

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: Імітаційна модель системи передачі цифрової інформації на основі
формалізму мереж Петрі

Керівник:

Дифучина О. Ю.

«Допущено до захисту»

«___» _____ 2024 р.

Захищено з оцінкою

Члени комісії:

Виконавець:

Головня О.Р

студент групи ІІІ-11
залікова книжка № 1107

«5» грудня 2024 р.

Інна СТЕЦЕНКО

Олександра ДИФУЧИНА

АНОТАЦІЯ.....	3
ВСТУП	4
ЗАВДАННЯ	5
РОЗДІЛ 1. КОНЦЕПТУАЛЬНА МОДЕЛЬ	6
1.1. Загальні положення. Опис системи.	6
1.2. Вхідні та вихідні дані, параметри моделі	6
1.3. Візуалізація концептуальної моделі	8
Висновки	10
РОЗДІЛ 2. ФОРМАЛІЗОВАНА МОДЕЛЬ	11
2.1 Загальні положення формалізму Мереж Петрі	11
2.2 Побудова формалізованої моделі	13
Висновки	19
РОЗДІЛ 3. АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ.....	21
3.1. Вступ та загальні положення	21
3.2. Структура класів і об'єктно-орієнтований підхід	22
3.3. Верифікація моделі:	24
3.4. Верифікація моделі	Error! Bookmark not defined.
Висновки	27
РОЗДІЛ 4. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ	29
4.1 Постановка задачі та обґрунтування параметрів експерименту	29
4.2 Час симуляції та визначення перехідного періоду	29
4.3. Кількість прогонів і забезпечення точності результатів	31
4.3.1. Вибір параметрів для дослідження	31
4.3.2 Проведення експериментів і аналіз залежностей	32
4.4 Результати регресійного аналізу	34
4.4.1 Постановка та результати регресійного аналізу	34
4.4.2 Візуалізація результатів та рівняння регресії	35
4.5. Код для проведення експериментального дослідження	35
Висновки	36
РОЗДІЛ 5. ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ СИСТЕМИ, ФОРМУЛЮВАННЯ ВИСНОВКІВ ТА ПРОПОЗИЦІЙ	37
5.1 Узагальнення результатів моделювання	37
5.2 Пропозиції щодо вдосконалення системи	37
5.3 Оцінка ефективності запропонованих змін	38
Висновки	40
ВИСНОВКИ	41
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	42
ДОДАТКИ.....	43

АНОТАЦІЯ

Головня Олександр Ростиславович. Курсова робота на тему «Імітаційна модель роботи транзитних каналів для передачі інформації на основі формалізму мережі Петрі»

Текстова частина курсової роботи складається із вступу, 5 розділів, висновків, списку використаних джерел з 5 найменувань та 1 додатку, який викладено на 41 сторінці. В тексті роботи оформлено 15 рисунків, 11 таблиць.

У роботі поставлено завдання визначити частоту відкидання пакетів і частоту підключення додаткових ресурсів для оптимізації цієї системи.

Імітаційна модель розроблена з використанням формалізму мереж Петрі та програмного забезпечення PetriObjModel

Проведено дослідження впливу змінності часу надходження і передачі пакетів на стабільність роботи системи, зокрема на частоту знищення пакетів та підключення прискорення.

Результати моделювання свідчать, що ефективність системи значно залежить від часу обробки пакетів у звичайному та прискореному режимах, а також від інтервалів їх надходження.

Зроблено висновки, що запропонований підхід до моделювання дозволяє ефективно оцінити роботу системи передачі даних, визначити оптимальні параметри її функціонування та знизити частоту відмов і використання ресурсів, що забезпечує стабільну і продуктивну роботу системи.

Ключові слова: імітаційна модель, алгоритм імітації, мережа петрі, імітаційне моделювання, передача цифрової інформації

ВСТУП

У даній курсовій роботі проаналізовано цифрові системи передачі інформації, що використовуються для мовлення у цифровому форматі. Виходячи з завдання, ця система потребує детального вивчення її функціональності та оптимізації для досягнення оптимальної продуктивності.

Завдання полягає в тому, щоб визначити частоту відкидання пакетів і частоту підключення додаткових ресурсів для оптимізації цієї системи.

Для вирішення цього завдання буде використано метод імітаційного моделювання. Методи імітаційного моделювання дозволяють відтворити функціонування системи та дослідити її характеристики.

Для моделювання використовується формалізм мереж Петрі[1]. Мережа Петрі – це графічний і математичний засіб моделювання систем і процесів. Як правило, мережами Петрі моделюють паралельні(синхронні та асинхронні) системи і процеси. Мережі Петрі надають можливість відобразити паралельні процеси та взаємодію компонентів системи. Застосування цього формалізму спрощує аналіз та розуміння системи передачі цифрової інформації.

Програмна реалізація виконана мовою програмування Java, що дозволяє ефективно створювати об'єктно-орієнтовані моделі, реалізовувати події й часові обмеження, а також бібліотека PetriObjModelPaint[2]

ЗАВДАННЯ

Виконати дослідження для такої системи:

Завдання В8. 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 прийняттого 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 сформулювати 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.

РОЗДІЛ 1. КОНЦЕПТУАЛЬНА МОДЕЛЬ

1.1. Загальні положення. Опис системи.

Концептуальна модель системи – це деталізований опис складових елементів системи, правил їх взаємодії та процесу, який моделюється, разом із зазначенням мети моделювання. Метою моделювання є оцінка частоти знищення пакетів і частоти підключення додаткових ресурсів для забезпечення стабільності передачі мовлення. Для цього потрібно виділити вхідні змінні, параметри та вихідні характеристики моделі.

Система призначена для передачі цифрового мовлення у вигляді мовних пакетів, що надходять через два транзитних канали. Кожен пакет проходить буферизацію в накопичувачах перед кожним каналом. У разі затримки пакету більше 10 мс він знищується, оскільки його надходження до декодера погіршує якість мовлення.

У випадку, коли кількість знищених пакетів перевищує 30%, система активує додатковий ресурс, який зменшує час передачі пакета через канал з 5 мс до 4 мс. Після стабілізації (коли частка знищених пакетів знову стає меншою за 30%) ресурс автоматично вимикається.

1.2. Вхідні та вихідні дані, параметри моделі

Таблиця 1.1 – Вхідні параметри моделі

Параметри	Значення	Пояснення
$N_{generated}$	1	Кількість згенерованих пакетів, що надходять
T_1	6 ± 3 мс	Швидкість надходження пакетів (Розподіл: рівномірний)
T_2	5 мс	Стандартний час передачі пакетів
T_3	4 мс	Прискорений час передачі пакетів
A_1	30%	Відсоток знищених, при якому передача прискорюється до 4 мс
T_{mod}	X мс	Час моделювання системи

У таблиці 1.1 наведено ключові вхідні параметри системи передачі мовних пакетів. Ці параметри описують характеристики потоку пакетів, зокрема час їх надходження і передачі через канали. Кожен параметр включає початкове значення, допустимий діапазон змін і розподіл імовірностей (якщо параметр має стохастичний характер). Ці значення є основою для моделювання поведінки системи в умовах реальної експлуатації.

Таблиця 1.2 – Вихідні параметри моделі

Параметри	Пояснення
$N_{generated}$	Кількість згенерованих пакетів
N_{lost}	Кількість знищених пакетів
N_{proc}	Кількість оброблених пакетів
F_{lost}	Частота знищення пакетів
N_{res}	Кількість підключення ресурсу
F_{res}	Частота підключення ресурсу

У таблиці 1.2 представлено вихідні параметри, що відображають результати роботи моделі. До них належать частота знищення пакетів, визначається як відношення кількості знищених пакетів до загальної кількості згенерованих пакетів за період моделювання $F_{lost} = \frac{N_{lost}}{N_{generated}}$ і частота підключення додаткового ресурсу $F_{res} = \frac{N_{res}}{T_{mod}}$. Ці параметри є основними показниками ефективності системи: вони показують, як часто система втрачає пакети через затримки та як часто відбувається активація ресурсу для підтримки стабільної якості передачі. Ці дані дозволяють оцінити стабільність системи та необхідність коригування вхідних параметрів.

Також, у системі передбачені наступні обмеження, які забезпечують коректність роботи моделі та стабільність передачі пакетів:

- $N_{generated} > 0$ — кількість мовних пакетів має бути більшою за нуль, що гарантує наявність даних для передачі.
- $T_1 > 0$ — час передачі пакету через перший канал не може бути нульовим або від'ємним, оскільки передача потребує позитивного часу.
- $T_3 > 0$ — час передачі пакету через другий канал також має бути додатним, що відображає реальний процес передачі.
- $A_1 > 0$ — максимально допустимий відсоток знищення, при якому відбувається прискорення передачі.

Ці обмеження встановлюють базові вимоги до параметрів системи, що дозволяє уникнути некоректних значень та забезпечити ефективність моделювання.

1.3. Візуалізація концептуальної моделі

Таким чином, на рисунку 1.1 зображено концептуальну модель роботи системи передачі пакетів через транзитні канали. Модель відображає основні етапи обробки пакетів у системі:

Буферизація: Пакети надходять у буфер перед першим каналом. Це забезпечує тимчасове зберігання пакету до його передачі.

Передача пакету через канал: Після буферизації пакет передається через канал, який займає певний час — за замовчуванням 5 мс.

Перевірка часу життя пакету: По завершенні передачі відбувається перевірка часу, що витрачено на доставку пакета. Якщо він перевищує 10 мс, пакет вважається знищеним і вилучається з системи; в іншому випадку він рахується обробленим і надходить на вихід системи.

Контроль рівня знищення пакетів: Протягом роботи програми відслідковується частка знищених пакетів. Якщо вона перевищує 30% від загальної кількості створених пакетів, активується прискорення передачі: час передачі пакета через канал скорочується на 1 мс (з 5 мс до 4 мс). Це прискорення дозволяє зменшити частку знищених пакетів і стабілізувати систему.

Циклічність процесу: Процес обробки пакетів, перевірки часу життя і контролю рівня знищення триває до завершення заданого часу моделювання.

