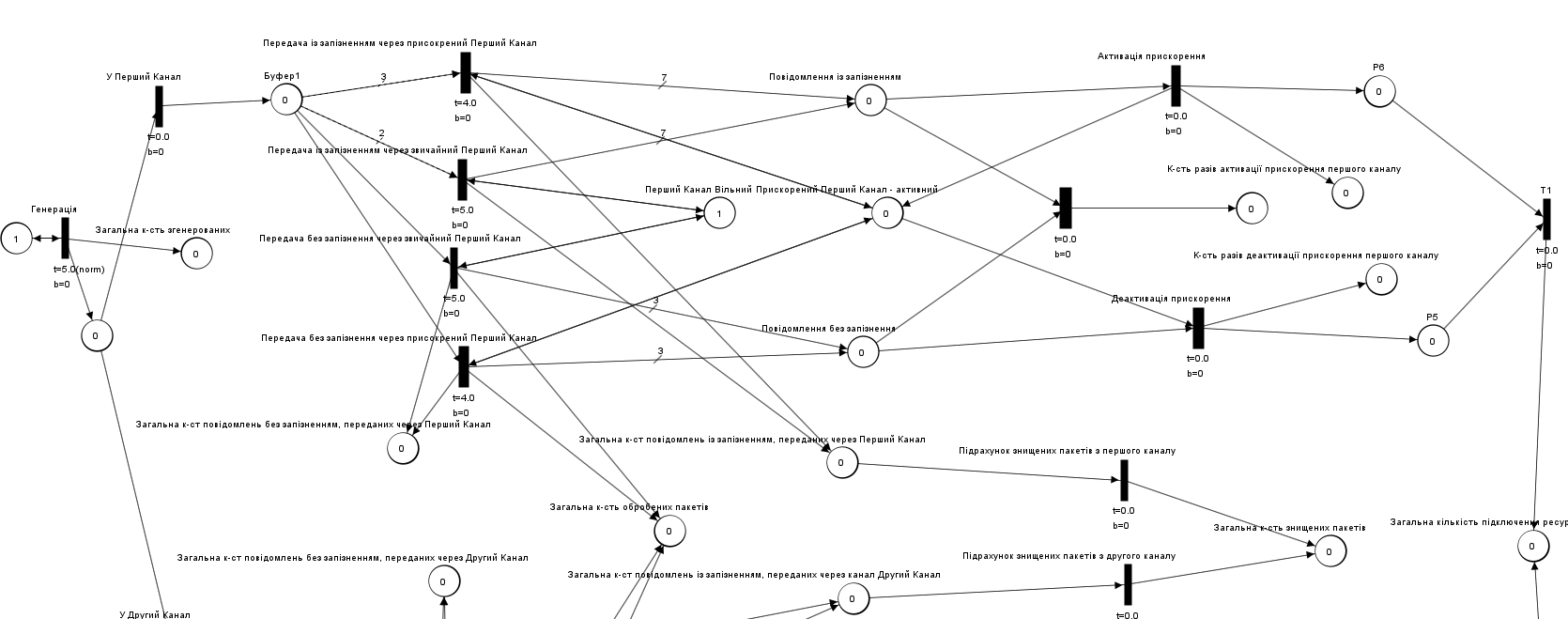


Рисунок 2.3 – Схема першого каналу

Схема другого каналу має ідентичний вигляд.