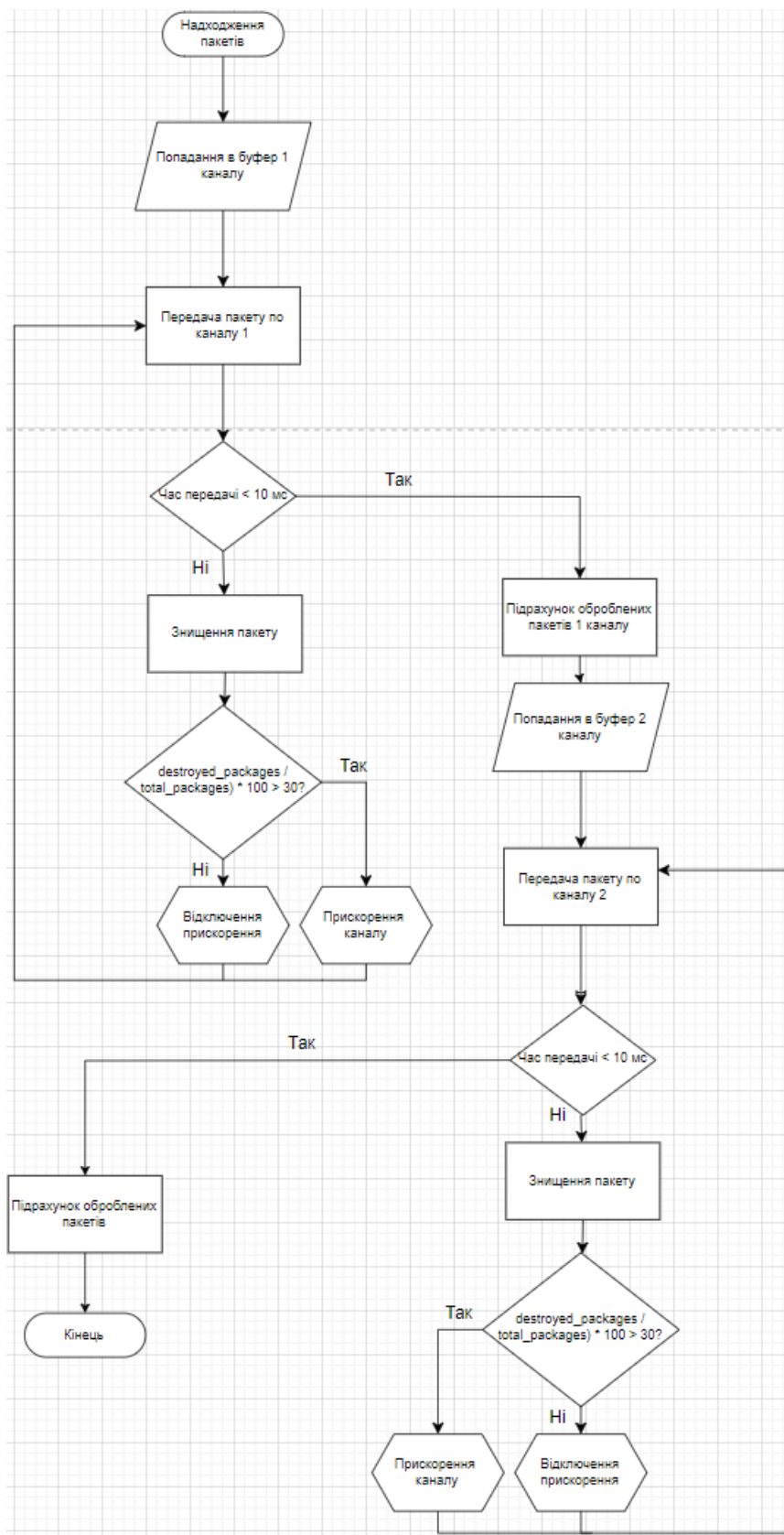


Рисунок 1.1 – Концептуальна модель роботи системи

Візуалізація концептуальної моделі показує послідовність етапів обробки пакетів, що дозволяє наочно зрозуміти логіку роботи системи і вплив на неї різних параметрів, зокрема, прискорення передачі у разі надмірного знищення пакетів.

Висновки

У цьому розділі було сформовано концептуальну модель системи передачі мовних пакетів. Визначено мету моделювання, а також описано складові елементи системи, їхні взаємодії та ключові процеси. Зазначено вхідні та вихідні параметри моделі, а також встановлено обмеження, які забезпечують коректність і стабільність роботи системи.

Було розроблено деталізовану візуалізацію концептуальної моделі, яка ілюструє послідовність етапів обробки пакетів у системі та механізм активації прискорення. Це забезпечує чітке розуміння логіки функціонування системи та закладає основу для подальшого математичного моделювання та експериментальних досліджень.

РОЗДІЛ 2. ФОРМАЛІЗОВАНА МОДЕЛЬ

Формалізована модель системи передачі цифрової інформації — це детальний опис її елементів, процесів та взаємодій у термінах вибраного формалізму. Нижче наведено формалізований опис кожного елементу системи, обраного у завданні формалізму, з обґрунтуванням числових параметрів та розрахункових формул.


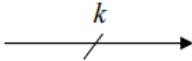
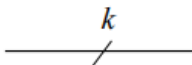

2.1 Загальні положення формалізму Мереж Петрі

Мережа Петрі є орієнтованим дводольним графом, який має чотири базових елементи: вузли, або місця (places), переходи (transitions), дуги (arcs) і маркери (tokens). Вузли позначаються кружками і визначають стан, в якому може знаходитись мережа або її частина. Переходи-це активні елементи мережі, які позначають дії, виконувани під час спрацювання переходів. Для того щоб перехід міг спрацювати, необхідне виконання певних умов, які визначаються наявністю маркерів у вузлах мережі, з'єднаних з переходом. Якщо умови настання подій виконано, то вважають, що перехід збуджений. Переходи позначаються короткими вертикальними або горизонтальними лініями. Вузли та переходи з'єднуються орієнтованими ребрами (дугами). Два вузли або два переходи з'єднуватись дугами не можуть.

Функціонування мережі Петрі можна описати так: вузли як певні умови, а переходи - як події. Таким чином, стан мережі в кожний момент часу задається системою умов. Для зручності задання умов мережі Петрі вводяться маркери (фішки), які зображуються крапками всередині вузлів. Виникнення певної комбінації маркерів у вузлах приводить до настання деякої події, яка у свою чергу викликає зміну стану умов мережі. Стан маркування або стан мережі Петрі визначається сукупністю маркерів кожного окремого вузла

У таблиці 2.1 (що взята із зазначеного навчального посібника[3]) зображено елементи формалізації системи за допомогою мережі Петрі. Основними компонентами є позиції, переходи та дуги, які відображають етапи обробки пакетів у системі.

Таблиця 2.1. Елементи формального опису мережею Петрі.

Елемент	Графічне позначення	Зміст	Числові параметри
Позиція		Відтворює виконання умов для певних подій.	m – кількість маркерів, що вказує на виконання умови.
Вхідна дуга		З'єднує позицію та перехід. Слугує для визначення умов, які мають бути виконані для спрацьовування переходу.	k – кількість маркерів, що віднімаються з позиції при спрацьовуванні переходу.
Вихідна дуга		З'єднує перехід та позицію. Слугує для визначення умов, які є наслідком спрацьовування переходу.	k – кількість маркерів, що додаються до позиції при спрацьовуванні переходу.
Перехід		Зміна стану моделі при виконанні події за рахунок зміни свого стану та стану позицій, які є його вхідними та вихідними позиціями. Використовує параметри пріоритету та ймовірності запуску для вирішення конфлікту з іншими переходами	d – часова затримка в умовних одиницях часу, r – пріоритет переходу, b – ймовірність спрацьовування переходу

Позиції представляють стани системи, у яких перебувають пакети на різних етапах передачі, включаючи буферизацію перед каналами, передачу через кожен канал, вихід системи та знищення пакету.

Переходи моделюють події, що відбуваються під час переміщення пакета між позиціями, як-от надходження в буфер, передача через канали, знищення після перевищення допустимого часу.

Дуги з'єднують позиції та переходи, відображаючи логіку послідовності дій у системі, наприклад, передача пакета від буфера до каналу або від каналу до виходу системи.

2.2 Побудова формалізованої моделі

Для побудови формалізованої мережі Петрі для даного завдання розіб'ємо мережу на компоненти (ключові етапи):

Генерація Пакетів(рис.2.1):

На вхід системи регулярно надходять пакети даних із затримкою 6 ± 3 мс (розподіленим за нормальним законом), і кожен новий пакет запускає процес передачі через канали. Ця генерація є основою потоку даних у системі.

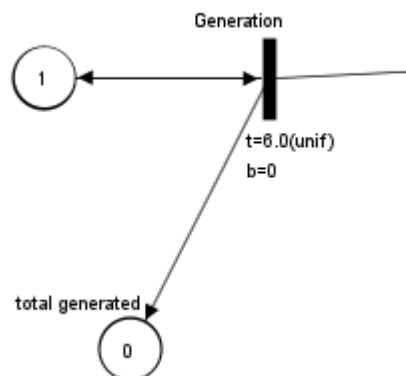


Рисунок 2.1 – Етап генерації пакетів

На схемі генерація представлена у вигляді потоку нових пакетів, що надходять у буфер для подальшої обробки з заданою інтенсивністю.

Буферизація перед кожним каналом(рис.2.2-2.3):

Перед кожним каналом пакети тимчасово зберігаються у буфері, що дозволяє оптимізувати потік даних і запобігає перевантаженню каналів. Буферизація забезпечує контроль черги пакетів, не допускаючи затримок.

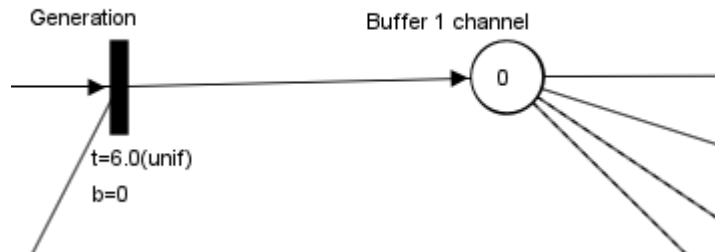


Рисунок 2.2 – Етап надходження пакетів в 1 канал

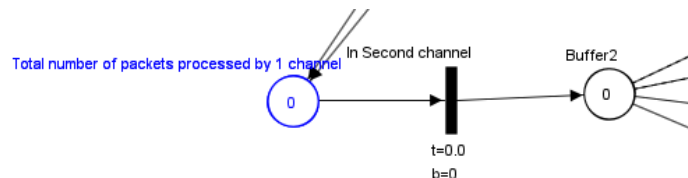


Рисунок 2.3 – Етап надходження пакетів в 2 канал

Буфери в каналах зображені як проміжні накопичувачі перед кожним каналом, куди пакети потрапляють перед передачею. Швидкість передачі по каналу становить 5 мс. Якщо у буфері накопичується черга із двох або більше пакетів, починають з'являтися пакети із затримкою (якщо в системі з прискоренням – від трьох пакетів). Це є сигналом до того, що система може не встигати обробляти дані вчасно.

Обробка пакетів(рис.2.4):

Кожен пакет проходить обробку через два послідовні канали, де на кожному з них витрачається певний час. Після проходження обробки пакет або виходить із системи, або знищується, якщо затримка перевищує встановлений ліміт.

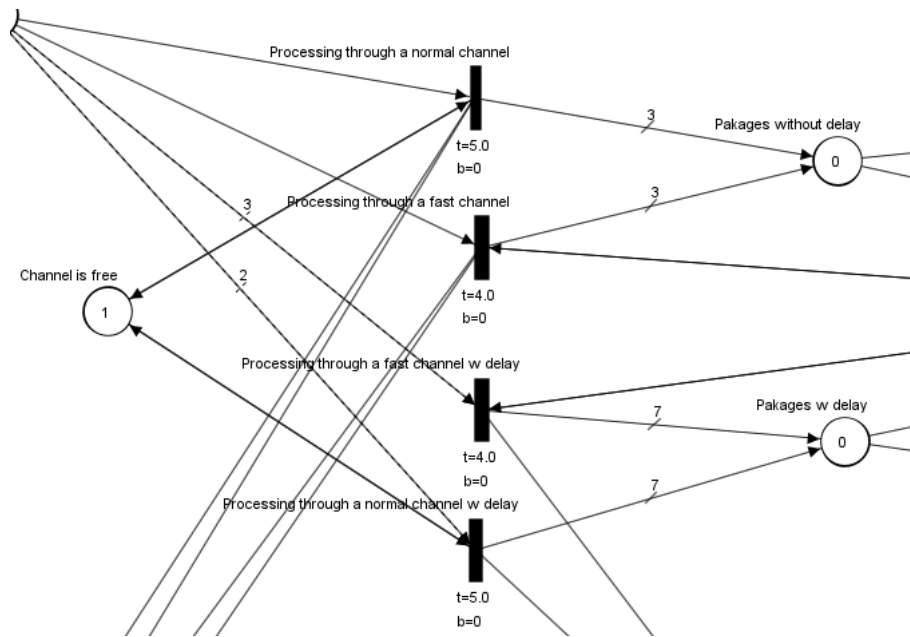


Рисунок 2.4 – Етап обробки пакетів

Загалом, система працює за таким принципом: обробка пакетів йде послідовним рухом через два канали, де відбувається передача даних і знищення пакету у кожному з каналів. Пакет надходить в перший канал з інтервалом 6 ± 3 мс, після чого потрапляє в буфер каналу, а потім потрапить на обробку. За один раз можна пропустити лише один пакет через канал, швидкість передачі якого 5 мс, тому необхідно використовувати позицію-обмежувач.

Основна задумка в тому, щоб використати лічильники, тобто надати пакету певну кількість очків. Якщо отриманий пакет із затримкою, то кількість його очків буде дорівнювати 7, а якщо пакет без затримки, то 3, в сумі 10. Далі відбувається порівняння цих очків між собою. Якщо після обробки пакету із затримкою залишаються очки (в даному випадку залишиться 7), то це означає, що треба підключити прискорення з 5 мс до 4 мс.

Варто зазначити, що усі 7 маркерів не спричинять 7 спрацювань переходу, тому що маркери із затримкою будуть лише тоді, коли є черга на обробку в 2 маркери для звичайного каналу та в 3 маркери для каналу із прискоренням, дуги з такими маркерами (з кратністю 2 та 3) є інформативними. У канал не йдуть 2 чи 3 пакети, тільки один, а інші очікують обробки у буфері. (рис. 2.5)

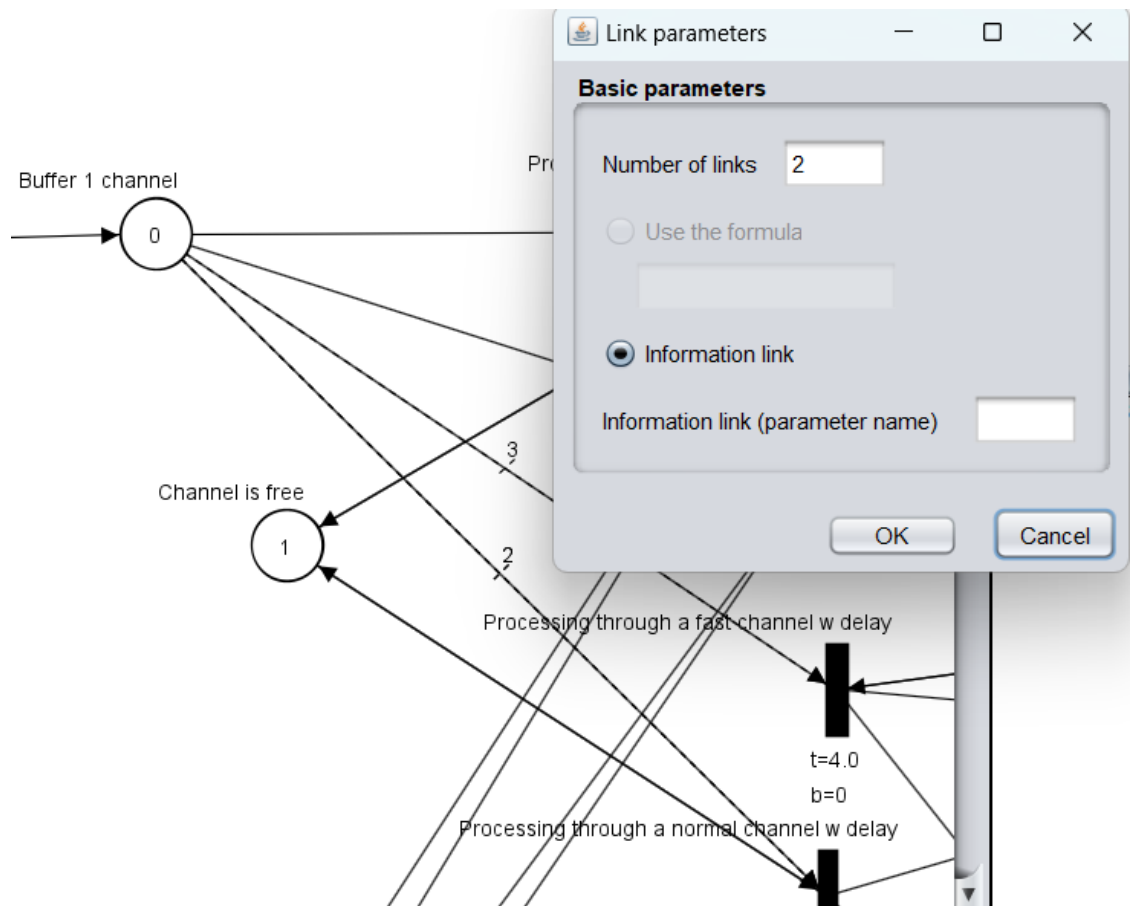


Рисунок 2.5 – Інформаційні дуги

Звідси і пояснення чому саме такі кратності дуг. Отже, пакет не може бути із затримкою, якщо звичайний канал обробки вільний, тому у нас завжди буде з чим порівнювати маркери.

Якщо ж пакети надходять без затримок, то система вже працює добре і не потрібно підключати прискорення. Таким чином, система обробки пакетів вирішує задачу динамічно в залежності від кількості пакетів та їх очок.

В результаті маємо: генерація, надходження пакету, його обробка, де наш канал вільний, пакет обробляється, йому надається 3 очки. Далі система очікує ще пакети, вимкнення прискорення може бути лише тоді, коли було піключення прискорення. Очікуємо поки будуть маркери для порівняння, щоб порівняти і зробити висновок чи потрібно підключити прискорення. Відбувається це, тому що стоїть найвищий пріоритет в позиції що порівнює очки. Виконується це до того часу, коли система не згенерує пакети частіше, ніж може обробити канал. Якщо канал обробив 2 пакети без затримки, у нас буде 6 очків, а потім відразу прийшло ще 2 пакети(це сигнал для передачі із

затримкою через звичайний канал), що запишуться у вигляді 7 очків, далі порівня, під час якого 1 маркер йде у підключення прискорення.

А для розв'язання конфлікту переходів у мережах Петрі використовуються різні пріоритети:

- для пакету з затримкою через прискорений канал пріоритет дорівнює 4;
- для пакету без затримки через прискорений канал пріоритет дорівнює 3;
- для пакету з затримкою через звичайний канал пріоритет дорівнює 2;
- для пакету без затримки через прискорений канал пріоритет дорівнює 1;

Підрахунок Знищених Пакетів(рис.2.6):

Пакети, які проходять через систему із затримкою понад 10 мс, вважаються застарілими та видаляються. Це підвищує загальну якість передачі, але вимагає контролю за рівнем знищення, який не повинен перевищувати 30%.