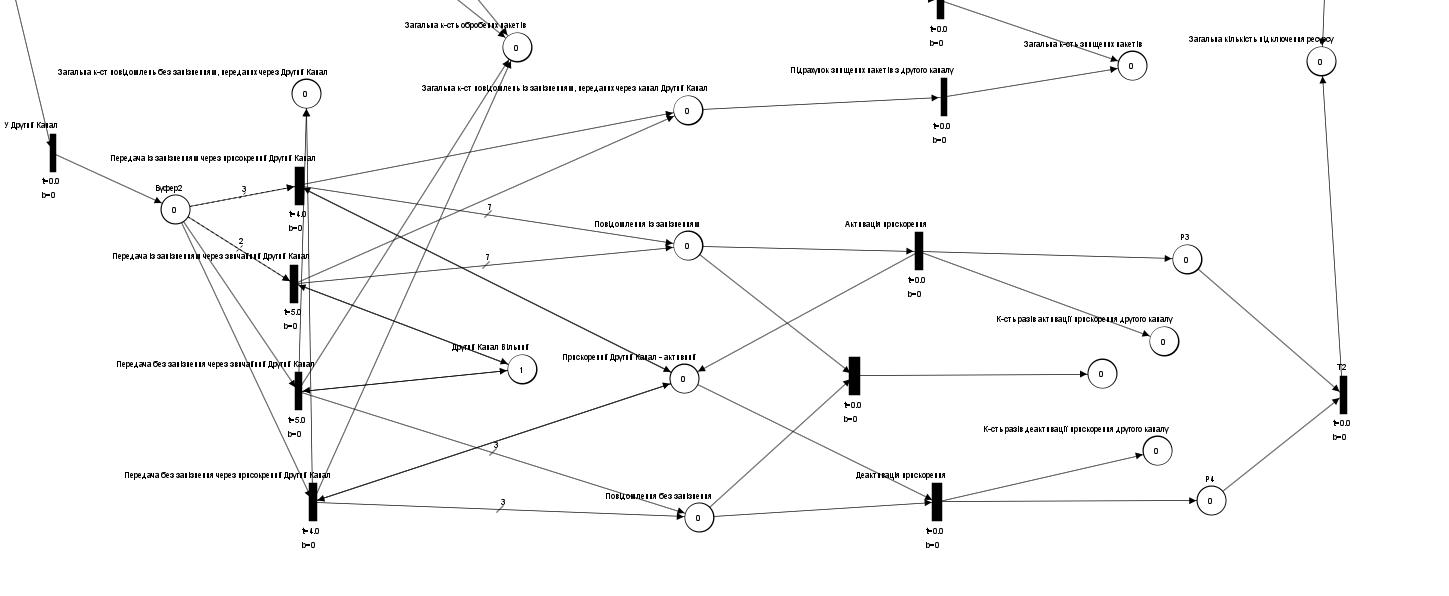


Рисунок 2.4 – Схема другого каналу

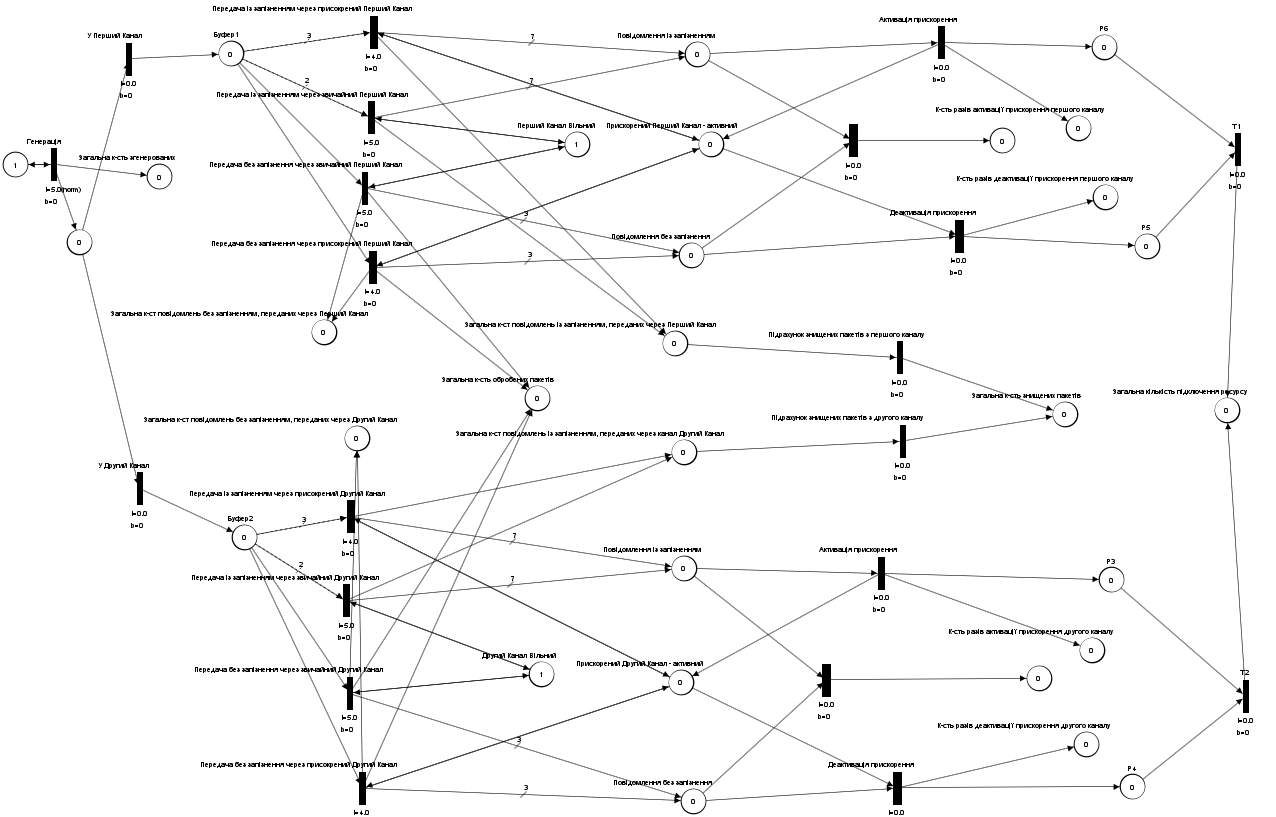


Рисунок 2.5 – Загальна схема

Загальний вигляд мережі для даної задачі.

**РОЗДІЛ 3 РЕАЛІЗАЦІЯ МОДЕЛІ**

Для розробки моделі була використана бібліотека PetriObjModelPaint, яка спрямована на моделювання систем на базі об'єктів Петрі. Цей підхід передбачає швидке та гнучке створення коду для моделювання складних систем дискретних подій. Основною ідеєю є використання стохастичної багатоканальної мережі Петрі для опису поведінки моделі, а її композицію базується на об'єктно-орієнтованій технології.

Програмне забезпечення для моделювання об'єктів Петрі написано на мові програмування Java. Воно включає пакет PetriObjLib, що реалізує алгоритм симуляції Петрі-об'єктів, а також пакети для графічного представлення мережі. Важливо відзначити, що виникне помилка, якщо перехід у мережі Петрі не має вхідних або вихідних місць.

Процес створення моделі передбачає підготовку списку Петрі-об'єктів та визначення зв'язків між ними. Для цього використовується клас PetriObjModel. Метод go(double time) цього класу запускає процес моделювання. Важливо відзначити, що виникне помилка, якщо генератор затримки видає від'ємне значення.

Основними завданнями програмного забезпечення є забезпечення правильного алгоритму моделювання та отримання вірних результатів, включаючи середні значення маркерів у місцях мережі Петрі, середнє значення буферів у переходах і стан мережі Петрі в останній момент моделювання.

Таблиця 3.1 – Бібліотека класів PetriObjModelPaint

|  |  |
| --- | --- |
| **Назва класу** | **Опис** |
| ArcIn | Описує параметри та методи вхідної дуги |
| ArcOut | Описує параметри та методи вихідної дуги |
| ExceptionInvalidNetStructure | Обробляє виключення про некоректну структуру мережі |
| ExceptionInvalidTimeDelay | Обробляє виключення про некоректно задану затримку |
| FunRand | Реалізація генераторів за заданими законами розподілу |
| PetriNet | Містить інструменти для конструювання мережі Петрі |
| PetriObjModel | Містить інструменти для конструювання об’єктної моделі Петрі |
| PetriP | Описує параметри та методи позицій |
| PetriSim | Містить інструменти для симуляції мережі Петрі |
| PetriT | Описує параметри та методи переходів |
| StateTime | Визначає поточний час та час імітації |

На рисунку 3.1 зображено графічний інтерфейс PetriObjectModelPaint з побудованою простою мережею.

A screenshot of a computer

Description automatically generated

Рисунок 3.1 – Графічний інтерфейс PetriObjectModelPaint

Побудуємо мережу для даної задачі і вона буде мати вигляд як у попередньому розділі:

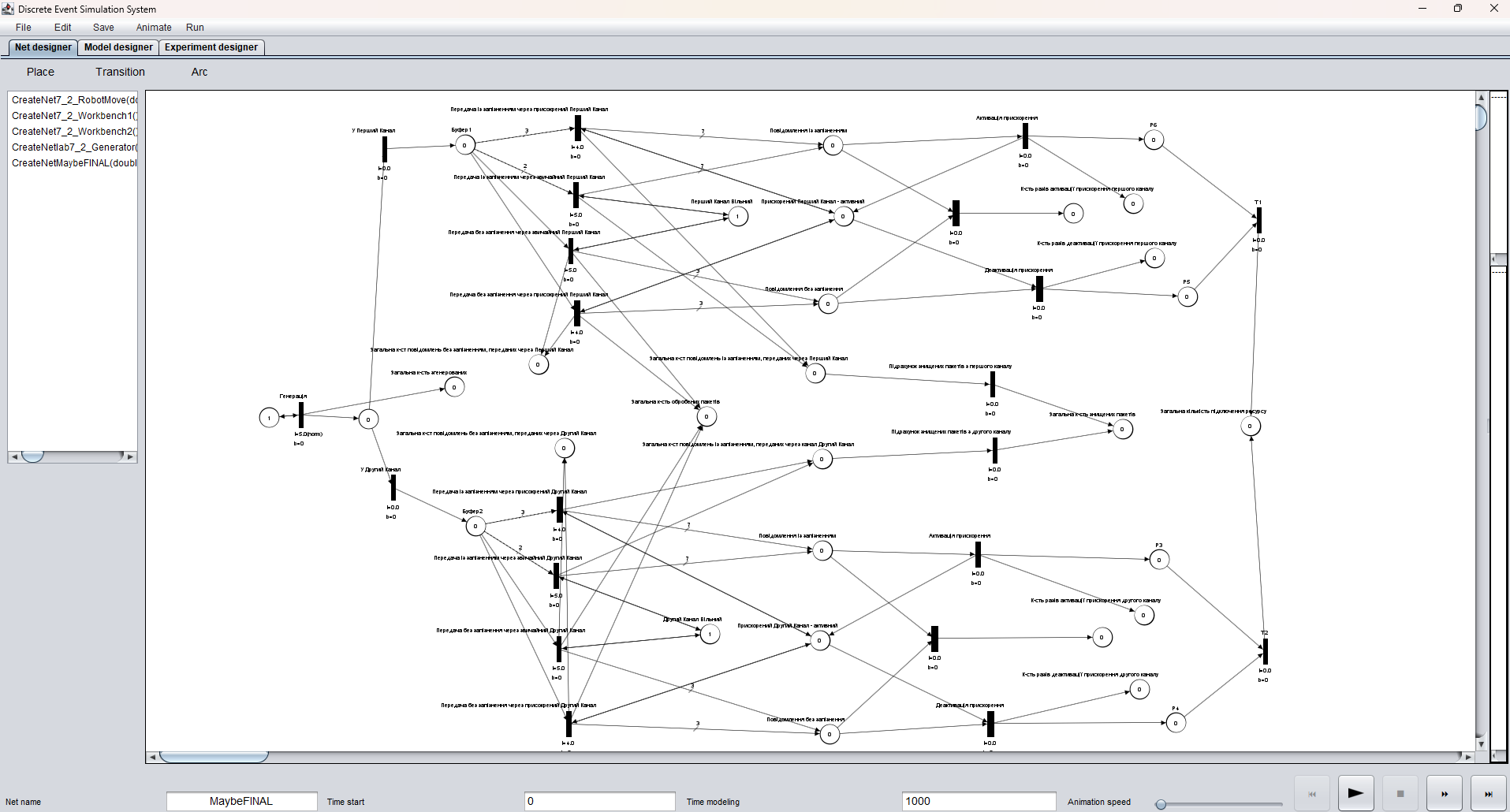


Рисунок 3.2 – Мережа Петрі побудована у PetriObjectModelPaint

Запустимо мережу та отримаємо готовий звіт, де є статистика по всім об’єктам(рис. 3.3).

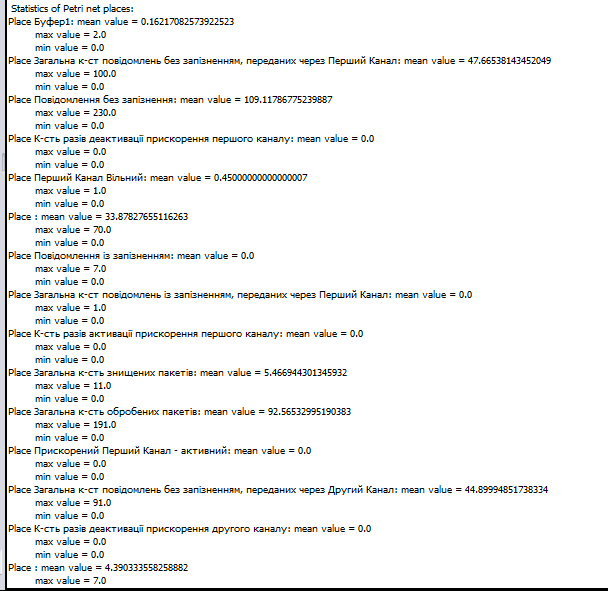


Рисунок 3.3 – Статистика виконання моделі

Таблиця 3.2 – Результати верифікації моделі

Зображення, що містить текст, знімок екрана, число, Паралель

Автоматично згенерований опис

Товстим шрифтом виділені параметри, які змінювалися та досліджувалися для проведення верифікації і порівняння з базовою моделлю(з початковими даними). Всі вихідні значення є середнім з 5 прогонів. Тобто тест №1 з параметрами (мат. Сподівання надходження 3, середнє відхилення 3, час передачі по каналу 5 і тд) запускався 5 разів, після чого знаходилося середнє значення і записувалося в таблицю. Для більш детального ознайомлення ось таблиця прогонів кожного з параметрів:

Таблиця 3.3 – Детальні результати верифікації моделі



З результатів прогону моделі можна зробити висновок, що швидкість обробки елементу в 5 мс. Цілком вистачає при нормально розподіленій генерації 6 ± 3 мс. Як бачимо при таких значеннях частота підключення ресурсу складає всього 0,0022 і зростає відносно того, як зменшується швидкість генерації пакетів, також при зменшені швидкості зростає частота знищення пакетів як і кількість підключень ресурсу. В даному завданні частота знищення пакетів характеризується, як відношення знищених пакетів до часу моделювання і частота підключення ресурсу це відношення кількості підключеного ресурсу до часу моделювання.