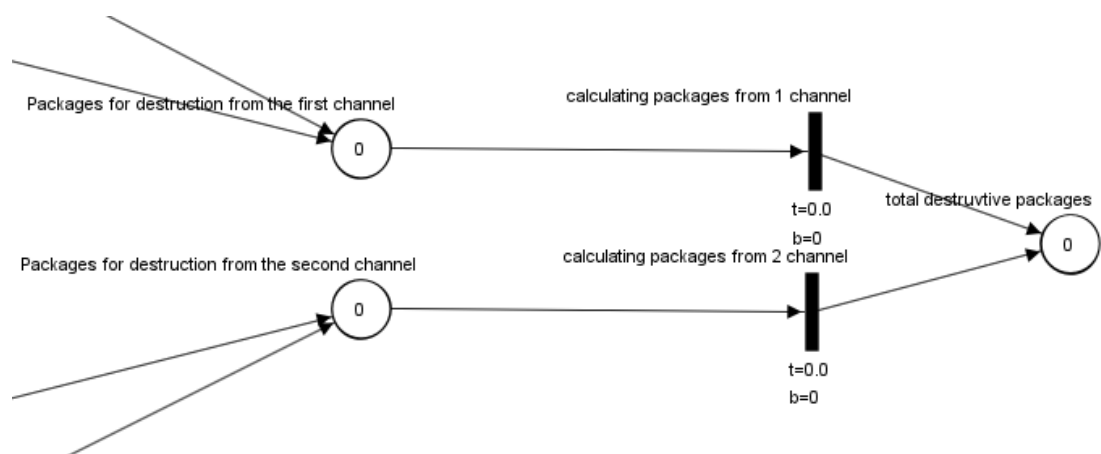


Рисунок 2.6 – Етап підрахунок знищених пакетів

На схемі показано місце для знищених пакетів, куди вони потрапляють після досягнення граничної затримки.

Прискорення/відключення прискорення(рис.2.7):

Якщо рівень знищення пакетів перевищує допустимий поріг у 30%, система активує прискорення, зменшуючи час проходження через кожен канал. При стабілізації рівня знищення система автоматично відключає прискорення для економії ресурсів

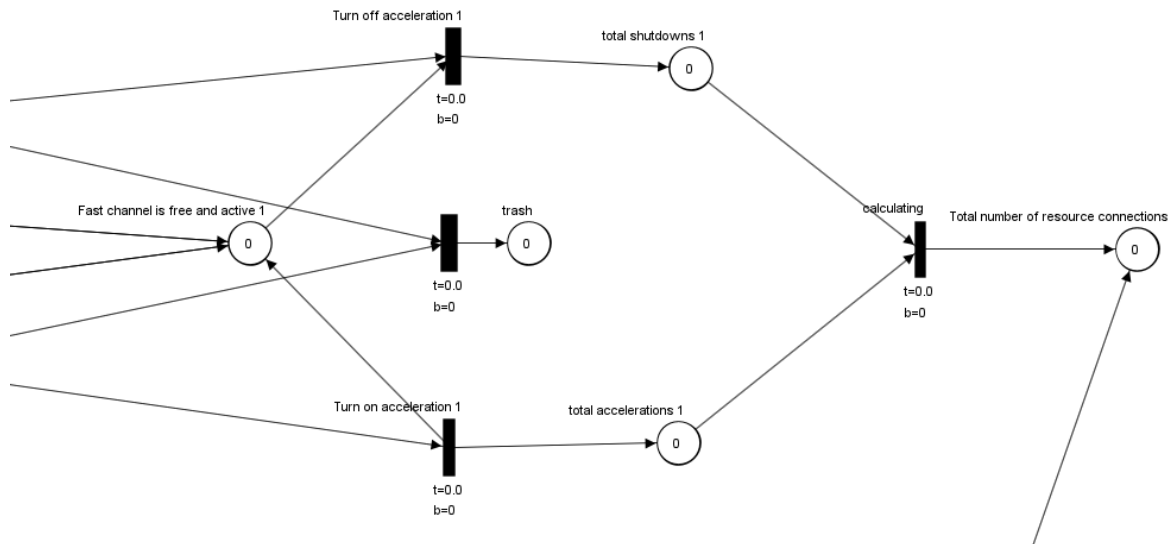


Рисунок 2.7 – Етап керування прискоренням

На схемі прискорення зображено як додатковий ресурс, який активується за умов перевищення допустимого рівня знищення і деактивується при поверненні до норми. Якщо різниця між очками затриманих і вчасних пакетів перевищує певний поріг, система активує механізм прискорення передачі, знижуючи затримку по каналу з 5 мс до 4 мс. Якщо в потік повертається нормальна швидкість надходження пакетів (зменшується кількість затриманих), і у системі з'являються пакети з «вчасними» маркерами, прискорення автоматично вимикається, так як немає необхідності у підвищеній швидкості. Це підвищує пропускну здатність і дозволяє швидше обробляти пакети.

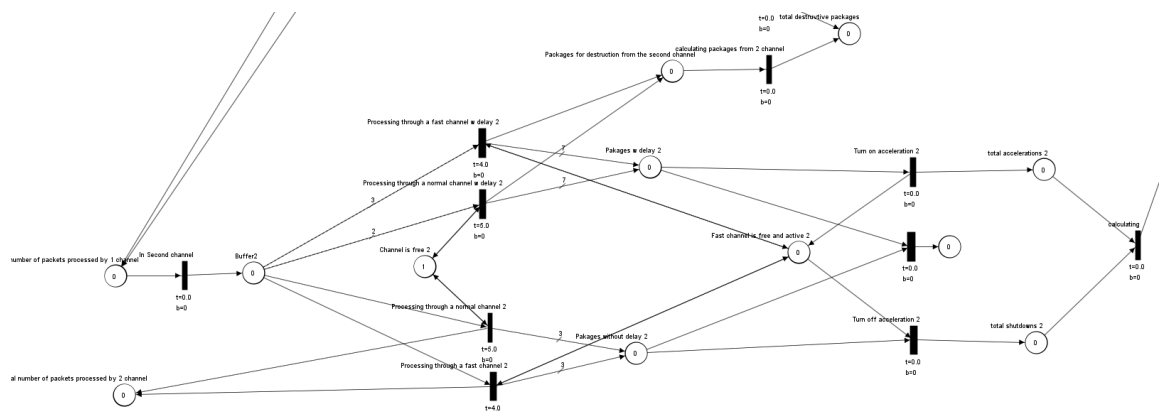


Рисунок 2.8 – Зображення схеми другого каналу

Другий канал повторює етапи першого(рис.2.8), від буферизації до обробки/знищення пакету, а також регуляцію прискорення та підрахунок усіх потрібних вихідних змінних

Висновки

Система автоматично адаптується до умов завдяки порівнянню очок затриманих і вчасних пакетів. Якщо є хоча б один «вчасний» маркер, то активація прискорення не потрібна. Таким чином, канали обробки працюють в звичайному режимі, доки надходження пакетів залишається оптимальним.

Також варто зазначити що два канали є послідовними, де пакети проходять послідовно через кожен, і можлива затримка. У такому випадку буфери дозволяють накопичувати черги та виникає потреба в прискоренні обробки при перевантаженні.

Числові параметри: Для коректного функціонування моделі важливо визначити числові параметри кожного елемента системи:

- Час передачі через канал: 5 мс, при активації ресурсу - 4 мс.
- Граничний час знищення пакету: 10 мс.
- Частка знищених пакетів для активації ресурсу: 30%.

Опис параметрів випадкових змінних: Для випадкових процесів, як-от надходження пакетів у систему, необхідно вказати формули для генерації випадкових чисел. Наприклад, інтервали надходження пакетів генеруються за рівномірним розподілом у діапазоні 6 ± 3 мс.

Вихідні характеристики моделі

Частота знищення пакетів:

Формула для розрахунку частоти знищених пакетів визначається як відношення кількості знищених пакетів N_{lost} до загальної кількості пакетів, що надійшли в систему N_{total}

$$F_{lost} = \frac{N_{lost}}{N_{total}}$$

Частота активації ресурсу:

Визначається як кількість спрацювань додаткового ресурсу N_{res} за весь час моделювання T_{mod}

$$F_{res} = \frac{N_{res}}{T_{mod}}$$

Для досягнення точності розрахунку вихідних параметрів (зокрема частоти знищення пакетів) необхідно проводити повторні вимірювання у ході моделювання. При значній кількості пакетів відхилення у значеннях буде мінімізоване, що забезпечить високу точність даних.

Таким чином, формалізована модель забезпечує опис усіх етапів проходження пакету через систему і дозволяє обчислити важливі характеристики для оцінки ефективності системи та прийняття рішень щодо оптимізації її параметрів.

РОЗДІЛ 3. АЛГОРИТМІЗАЦІЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ ІМІТАЦІЙНОЇ МОДЕЛІ СИСТЕМИ

3.1. Вступ та загальні положення

В цьому розділі розглянуто основні підходи до алгоритмізації та програмної реалізації імітаційної моделі системи передачі пакетів. Метою моделі є симуляція передачі цифрових пакетів через буферизовані канали, оптимізація обробки затриманих пакетів і застосування прискорення передачі. Програмна реалізація виконана мовою програмування Java[4], що дозволяє ефективно створювати об'єктно-орієнтовані моделі, реалізовувати події й часові обмеження. Нижче наведено основні етапи алгоритмізації моделі, програмні рішення для реалізації кожного етапу і процес верифікації моделі.

Для розробки моделі використана бібліотека PetriObjModelPaint. Це проект реалізації методу імітаційного моделювання за допомогою об'єктів Петрі. Необхідна техніка Петрі-об'єктного моделювання, основна концепція якої полягає у швидкому та гнучкому складанні коду моделі складної дискретно-подієвої системи з одночасним забезпеченням швидкого запуску симуляції. Опис поведінки моделі базується на стохастичній багатоканальній мережі Петрі, а композиція моделі ґрунтується на об'єктно-орієнтованій технології. Програмне забезпечення для імітаційного моделювання об'єктів Петрі забезпечує масштабований алгоритм моделювання, графічний редактор, коректне перетворення графічних зображень у модель, коректні результати моделювання. Графічний редактор допомагає впоратися з схильним до помилок процесом зв'язування елементів один з одним.