З таблиць можна зробити висновок, що взагалі ресурс прискорення не сильно потрібний в базовій моделі лише коли швидкість генерації зросте до 3 ± 3, тільки тоді є якийсь сенс в цьому. Подивимося що буде якщо погратися з параметром допустимості знищення пакетів:

Таблиця 3.4 – Зміни в моделі при різних параметрах  
Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

З таблиці можемо зробити висновок, що частота підключення ресурсу зростає при зменшенні допустимого відсотка знищених пакетів. А от частота знищення пакетів тримається приблизно на одному рівні для того, щоб відповідати заданим параметрам моделі. Але від себе ще можу додати – що дана задача погано реалізується в пакеті PetriObjModel, тому я рекомендував би використовувати Arena Simulation, тому є можливість того, що параметри матимуть розходження.

**РОЗДІЛ 4 ПРОВЕДЕННЯ ЕКСПЕРИМЕНТІВ НА МОДЕЛІ**

## **4.1 Постановка задачі**

Вибір параметрів експерименту. Для початку я звернувся до Пані Інни за порадою, який експеримент мені варто провести і шляхом обговорення прийшли до висновку, що тут потрібен Регресійний експеримент, а от над якими параметрами – вже залишається за мною. Насправді на консультаціях ми прийшли до висновку що тут мало що можна оцінити та порівняти, тому всі подальші експерименти можуть мати мало чого вагомого для системи. Але основною умовою є те що не можна змінювати час надходження пакетів у систему. Тому давайте подумаємо, які параметри важливі під час передачі інформації в системі? Перше що приходить в голову – це швидкість обробки, адже від цього залежить те скільки пакетів буде оброблено, знищено та як часто буде підключатися ресурс прискорення.

## **4.2 Тактичне планування**

Нам потрібно визначити час симуляції для експериментів та кількість прогонів для кожного.

Почнемо з визначення часу симуляції. Для цього нам необхідно знайти перехідний період, який можливо визначити за результатами пробних експериментів з моделлю системи. Візьмемо до прикладу частоту підключення ресурсу. Запустимо кожні 500 одиниць час модель по 15 разів і зберимо середні показники(рис 4.2).

Зображення, що містить текст, знімок екрана, Шрифт, ряд

Автоматично згенерований опис

Рисунок 4.2 – Середня частота підключення ресурсу кожні 500 одиниць часу

Як ми бачимо, в цілому система нормалізується десь після 2000 одиниць часу. Тому для досліджень будемо брати 2000 або 2500.

Далі за наведеною формулою Чебишева[2] 4.3, визначено необхідну кількість запусків алгоритму.

(4.2),

За умови, що довірча ймовірність дорівнює β=0,95 і точність вимірювання відгуку моделі рівна дисперсії відгуку моделі, то кількість прогонів буде 20 (табл. 4.4). Отже для всіх подальших експериментів будемо робити по 20 прогонів.

Таблиця 4.3 — Кількість прогонів за нерівністю Чебишева[2]

A table with numbers and symbols

Description automatically generated

## **4.3 Стратегічне планування**

Проведено запуск моделі з різними параметрами. За мету поставлено максимізувати кількість оброблених пакетів і мінімізувати кількість підключень ресурсу. Базові параметри моделі це 5 мс звичайна передача і 4 мс при прискореній передачі. Також час симуляції моделі становив 2000 умовних одиниць.

Далі подивимося на те, як впливає час передачі пакету по каналу звичайному і прискореному на частоту знищення пакетів (на кількість повідомлень за умовну одиницю часу). Графік наведено на рисунку 4.4. Проведено експеримент для пошуку оптимальних значень на деякій множині можливих значень змінної n(standard) and n(speedup). І як ми бачимо кількість активацій прискорення залежить експоненційно від кількості знищених пакетів.

Звичайний:

Зображення, що містить текст, знімок екрана, ряд, число

Автоматично згенерований опис

Рисунок 4.4 — Вплив швидкості оброблення в звичайному каналі на частоту знищення ресурсів.

Прискорений:

Зображення, що містить текст, ряд, знімок екрана, схема

Автоматично згенерований опис

Рисунок 4.5 — Вплив швидкості оброблення в прискореному каналі на частоту знищення ресурсів.

Як ми бачимо, то частота знищення пакетів нижча коли швидкість обробки на каналах вища і це логічно, бо тоді система встигає опрацювати більше пакетів за одиницю часу. Хоча загалом це також залежить від швидкості надходження пакетів – проте за умови задачі даний параметр не змінний, тому не можу привести результати його дослідження.

Далі проведемо регресивний аналіз впливу фактора.

У якості вхідних параметрів візьмемо той же час звичайної обробки(Х0) та час пришвидшеної обробки , а вихідні параметри візьмемо: частоту знищення пакетів(У1). Як бачимо це буде лінійна багатофакторна модель.

Всі розрахунки були виконані за допомогою Microsoft Excel (за умовою не було ніяких обмежень на це).

Для початку в нас є задана таблиця:

Таблиця 4.6 — Таблиця для регресійного аналізу

Зображення, що містить текст, знімок екрана, число, ряд

Автоматично згенерований опис

Тепер встановимо параметри запуску регресійного аналізу:

Таблиця 4.7 — Параметри запуску для регресійного аналізу

Зображення, що містить текст, ряд, знімок екрана, число

Автоматично згенерований опис

Запускаємо аналіз та отримаємо статистику по нашому експерименту.

Зображення, що містить текст, знімок екрана, число, Шрифт

Автоматично згенерований опис

Рисунок 4.8 — Регресійний Аналіз

З того що R^2 має досить наближене значення до 1, можна зробити висновок, що наша побудована модель якісна та цілком достатня для аргументації залежностей частоти знищення пакетів від часу обробки. Також за критерієм Фішера знаущий F 0.0003 і це менше за 0,05 – робимо висновок що наша модель адекватна за крітерієм Фішера і наші параметри мають значний вплив на нашу модель. Також ми можемо побачити, що для Базової швидкості коефіцієнти є статистично не значимими, або статистичні нулі, тобто дорівнюють 0, але в той же час наша модель є адекватною. Це один з рідких випадків, коли коефіцієнти за аналізом наче і не значать нічого але в той же час і значать. Це значить що ми не можемо пояснити вплив кожної змінної окремо але ми можемо оцінити їхній сумарний ефект на модель.

Виведемо на графік результати нашого аналізу та рвіняння регресії.

Зображення, що містить текст, ряд, знімок екрана, Графік

Автоматично згенерований опис

Рисунок 4.9 – Графік та рівняння регресії

**ВИСНОВКИ**

Уході виконання даної роботи було розглянуто побудову моделі для роботи системи передачі пакетів. Модель була побудована з використання мереж Петрі та програми для імітації роботи PetriObjModel.

У процесі моделювання системи передачі цифрової інформації були враховані різні параметри, такі як час передачі пакетів по каналах, інтервал між надходженням пакетів, час передачі пакету по каналу з виростанням ресурсів, неприпустимий час передачі пакету по каналах та відсоткове значення неприпустимості знищення пакетів.

Було проведено декілька експериментів для перевірки роботи моделі, і отримані результати підтвердили коректність її функціонування. Зокрема, виявлено, що зі збільшенням математичного сподівання надходження пакетів - частота відмов та частота підключення ресурсу системи зменшуються (так як час надходження пакетів більший за час їх обробки).

Додатково був проведений регресійний аналіз для альтернативних моделей, де швидкість передачі пакетів по каналу були більшими або меншими за базові. Результати свідчать, що час обробки пакетів має вагомий вплив на роботу програми, кількість знищених пакетів та підключення ресурсу.

Ця робота підкреслила важливість моделювання систем передачі інформації для визначення оптимальних параметрів та покращення ефективності системи. За допомогою моделювання можна зменшити кількість відмов та підключень додаткового ресурсу, що, в свою чергу, сприятиме підвищенню продуктивності системи.

**СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ**

**1)**[**https://www.wikiwand.com/uk/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D1%96\_%D0%9F%D0%B5%D1%82%D1%80%D1%96**](https://www.wikiwand.com/uk/%D0%9C%D0%B5%D1%80%D0%B5%D0%B6%D1%96_%D0%9F%D0%B5%D1%82%D1%80%D1%96)

**2)** **Стеценко, І.В. Моделювання систем: Навчальний посібник**

[**https://do.ipo.kpi.ua/mod/resource/view.php?id=39170**](https://do.ipo.kpi.ua/mod/resource/view.php?id=39170)

**ДОДАТОК А**

***NetSim.java***

package LibNet;

import PetriObj.ArcIn;

import PetriObj.ArcOut;

import PetriObj.ExceptionInvalidNetStructure;

import PetriObj.ExceptionInvalidTimeDelay;

import PetriObj.PetriNet;

import PetriObj.PetriP;

import PetriObj.PetriT;

import java.util.ArrayList;

import java.util.Random;

public class NetLibrary {

public static PetriNet CreateNetMaybeFINAL(double generation, double deviation, int points\_for\_lost, int points\_for\_inTime, double basic, double speedUp) throws ExceptionInvalidNetStructure, ExceptionInvalidTimeDelay {

ArrayList<PetriP> d\_P = new ArrayList<>();

ArrayList<PetriT> d\_T = new ArrayList<>();

ArrayList<ArcIn> d\_In = new ArrayList<>();

ArrayList<ArcOut> d\_Out = new ArrayList<>();

d\_P.add(new PetriP("Буфер1",0));

d\_P.add(new PetriP("Загальна к-ст повідомлень без запізненням, переданих через Перший Канал",0));

d\_P.add(new PetriP("Повідомлення без запізнення",0));

d\_P.add(new PetriP("К-сть разів деактивації прискорення першого каналу",0));

d\_P.add(new PetriP("Перший Канал Вільний",1));

d\_P.add(new PetriP("",0));

d\_P.add(new PetriP("Повідомлення із запізненням",0));

d\_P.add(new PetriP("Загальна к-ст повідомлень із запізненням, переданих через Перший Канал",0));

d\_P.add(new PetriP("К-сть разів активації прискорення першого каналу",0));

d\_P.add(new PetriP("Загальна к-сть знищених пакетів",0));

d\_P.add(new PetriP("Загальна к-сть обробених пакетів",0));

d\_P.add(new PetriP("Прискорений Перший Канал - активний",0));

d\_P.add(new PetriP("Загальна к-ст повідомлень без запізненням, переданих через Другий Канал",0));

d\_P.add(new PetriP("К-сть разів деактивації прискорення другого каналу",0));

d\_P.add(new PetriP("",0));

d\_P.add(new PetriP("",1));

d\_P.add(new PetriP("Загальна к-ст повідомлень із запізненням, переданих через канал Другий Канал",0));

d\_P.add(new PetriP("Другий Канал Вільний",1));

d\_P.add(new PetriP("Повідомлення без запізнення",0));

d\_P.add(new PetriP("К-сть разів активації прискорення другого каналу",0));

d\_P.add(new PetriP("Повідомлення із запізненням",0));

d\_P.add(new PetriP("Буфер2",0));

d\_P.add(new PetriP("Прискорений Другий Канал - активний",0));