Програмне забезпечення складається з пакету PetriObjLib, що реалізує алгоритм моделювання об'єктів Петрі, та пакетів, що забезпечують графічне представлення мереж. Графічний редактор дозволяє побудувати мережу Петрі, зберегти її як метод Java та додати метод до класу NetLibrary. Програма підтримує відкриття мережі з файлу або відтворення мережі з Java-методу. Крім того, передбачена анімація симуляції для перевірки правильності створеної мережі Петрі. Слід зазначити, що буде згенеровано виключення,

якщо мережа Петрі складається з переходів, які не мають вхідних або вихідних місць. Після успішного збереження метод можна використовувати для створення Петрі-об'єктів. Коли список об'єктів Петрі підготовлено і зв'язки між

3.2. Структура класів і об'єктно-орієнтований підхід

Для кращого розуміння бібліотеки, опис основних класів бібліотеки наведений у таблиці 3.1

Таблиця 3.1 – Опис основних класів бібліотеки PetriObjModelPaint

Назва класу	Опис
ArcIn	Моделює вхідну дугу між місцем (Place) і переходом (Transition) у мережі Петрі.
ArcOut	Моделює вихідну дугу між переходом (Transition) і місцем (Place) у мережі Петрі. Він задає параметри дуги, такі як кратність (множинність) і зв'язки між переходом та місцем, а також забезпечує методи для зміни параметрів і клонування дуги.
ExceptionInvalid-NetStructure та ExceptionInvalid-TimeDelay	Класи обробляють виключення про помилку у структурі мережі та некоректну затримку часу
FunRand	Надає методи для генерації випадкових чисел відповідно до різних ймовірнісних розподілів.
PetriNet	Клас відповідає за створення та управління мережею Петрі.
PetriObjModel	Клас представляє імітаційну модель, що складається з набору об'єктів мережі Петрі (PetriSim).
PetriP	Клас представляє місце в мережі Петрі, зберігаючи його характеристики та стан.

PetriSim	Клас PetriSim моделює поведінку одного об'єкта мережі Петри, реалізуючи її динаміку та керуючи переходами.
PetriT	Клас для створення переходів у мережі Петрі
StateTime	Клас, який зберігає і управляє поточним часом та часом моделювання

Відповідно до формалізованої моделі, наступним кроком стане побудова мережі для нашої задачі. Побудована мережа наочно відображає ключові компоненти системи — канали, буфери, механізм прискорення та логіку обробки пакетів у різних станах. На рисунку 3.1 можна розглянути структуру мережі, взаємозв'язки між її компонентами та механізми, які забезпечують передачу пакетів і управління чергами в буферах. Така графічна схема мережі служить важливим інструментом для візуалізації роботи моделі та подальшого її аналізу й оптимізації для нашої задачі, опис обробки пакетів детальніше у розділі 2.

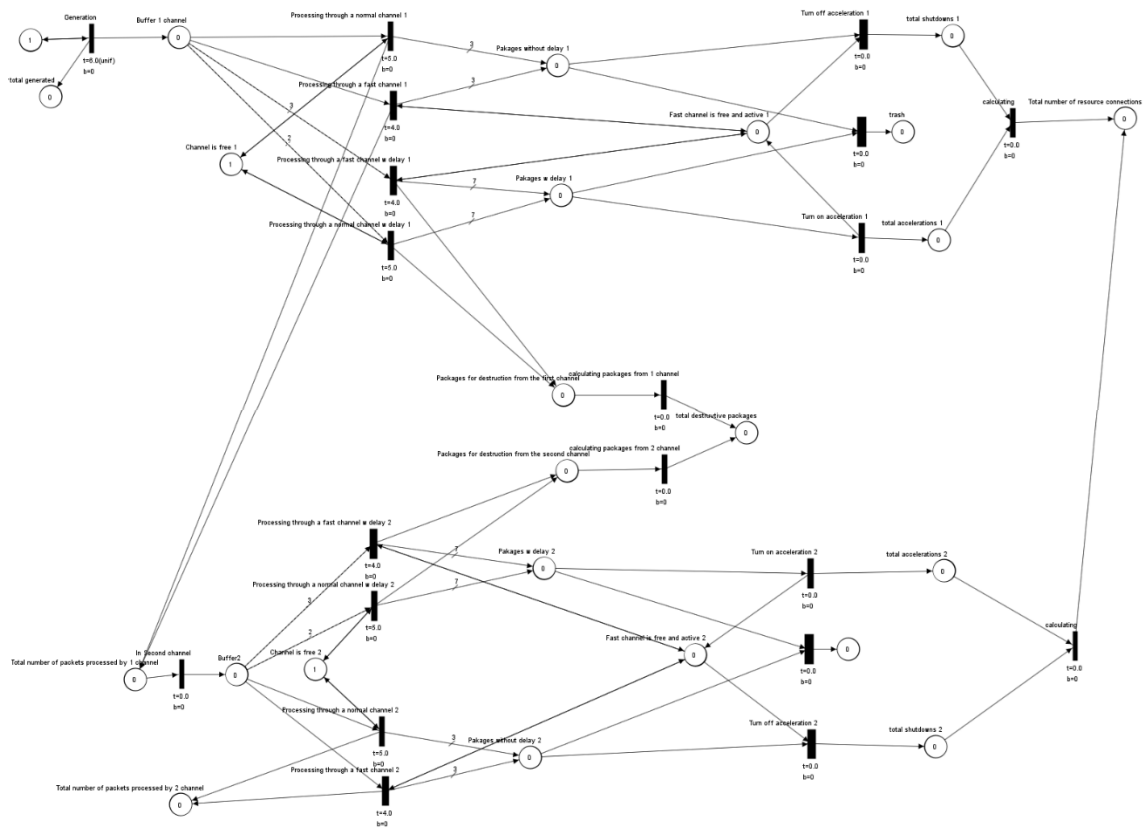
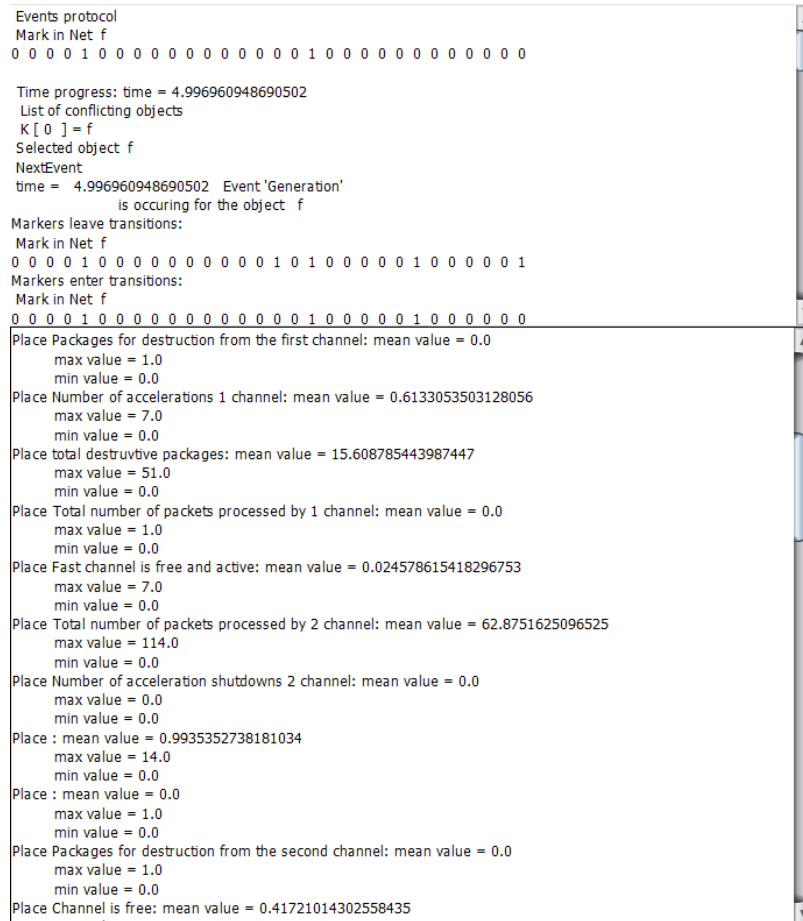


Рисунок 3.1 – Мережа Петрі побудована у PetriObjectModelPaint

Після запуску програми з реалізованою мережею ми отримаємо готовий звіт, який включає в себе детальну статистику по всіх об'єктах моделі, яку можна побачити на рисунку 3.2. Цей звіт містить інформацію про кожен етап обробки пакетів: кількість переданих і затриманих пакетів, обсяг буферів, кількість знищених пакетів, а також роботу прискорення.



```

Events protocol
Mark in Net f
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0

Time progress: time = 4.996960948690502
List of conflicting objects
K[ 0 ] = f
Selected object f
NextEvent
time = 4.996960948690502 Event 'Generation'
is occurring for the object f
Markers leave transitions:
Mark in Net f
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 1 0 0 0 0 0 1
Markers enter transitions:
Mark in Net f
0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0
Place Packages for destruction from the first channel: mean value = 0.0
max value = 1.0
min value = 0.0
Place Number of accelerations 1 channel: mean value = 0.6133053503128056
max value = 7.0
min value = 0.0
Place total destructive packages: mean value = 15.608785443987447
max value = 51.0
min value = 0.0
Place Total number of packets processed by 1 channel: mean value = 0.0
max value = 1.0
min value = 0.0
Place Fast channel is free and active: mean value = 0.024578615418296753
max value = 7.0
min value = 0.0
Place Total number of packets processed by 2 channel: mean value = 62.8751625096525
max value = 114.0
min value = 0.0
Place Number of acceleration shutdowns 2 channel: mean value = 0.0
max value = 0.0
min value = 0.0
Place : mean value = 0.9935352738181034
max value = 14.0
min value = 0.0
Place : mean value = 0.0
max value = 1.0
min value = 0.0
Place Packages for destruction from the second channel: mean value = 0.0
max value = 1.0
min value = 0.0
Place Channel is free: mean value = 0.41721014302558435
max value = 1.0
min value = 0.0

```

Рисунок 3.2 – Статистика після виконання моделі

3.3. Верифікація моделі:

Верифікація моделі є ключовим етапом перевірки коректності та надійності імітаційної системи. Цей процес дозволяє оцінити, наскільки модель відповідає початковим вимогам і специфікаціям, а також підтвердити її здатність адекватно відображати реальні умови роботи системи передачі даних. Основна мета верифікації — виявити потенційні помилки в логіці

моделі, забезпечити стабільність її роботи при змінних параметрах і переконатися, що вона коректно реагує на заплановані події.