d\_P.add(new PetriP("Загальна к-сть згенерованих",0));

d\_P.add(new PetriP("Загальна кількість підключення ресурсу",0));

d\_P.add(new PetriP("P3",0));

d\_P.add(new PetriP("P4",0));

d\_P.add(new PetriP("P5",0));

d\_P.add(new PetriP("P6",0));

d\_P.add(new PetriP("P1",0));

d\_T.add(new PetriT("Передача без запізнення через звичайний Перший Канал",basic));

d\_T.add(new PetriT("Деактивація прискорення",0.0));

d\_T.get(1).setPriority(5);

d\_T.add(new PetriT("Передача із запізненням через звичайний Перший Канал",basic));

d\_T.get(2).setPriority(2);

d\_T.add(new PetriT("",0.0));

d\_T.get(3).setPriority(6);

d\_T.add(new PetriT("Активація прискорення",0.0));

d\_T.get(4).setPriority(2);

d\_T.add(new PetriT("Підрахунок знищених пакетів з другого каналу",0.0));

d\_T.add(new PetriT("Передача без запізнення через присокрений Перший Канал",speedUp));

d\_T.get(6).setPriority(3);

d\_T.add(new PetriT("Передача із запізненням через присокрений Перший Канал",speedUp));

d\_T.get(7).setPriority(4);

d\_T.add(new PetriT("Передача без запізнення через присокрений Другий Канал",speedUp));

d\_T.get(8).setPriority(3);

d\_T.add(new PetriT("",0.0));

d\_T.get(9).setPriority(6);

d\_T.add(new PetriT("Передача із запізненням через присокрений Другий Канал",speedUp));

d\_T.get(10).setPriority(4);

d\_T.add(new PetriT("Генерація",generation));

d\_T.get(11).setDistribution("norm", d\_T.get(11).getTimeServ());

d\_T.get(11).setParamDeviation(deviation);

d\_T.add(new PetriT("Деактивація прискорення",0.0));

d\_T.get(12).setPriority(5);

d\_T.add(new PetriT("Передача без запізнення через звичайний Другий Канал",basic));

d\_T.add(new PetriT("Передача із запізненням через звичайний Другий Канал",basic));

d\_T.get(14).setPriority(2);

d\_T.add(new PetriT("Активація прискорення",0.0));

d\_T.get(15).setPriority(2);

d\_T.add(new PetriT("Підрахунок знищених пакетів з першого каналу",0.0));

d\_T.add(new PetriT("T1",0.0));

d\_T.add(new PetriT("T2",0.0));

d\_T.add(new PetriT("У Другий Канал",0.0));

d\_T.add(new PetriT("У Перший Канал",0.0));

d\_In.add(new ArcIn(d\_P.get(4),d\_T.get(2),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(2),2));

d\_In.get(1).setInf(true);

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(2),1));

d\_In.add(new ArcIn(d\_P.get(18),d\_T.get(9),1));

d\_In.add(new ArcIn(d\_P.get(20),d\_T.get(9),1));

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(13),1));

d\_In.add(new ArcIn(d\_P.get(17),d\_T.get(13),1));

d\_In.add(new ArcIn(d\_P.get(16),d\_T.get(5),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(0),1));

d\_In.add(new ArcIn(d\_P.get(4),d\_T.get(0),1));

d\_In.add(new ArcIn(d\_P.get(28),d\_T.get(17),1));

d\_In.add(new ArcIn(d\_P.get(27),d\_T.get(17),1));

d\_In.add(new ArcIn(d\_P.get(2),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(11),d\_T.get(1),1));

d\_In.add(new ArcIn(d\_P.get(22),d\_T.get(8),1));

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(8),1));

d\_In.add(new ArcIn(d\_P.get(15),d\_T.get(11),1));

d\_In.add(new ArcIn(d\_P.get(25),d\_T.get(18),1));

d\_In.add(new ArcIn(d\_P.get(26),d\_T.get(18),1));

d\_In.add(new ArcIn(d\_P.get(7),d\_T.get(16),1));

d\_In.add(new ArcIn(d\_P.get(20),d\_T.get(15),1));

d\_In.add(new ArcIn(d\_P.get(11),d\_T.get(6),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(6),1));

d\_In.add(new ArcIn(d\_P.get(2),d\_T.get(3),1));

d\_In.add(new ArcIn(d\_P.get(6),d\_T.get(3),1));

d\_In.add(new ArcIn(d\_P.get(29),d\_T.get(20),1));

d\_In.add(new ArcIn(d\_P.get(22),d\_T.get(10),1));

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(10),3));

d\_In.get(27).setInf(true);

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(10),1));

d\_In.add(new ArcIn(d\_P.get(17),d\_T.get(14),1));

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(14),2));

d\_In.get(30).setInf(true);

d\_In.add(new ArcIn(d\_P.get(21),d\_T.get(14),1));

d\_In.add(new ArcIn(d\_P.get(6),d\_T.get(4),1));

d\_In.add(new ArcIn(d\_P.get(11),d\_T.get(7),1));

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(7),3));

d\_In.get(34).setInf(true);

d\_In.add(new ArcIn(d\_P.get(0),d\_T.get(7),1));

d\_In.add(new ArcIn(d\_P.get(29),d\_T.get(19),1));

d\_In.add(new ArcIn(d\_P.get(18),d\_T.get(12),1));

d\_In.add(new ArcIn(d\_P.get(22),d\_T.get(12),1));

d\_Out.add(new ArcOut(d\_T.get(2),d\_P.get(4),1));

d\_Out.add(new ArcOut(d\_T.get(2),d\_P.get(6),points\_for\_lost));

d\_Out.add(new ArcOut(d\_T.get(2),d\_P.get(7),1));

d\_Out.add(new ArcOut(d\_T.get(9),d\_P.get(14),1));

d\_Out.add(new ArcOut(d\_T.get(13),d\_P.get(18),points\_for\_inTime));

d\_Out.add(new ArcOut(d\_T.get(13),d\_P.get(12),1));

d\_Out.add(new ArcOut(d\_T.get(13),d\_P.get(17),1));

d\_Out.add(new ArcOut(d\_T.get(13),d\_P.get(10),1));

d\_Out.add(new ArcOut(d\_T.get(5),d\_P.get(9),1));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(2),points\_for\_inTime));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(1),1));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(4),1));

d\_Out.add(new ArcOut(d\_T.get(0),d\_P.get(10),1));

d\_Out.add(new ArcOut(d\_T.get(17),d\_P.get(24),1));

d\_Out.add(new ArcOut(d\_T.get(1),d\_P.get(3),1));

d\_Out.add(new ArcOut(d\_T.get(1),d\_P.get(27),1));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(22),1));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(18),points\_for\_inTime));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(12),1));

d\_Out.add(new ArcOut(d\_T.get(8),d\_P.get(10),1));

d\_Out.add(new ArcOut(d\_T.get(11),d\_P.get(23),1));

d\_Out.add(new ArcOut(d\_T.get(11),d\_P.get(15),1));

d\_Out.add(new ArcOut(d\_T.get(11),d\_P.get(29),1));

d\_Out.add(new ArcOut(d\_T.get(18),d\_P.get(24),1));

d\_Out.add(new ArcOut(d\_T.get(16),d\_P.get(9),1));

d\_Out.add(new ArcOut(d\_T.get(15),d\_P.get(19),1));

d\_Out.add(new ArcOut(d\_T.get(15),d\_P.get(22),1));

d\_Out.add(new ArcOut(d\_T.get(15),d\_P.get(25),1));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(11),1));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(2),points\_for\_inTime));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(1),1));

d\_Out.add(new ArcOut(d\_T.get(6),d\_P.get(10),1));

d\_Out.add(new ArcOut(d\_T.get(3),d\_P.get(5),1));

d\_Out.add(new ArcOut(d\_T.get(20),d\_P.get(0),1));

d\_Out.add(new ArcOut(d\_T.get(10),d\_P.get(22),1));

d\_Out.add(new ArcOut(d\_T.get(10),d\_P.get(16),1));

d\_Out.add(new ArcOut(d\_T.get(10),d\_P.get(20),points\_for\_lost));

d\_Out.add(new ArcOut(d\_T.get(14),d\_P.get(17),1));

d\_Out.add(new ArcOut(d\_T.get(14),d\_P.get(20),points\_for\_lost));

d\_Out.add(new ArcOut(d\_T.get(14),d\_P.get(16),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(8),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(11),1));

d\_Out.add(new ArcOut(d\_T.get(4),d\_P.get(28),1));

d\_Out.add(new ArcOut(d\_T.get(7),d\_P.get(11),1));

d\_Out.add(new ArcOut(d\_T.get(7),d\_P.get(7),1));

d\_Out.add(new ArcOut(d\_T.get(7),d\_P.get(6),points\_for\_lost));

d\_Out.add(new ArcOut(d\_T.get(19),d\_P.get(21),1));

d\_Out.add(new ArcOut(d\_T.get(12),d\_P.get(13),1));

d\_Out.add(new ArcOut(d\_T.get(12),d\_P.get(26),1));

PetriNet d\_Net = new PetriNet("MaybeFINAL",d\_P,d\_T,d\_In,d\_Out);

PetriP.initNext();

PetriT.initNext();

ArcIn.initNext();

ArcOut.initNext();

return d\_Net;

}

}

***TestPetriObjSimulation.java***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package LibTest;

//import PetriObj.PetriObjModel;

import LibNet.NetLibrary;

import PetriObj.ExceptionInvalidNetStructure;

import PetriObj.ExceptionInvalidTimeDelay;

import PetriObj.PetriObjModel;

import PetriObj.PetriSim;

import java.util.ArrayList;

import java.util.List;

/\*\*

\*

\* @author Inna V. Stetsenko

\*/

public class TestPetriObjSimulation {

public static void main(String[] args) throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {

try{

List<Double> destroyedPacketsList = new ArrayList<>();

List<Double> processedPacketsList = new ArrayList<>();

List<Double> resourceConnectionsList = new ArrayList<>();

List<Double> generatedPacketsList = new ArrayList<>();

int sim\_amount = 200;

for (int i = 0; i < sim\_amount; i++) {

PetriObjModel model = getModel();

model.setIsProtokol(false);

double timeModeling = 2000; // Adjust the modeling time as needed

model.go(timeModeling);

String destName;

String proccessed;

String speedup;

String ganarated;

// Record the parameters after each run

double destroyedPackets = model.getListObj().get(0).getNet().getListP()[9].getMark();

// destName = model.getListObj().get(0).getNet().getListP()[9].getName();

double processedPackets = model.getListObj().get(0).getNet().getListP()[10].getMark();

// proccessed = model.getListObj().get(0).getNet().getListP()[10].getName();

double resourceConnections = model.getListObj().get(0).getNet().getListP()[24].getMark();

// speedup = model.getListObj().get(0).getNet().getListP()[24].getName();

double generatedPackets = model.getListObj().get(0).getNet().getListP()[23].getMark();

// ganarated = model.getListObj().get(0).getNet().getListP()[23].getName();

destroyedPacketsList.add(destroyedPackets);

processedPacketsList.add(processedPackets);

resourceConnectionsList.add(resourceConnections);

generatedPacketsList.add(generatedPackets);

// Print or store the recorded parameters

//System.out.println("Run " + (i + 1) + ":");

//System.out.println("Destroyed Packets: " + destroyedPackets);

//System.out.println("Destr Name: " + destName);

//System.out.println("Processed Packets: " + processedPackets);

//System.out.println("Proccessed Name: " + proccessed);

//System.out.println("Resource Connections: " + resourceConnections);

//System.out.println("Speed Up Name: " + speedup);

//System.out.println("Generated Packets: " + generatedPackets);

//System.out.println("Generated Name: " + ganarated);

//System.out.println("----------------------------------");

}

double avgDestroyedPackets = calculateAverage(destroyedPacketsList);

double avgProcessedPackets = calculateAverage(processedPacketsList);

double avgResourceConnections = calculateAverage(resourceConnectionsList);

double avgGeneratedPackets = calculateAverage(generatedPacketsList);