Верифікація проводилася шляхом тестування моделі при різних значеннях параметрів, які впливають на швидкість надходження пакетів, їх затримку, а також частоту активації прискорення передачі. Таблиці 3.2 та 3.3 представляють собою детальні результати верифікації, де були змінені та досліджені вхідні змінні для порівняння з базовою моделлю, побудованою на початкових значеннях.

Кожен тестовий набір параметрів запускався 20 раз для отримання більш точних даних і усереднення результатів. Це дозволило оцінити поведінку системи у стабільних умовах і уникнути впливу випадкових факторів.

Таблиця 3.2 – Результати верифікації моделі при зміні часу моделювання (Час надходження = 6 мс, Середнє відхилення = 3 мс, Час передачі по каналу = 5 мс (прискорений – 4 мс), Відсоток передачі = 30%)

	T_{mod}	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	100	16.085	12.925	1.37	1.705	0.0137	0.01705
2	500	82.17	68.12	12.14	5.88	0.02428	0.01176
3	2500	415.205	336.075	77.245	9.39	0.030898	0.003756
4	5000	829.38	669.595	157.84	9.02	0.031568	0.001804
5	10000	1659.495	1335.77	321.745	11.03	0.032175	0.001103

Таблиця 3.3 – Результати верифікації моделі при зміні часу надходження (Час моделювання = 1000, Середнє відхилення = 3 мс, Час передачі по каналу = 5 мс (прискорений – 4 мс), Відсоток передачі = 30%)

	T_1 (час надх.)	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	3	307.87	149.19	154.83	514.66	0.154825	0.51466
2	4	241.24	119.53	118.47	260.09	0.118465	0.26009
3	5	197.63	118.7	76.37	85.145	0.07637	0.085145
4	6	165.57	135.8	27.89	9.365	0.02789	0.009365
5	7	142.51	135.59	5.415	0.63	0.005415	0.00063
6	8	124.86	122.48	1.04	0.075	0.00104	0.000075

Згідно з результатами моделювання, зазначеними в таблиці вище, швидкість обробки елементу в 5 мс виявилася досить оптимальною для вхідного потоку з нормально розподіленим інтервалом надходження 6 ± 3 мс. При таких параметрах середня частота підключення ресурсу прискорення становила лише 0.0117. Це вказує на те, що ресурс прискорення при стандартних умовах є малозатребуваним, оскільки швидкість надходження пакетів не призводить до надмірного накопичення черги.

При зменшенні середнього інтервалу надходження пакетів частота підключення ресурсу та частота знищення пакетів поступово збільшуються. Це пов'язано з тим, що при зменшенні інтервалу надходження система не встигає обробляти всі пакети вчасно, що призводить до накопичення черги та необхідності прискорення.

Виконаємо верифікацію при зміні неприпустимого відсотку, при якому прискорюється передача. У таблиці 3.4 наведено результати такої верифікації при різних значеннях

Таблиця 3.4 – Зміни в моделі при різних параметрах допустимого відсотку передачі

	A_1 (відсоток знищення)	$N_{generate}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	10	164.89	143.3	19.825	45.18	0.0198	0.0452
2	20	166.24	138.14	26.125	23.855	0.0261	0.0239
3	30	165.56	134.96	28.555	8.715	0.0286	0.0087
4	40	166.12	133.29	30.91	3.49	0.0309	0.0035
5	50	165.88	133.49	30.355	1.835	0.0304	0.0018

Зі зміною цього параметра помітно, що частота підключення ресурсу зростає при зниженні допустимого відсотка знищення.

Це зумовлено тим, що система намагається активніше уникати знищення пакетів, підвищуючи швидкість обробки і, як наслідок, частіше активує прискорення. Однак частота знищення пакетів залишається

приблизно на одному рівні, що свідчить про здатність системи адаптуватися до заданих параметрів моделі.

Таким чином, при зниженні допустимого рівня знищення система автоматично збільшує частоту підключення ресурсу, намагаючись зберегти пакетний потік в межах допустимих значень. Це підтверджує коректність налаштування моделі, оскільки вона коректно реагує на зміну критичних параметрів і підтримує заданий рівень якості передачі даних.

Виконаємо верифікацію при зміні часу прискореної передачі. У таблиці 3.5 наведено результати верифікації при різних значеннях часу обробки в прискорених каналах

Таблиця 3.5 – Зміни в моделі при різних параметрах прискореного часу

	T_3 (Прискорений час)	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	4	165.16	135.40	27.84	8.19	0.02784	0.00819
2	3	164.895	135.88	27.14	7.225	0.02714	0.007225
3	2	165.695	135.93	27.84	8.145	0.02784	0.008145

Зі зміною цього параметра бачимо, що частота підключення ресурсу та частота знищення ніяк не змінилася.

Це можна пояснити тим, що система вже працює в оптимальному режимі при заданих вхідних параметрах, зменшення цього параметру може не призводити до покращення роботи системи через низький рівень завантаження прискореного каналу.

Висновки

Проведена верифікація показала, що модель відповідає початковим вимогам у базових умовах і коректно реагує на зміни у швидкості надходження пакетів та допустимих рівнях знищення. Базова модель, яка працює з інтервалом надходження пакетів 6 ± 3 мс і часом обробки 5 мс, демонструє стабільну продуктивність, і потреба у прискоренні виникає лише при значному збільшенні потоку вхідних даних.

Однак при зміні параметрів генерації пакетів на значення, наприклад, 3 ± 3 мс система починає частіше активувати прискорення, що вказує на збільшення навантаження на буфер і необхідність оптимізації обробки, що є правильним рішенням з точки логіки. В цілому, частота знищення пакетів утримується на низькому рівні навіть при зміні критичних параметрів, що підтверджує адекватність налаштувань моделі.

Результати верифікації свідчать про те, що модель виконана відповідно до початкових вимог і забезпечує достатню ефективність при стандартних умовах.

РОЗДІЛ 4. ЕКСПЕРМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МОДЕЛІ

4.1 Постановка задачі та обґрунтування параметрів експерименту

Метою експериментального дослідження є проведення регресійного аналізу над параметрами швидкості обробки, тому що саме від цього залежить те, скільки пакетів буде оброблено, знищено та як часто треба підключати прискорення. Потрібно проаналізувати залежності ключових параметрів, таких як частота знищення пакетів і активація ресурсу, від часу обробки в звичайному і прискореному режимах.

Однією з основних вимог для експерименту було збереження незмінності часу надходження пакетів у систему, оскільки зміна цього параметра могла б вплинути на поведінку моделі в напрямку, який не є релевантним до умов реального використання.

4.2 Час симуляції та визначення перехідного періоду

Для початку, потрібно визначити час симуляції для експериментів та кількість прогонів.

З метою оцінки перехідного періоду було проведено низку пробних експериментів, під час яких фіксувалася частота підключення ресурсу кожні 500 умовних одиниць часу, аналіз даних(приклад одного з чотирьох експериментів) із таблиці 4.1 дозволяє оцінити, як змінюються ключові метрики системи в залежності від часу симуляції.

Таблиця 4.1 – Результати експериментів моделі при зміні часу моделювання

	T_{mod}	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення пакетів)	F_{res} (Частота підключення ресурсу)
1	500	81.5	68.1	11.85	3.2	0.0237	0.0064
2	1000	165.35	130.95	32.45	11.95	0.03245	0.01195
3	1500	246	201.6	42.45	6.7	0.0283	0.004467
4	2000	327.8	276.55	49.35	7.7	0.024675	0.00385
5	2500	410.85	340.15	69	7.7	0.0276	0.00308
6	3000	497.95	406.35	89.65	3	0.029883	0.001
7	3500	576.45	478.1	96.8	10.6	0.027657	0.003029
8	4000	661.35	537.8	121.55	14.8	0.030388	0.0037
9	4500	748.4	604	142.7	9.25	0.031711	0.002056
10	5000	829.85	674.05	153.8	10.45	0.03076	0.00209
11	5500	913.2	733.95	177.2	5.85	0.032218	0.001064

12	6000	1000.4	795.3	203	13.5	0.033833	0.00225
13	6500	1079.85	866.95	210.8	10.45	0.032431	0.001608
14	7000	1161.8	939.7	220.45	8.4	0.031493	0.0012
15	7500	1238.65	1005.75	230.6	6.45	0.030747	0.00086
16	8000	1322.6	1082.75	237.75	5.7	0.029719	0.000713
17	8500	1409.5	1133.2	274.3	17.55	0.032271	0.002065
18	9000	1495.4	1196.1	297.45	11.1	0.03305	0.001233
19	9500	1584.4	1257.05	325.6	13.5	0.034274	0.001421
20	10000	1653.65	1335.05	316.7	12.55	0.03167	0.001255