System.out.println("Average Generated Packets: " + avgGeneratedPackets);

System.out.println("Average Processed Packets: " + avgProcessedPackets);

System.out.println("Average Destroyed Packets: " + avgDestroyedPackets);

System.out.println("Average Resource Connections: " + avgResourceConnections);

//for (int i = 0; i < sim\_amount; i++){

// System.out.println("Run " + (i + 1) + ":");

// System.out.println("Destroyed Packets: " + destroyedPacketsList.get(i));

// System.out.println("Processed Packets: " + processedPacketsList.get(i));

// System.out.println("Resource Connections: " + resourceConnectionsList.get(i));

// System.out.println("Generated Packets: " + generatedPacketsList.get(i));

// System.out.println("----------------------------------");

// }0

}catch (PetriObj.ExceptionInvalidTimeDelay e) {

// Handle the exception here, e.g., log it or perform some other action

System.err.println("Caught an Exception: " + e.getMessage());

}

}

private static double calculateAverage(List<Double> list){

double sum = 0;

for(Double value : list){

sum += value;

}

return sum/list.size();

}

public static PetriObjModel getModel() throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {

ArrayList<PetriSim> list = new ArrayList<>();

list.add(new PetriSim(NetLibrary.CreateNetMaybeFINAL(6, 3, 7, 3, 5, 4)));

return new PetriObjModel(list);

}

}

***TestSim.java***

/\*

\* To change this template, choose Tools | Templates

\* and open the template in the editor.

\*/

package LibTest;

//import PetriObj.PetriObjModel;

import LibNet.NetLibrary;

import PetriObj.ExceptionInvalidNetStructure;

import PetriObj.ExceptionInvalidTimeDelay;

import PetriObj.PetriObjModel;

import PetriObj.PetriSim;

import java.util.ArrayList;

import java.util.List;

import java.io.BufferedWriter;

import java.io.FileWriter;

import java.io.IOException;

import java.util.ArrayList;

import java.util.List;

/\*\*

\*

\* @author Inna V. Stetsenko

\*/

public class TestSim {

public static void main(String[] args) throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {

int numberOfSimulations = 10;

double timeModelingStep = 100;

// int totalSimulations = 100;

double timeModeling = 100;

List<Double> averageSimTime = new ArrayList<>();

List<Double> averageDestroyedPacketsList = new ArrayList<>();

List<Double> averageProcessedPacketsList = new ArrayList<>();

List<Double> averageResourceConnectionsList = new ArrayList<>();

List<Double> averageGeneratedPacketsList = new ArrayList<>();

while(timeModeling <= 10000){

List<Double> simTime = new ArrayList<>();

List<Double> destroyedPacketsList = new ArrayList<>();

List<Double> processedPacketsList = new ArrayList<>();

List<Double> resourceConnectionsList = new ArrayList<>();

List<Double> generatedPacketsList = new ArrayList<>();

for (int run = 0; run < numberOfSimulations; run++) {

PetriObjModel model = getModel();

model.setIsProtokol(false);

model.go(timeModeling);

double destroyedPackets = model.getListObj().get(0).getNet().getListP()[9].getMark();

double processedPackets = model.getListObj().get(0).getNet().getListP()[10].getMark();

double resourceConnections = model.getListObj().get(0).getNet().getListP()[24].getMark();

double generatedPackets = model.getListObj().get(0).getNet().getListP()[23].getMark();

simTime.add(timeModeling);

destroyedPacketsList.add(destroyedPackets);

processedPacketsList.add(processedPackets);

resourceConnectionsList.add(resourceConnections);

generatedPacketsList.add(generatedPackets);

if(timeModeling > 900){

timeModelingStep = 500;

}

}

timeModeling += timeModelingStep;

// Calculate averages for each simulation

double avgRunNumber = calculateAverage(simTime);

double avgDestroyedPackets = calculateAverage(destroyedPacketsList);

double avgProcessedPackets = calculateAverage(processedPacketsList);

double avgResourceConnections = calculateAverage(resourceConnectionsList);

double avgGeneratedPackets = calculateAverage(generatedPacketsList);

// Add averages to the main lists

averageSimTime.add(avgRunNumber);

averageDestroyedPacketsList.add(avgDestroyedPackets);

averageProcessedPacketsList.add(avgProcessedPackets);

averageResourceConnectionsList.add(avgResourceConnections);

averageGeneratedPacketsList.add(avgGeneratedPackets);

}

// Write averages to CSV file

try (BufferedWriter writer = new BufferedWriter(new FileWriter("D:\\Labs\\Course-Work-Petri-Model-KPI\\output.csv"))) {

// Writing header to CSV file

writer.write("Average Run Number,Average Generated Packets,Average Processed Packets,Average Destroyed Packets,Average Resource Connections\n");

int size = averageSimTime.size();

for (int i = 0; i < size; i++) {

double avgRunNumber = averageSimTime.get(i).intValue();

double avgDestroyedPackets = averageDestroyedPacketsList.get(i);

double avgProcessedPackets = averageProcessedPacketsList.get(i);

double avgResourceConnections = averageResourceConnectionsList.get(i);

double avgGeneratedPackets = averageGeneratedPacketsList.get(i);

// Writing averages to CSV file

writer.write(avgRunNumber + ";" + avgGeneratedPackets + ";" + avgProcessedPackets + ";" + avgDestroyedPackets + ";" + avgResourceConnections + "\n");

}

System.out.println("Data written to file: output.csv");

} catch (IOException e) {

e.printStackTrace();

}

}

private static double calculateAverage(List<Double> list){

double sum = 0;

for(Double value : list){

sum += value;

}

return sum/list.size();

}

public static PetriObjModel getModel() throws ExceptionInvalidTimeDelay, ExceptionInvalidNetStructure {

ArrayList<PetriSim> list = new ArrayList<>();

list.add(new PetriSim(NetLibrary.CreateNetMaybeFINAL(6, 3, 7, 3, 5, 4)));

return new PetriObjModel(list);

}

}