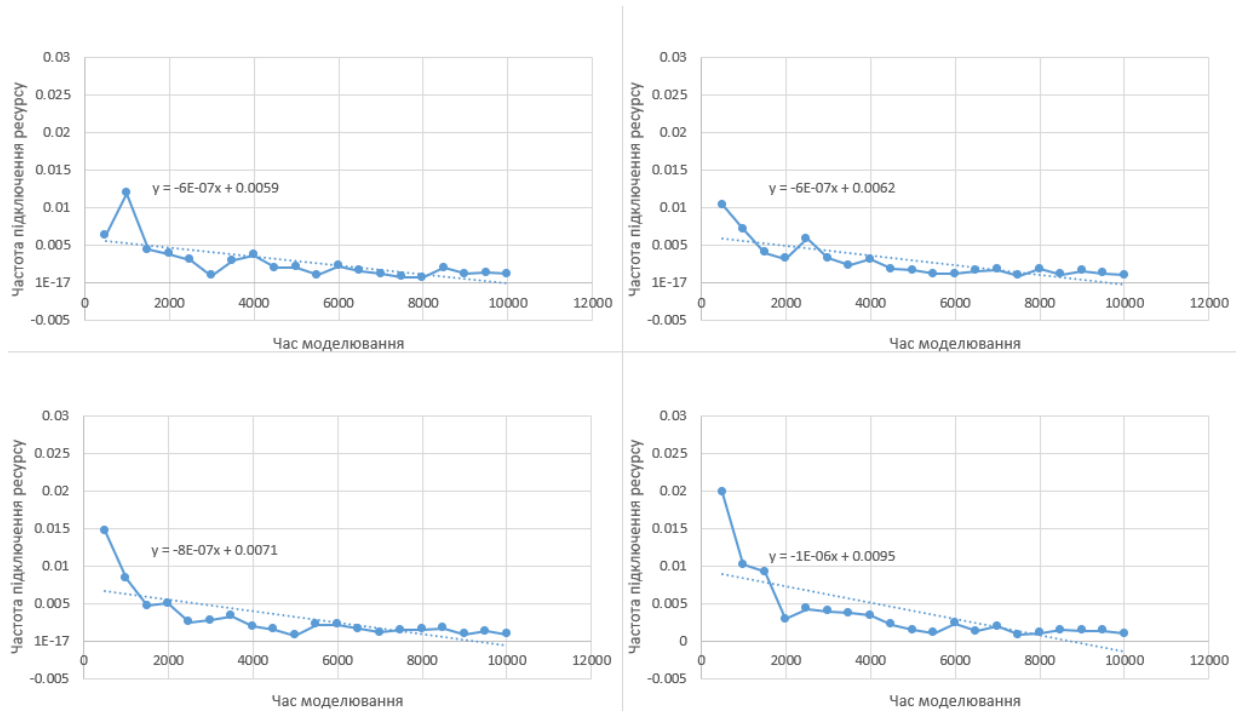


Рисунок 4.1 – Результати експериментів моделі при зміні часу моделювання

Результати цих експериментів, представлені на рисунку 4.2, показали, що після 6500 умовних одиниць часу частота активації ресурсу є досить стабільною, що свідчить про завершення перехідного періоду.

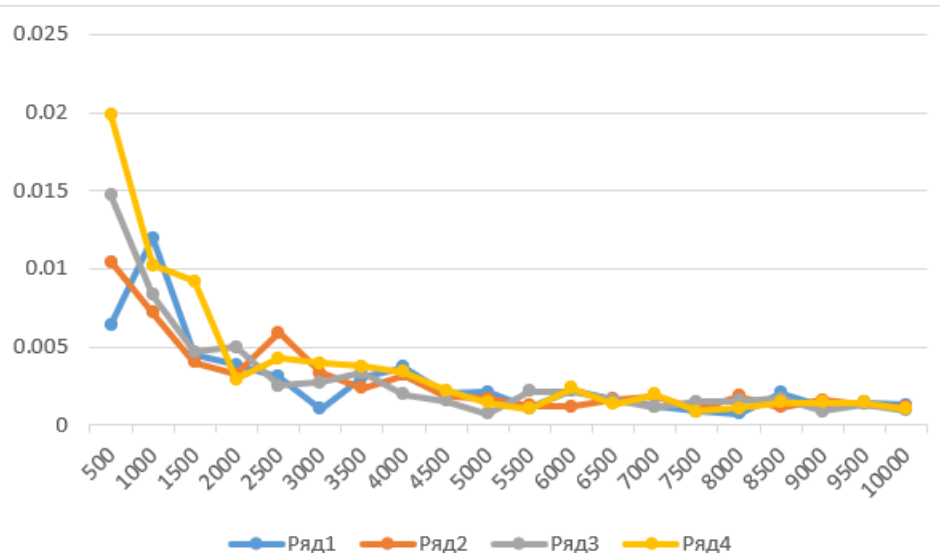


Рисунок 4.2 – Порівняння результатів експерименту

У такій ситуації можна обрати для експериментальних запусків час симуляції у 6500 умовних одиниць, щоб уникнути впливу початкових коливань на результати. Чим більше час симуляції, тим більшу вибірку можна отримати, але після певного моменту це лише збільшує обсяг обчислень без значного впливу на результати.

4.3. Кількість прогонів і забезпечення точності результатів

Для забезпечення точності результатів моделювання було прийнято рішення виконати **20 прогонів** моделі. Під час пробних запусків було встановлено, що середні значення основних метрик, таких як частота знищення пакетів і частота підключення ресурсу, стабілізуються після 15 прогонів. Додавання ще кількох прогонів до цієї кількості гарантує мінімізацію впливу випадкових коливань на результати.

Кожен з експериментів, таким чином, виконувався 20 разів, а отримані результати усереднювалися, що дозволяло знизити вплив випадкових відхилень та досягти високої точності.

4.3.1. Вибір параметрів для дослідження

Для оцінки впливу параметрів моделі проводилися експерименти зі змінними параметрами часу обробки в звичайному і прискореному режимах.

Базові налаштування системи включали час обробки в звичайному режимі на рівні 5 мс та 4 мс у прискореному режимі. Основна мета дослідження полягала у визначенні умов, за яких частота знищення пакетів і активація ресурсу прискорення будуть мінімальними, що вказує на оптимальний режим роботи системи.

4.3.2 Проведення експериментів і аналіз залежностей

З метою визначення впливу часу обробки було проведено серію експериментів, під час яких змінювався час обробки пакетів у звичайному та прискореному каналах. Результати досліджень, представлені на таблицях 4.2 та 4.3, показали, що зі збільшенням швидкості обробки у звичайному каналі збільшується частота знищення пакетів, оскільки система не встигає обробляти більше пакетів за умовну одиницю часу.

Таблиця 4.2 – Результати експериментів моделі при зміні часу обробки

	T_2 (Звичайний час)	T_3 (Прискорений час)	$N_{generate}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	1	0	332.9	331.75	0.7	0	0.0007	0
2	2	1	330.45	327.85	1.85	0	0.00185	0
3	3	2	331.9	323.8	7.05	0.2	0.00705	0.0002
4	4	3	331.55	310.8	19.2	0.15	0.0192	0.00015
5	5	4	331.2	270.35	58.9	6.85	0.0589	0.00685
6	6	5	331.8	202.7	126.25	115.05	0.12625	0.11505
7	7	6	330.7	166.15	161.3	276.9	0.1613	0.2769
8	8	7	330.05	161.45	165.55	416.2	0.16555	0.4162
9	9	8	332.45	161.8	166.95	507.1	0.16695	0.5071
10	10	9	332.75	161.2	167.5	551.75	0.1675	0.55175

Зручніше результати досліджень представлені на рис 4.2-4.3, де можна побачити збільшення частоти знищення пакетів

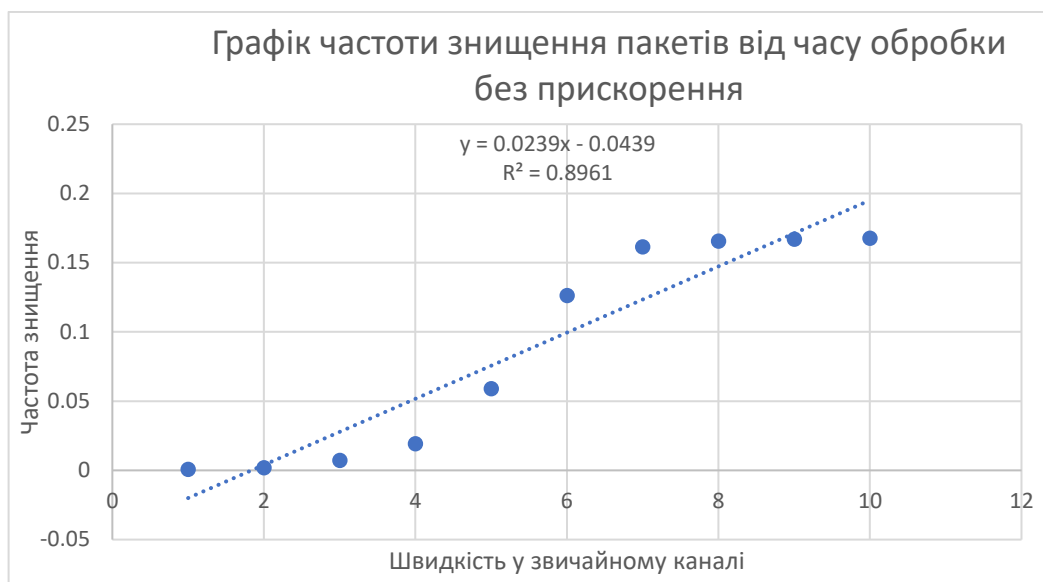


Рисунок 4.2 - Результати експериментів моделі при зміні часу обробки без прискорення



Рисунок 4.3 - Результати експериментів моделі при зміні часу обробки з прискоренням

Зазначу, що при зміні швидкості обробки від 4 до 10 мс було помічено експоненційне зростання частоти активацій ресурсу прискорення, що підтверджує високу залежність цього параметра від інтенсивності обробки. Загалом експериментальна серія продемонструвала значущість швидкості

обробки для зменшення частоти знищення пакетів та зниження необхідності в додаткових ресурсах.

4.4 Результати регресійного аналізу

Для глибшого розуміння впливу обраних параметрів на продуктивність системи було проведено регресійний аналіз[5], у якому в якості вхідних змінних були використані час обробки у звичайному (X_0) і прискореному режимі (X_1), а вихідна змінна — частота знищення пакетів (Y_1). Розрахунки виконувалися з використанням Microsoft Excel, що дозволило провести лінійний багатofакторний аналіз.

4.4.1 Постановка та результати регресійного аналізу

Визначивши вихідні дані для регресійного аналізу та параметри аналізу. Запуск регресійного аналізу дозволив отримати модель залежності частоти знищення пакетів від часу обробки в різних режимах. На рисунку 4.4 представлено результати регресійного аналізу, з якого видно, що коефіцієнт детермінації (R^2) має значення 0.891229, що близьке до 1. Це свідчить про високу точність моделі та відповідність залежностей параметрів досліджуваній системі.

ВИСНОВОК ПІДСУМКІВ								
Регресійна статистика								
Множинний R	0.944049							
R-квадрат	0.891229							
Нормований R-квадрат	0.87569							
Стандартна помилка	0.026207							
Спостереження	9							
Дисперсійний аналіз								
	df	SS	MS	F	Значимість F			
Регресия	1	0.039393	0.039393	57.35534402	0.00012912			
Залишок	7	0.004808	0.000687					
Разом	8	0.044201						
	Коефіцієнти	Стандартна помилка	t-статистика	P-Значення	Нижні 95%	Верхні 95%	Нижні 95.0%	Верхні 95%
Y-перетин	-0.05657	0.0221	-2.55962	0.037572244	-0.108826066	-0.00431	-0.10883	-0.00431
1	0.025623	0.003383	7.573331	0.00012912	0.01762295	0.033624	0.017623	0.033624

Рисунок 4.4 - Результати регресійного аналізу

Критерій Фішера (F) дозволяє порівнювати величини вибірових дисперсій двох незалежних вибірок. Він підтвердив адекватність моделі:

отримане значення (0.0001) є меншим за 0.05, що свідчить про статистичну значущість обраних параметрів. Водночас для базової швидкості обробки окремі коефіцієнти виявилися незначними, що може вказувати на низьку індивідуальну значущість кожного з них при високій сумарній ефективності моделі. Це означає, що ми не можемо пояснити вплив кожної змінної окремо, але можемо оцінити їхній комбінований вплив на частоту знищення пакетів.

4.4.2 Візуалізація результатів та рівняння регресії

На рисунку 4.5 показано рівняння регресії та графік отриманих результатів, які підтверджують адекватність обраної моделі та її здатність аргументувати залежність частоти знищення пакетів від часу обробки в каналах. Високе значення R^2 також підтверджує, що обрані параметри суттєво впливають на продуктивність моделі.

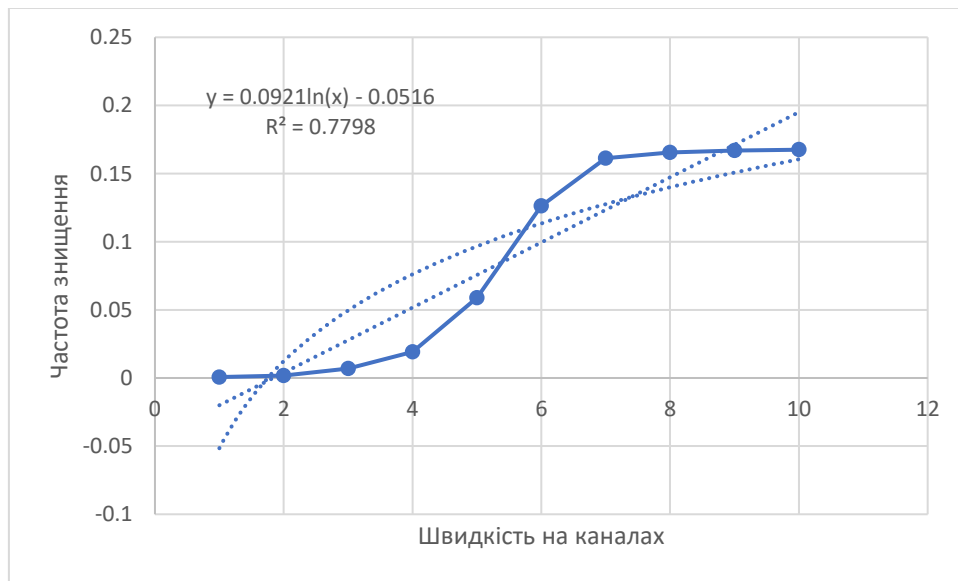


Рисунок 4.5 – Графік отриманих результатів та R^2

4.5. Код для проведення експериментального дослідження

Для автоматизації процесу експериментального дослідження було розроблено програмний код на Java, що дозволяє запускати модель з різними параметрами і здійснювати збір та обробку статистичних даних. Програмний код забезпечує повторювані запуски моделі з фіксованими параметрами, а також включає функції для обчислення середніх показників та зниження впливу випадкових факторів. Детальніше код можна знайти у Додатку А.

Висновки

Експериментальне дослідження продемонструвало, що швидкість обробки пакетів є ключовим фактором, який впливає на частоту знищення пакетів та частоту активації ресурсу прискорення. Зменшення часу обробки в звичайному та прискореному режимах дозволяє досягти стабільної роботи системи з мінімальною частотою знищення пакетів. Проведений регресійний аналіз підтвердив статистичну значущість моделі і дозволив оцінити комбінований ефект змінних на продуктивність системи, що дає підстави для подальшої оптимізації параметрів моделі.

РОЗДІЛ 5. ІНТЕРПРЕТАЦІЯ РЕЗУЛЬТАТІВ МОДЕЛЮВАННЯ СИСТЕМИ, ФОРМУЛЮВАННЯ ВИСНОВКІВ ТА ПРОПОЗИЦІЙ

На основі проведених експериментів і аналізу результатів можна зробити низку висновків щодо продуктивності та ефективності моделі системи передачі мовних пакетів. Основними метриками для оцінювання стали частота знищення пакетів та частота підключення ресурсу прискорення.

5.1 Узагальнення результатів моделювання

Базова модель, з параметрами часу надходження пакетів 6 ± 3 мс і часу передачі 5 мс (звичайний режим) та 4 мс (прискорений режим), показала стабільну роботу за стандартних умов. Частота підключення ресурсу становила лише 0.0117 при базових параметрах, що свідчить про рідкісну потребу у прискоренні. Частота знищення пакетів залишалася на прийнятному рівні(0.0305), що вказує на достатню ефективність системи для заданих умов.

Проведені експерименти продемонстрували значущість часу обробки на загальну продуктивність системи. При збільшенні часу обробки понад 5 мс частота знищення пакетів почала стрімко зростати, що свідчить про накопичення черги в системі. Аналіз також підтвердив, що прискорення (зменшення часу обробки до 4 мс) є ефективним засобом для стабілізації системи у випадках перевантаження.

Регресійний аналіз показав високу кореляцію між часом обробки і частотою знищення пакетів. Значення коефіцієнта детермінації ($R^2 = 0.891$) підтвердило точність моделі і дозволило оцінити комбінований вплив часу обробки в звичайному та прискореному режимах на результати роботи системи. Критерій Фішера ($F = 0.0001$) також вказав на статистичну значущість моделі.

5.2 Пропозиції щодо вдосконалення системи

На основі отриманих результатів розроблено низку пропозицій для покращення продуктивності системи:

- 1. Оптимізація алгоритму активації прискорення*

Налаштування порогу активації ресурсу слід переглянути. Рекомендується збільшити точність визначення моментів перевантаження для уникнення надмірного використання ресурсу.

2. Удосконалення буферизації

Розглянути можливість збільшення розміру буферів перед каналами або введення механізму пріоритетної обробки пакетів.

3. Дослідження інтенсивності надходження пакетів

Метою цього дослідження є аналіз впливу інтенсивності надходження пакетів на роботу системи. Змінюючи параметри, що визначають середній час між надходженням пакетів, можна оцінити, як система справляється з навантаженням і як змінюються ключові метрики, такі як частота знищення пакетів та підключення додаткових ресурсів.

5.3 Оцінка ефективності запропонованих змін

З попереднього розділу, очевидно, що при зменшенні часу обробки в каналах зменшеться й частота, тому для оцінки ефективності запропонованих змін було проведено пробний запуск моделі з новим параметром (табл. 5.1). Зокрема:

- Змінено механізм активації прискорення при частоті знищення, що перевищує 40%. Але варто зазначити, що зміна цього параметру вплине на якість переданих повідомлень і вони стануть нерозбірливими.

Таблиця 5.1 – Результати експериментів зміненої моделі

	T_{mod}	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення пакетів)	F_{res} (Частота підключенн я ресурсу)
Базова	1000	165.52	135.3	28.32	8.255	0.02832	0.008255
Змінена	1000	166.155	133.45	30.715	3.395	0.030715	0.003395

Результати експериментів показали:

- Частота знищення пакетів трохи збільшилась, але залишилася майже незмінною
- Частота підключення ресурсу при цьому зменшилась на 41%

А от пропозиція зменшувати час обробки на першому каналі, водночас компенсуючи це збільшенням часу обробки на іншому каналі не є хорошою.

Було проведено пробний запуск моделі з новим параметром (табл 5.2), зокрема:

- Час обробки без прискорення на першому каналі = 4 мс, з прискоренням = 3 мс.
- Час обробки без прискорення на другому каналі = 6 мс, з прискоренням = 5 мс.

Таблиця 5.2 – Результати експериментів зміненої моделі

	T_{mod}	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення пакетів)	F_{res} (Частота підключенн я ресурсу)
Базова	1000	165.52	135.3	28.32	8.255	0.02832	0.008255
Змінена	1000	165.91	110.385	53.045	12.185	0.053045	0.012185

Проведемо невелике дослідження при якому змінимо час надходження пакетів в систему:

Таблиця 5.3 – Результати верифікації моделі при зміні часу надходження (Час моделювання = 2000, Середнє відхилення = 3 мс, Час передачі по каналу = 5 мс (прискорений – 4 мс), Відсоток передачі = 30%)

	T_1 (час надх.)	$N_{generated}$	N_{proc}	N_{lost}	N_{res}	F_{lost} (Частота знищення)	F_{res} (Частота підключення ресурсу)
1	3	614.855	299.705	311.34	1040.525	0.31134	1.040525
2	4	484.04	238.385	242.335	534.25	0.242335	0.53425
3	5	395.975	233.15	160.065	164.215	0.160065	0.164215
4	6	331.885	268.94	61.025	9.685	0.061025	0.009685
5	7	285.2	272.895	10.72	0.62	0.01072	0.00062
6	8	249.57	246.27	2.075	0.06	0.002075	0.00006

З таблиці видно, що зі збільшенням часу надходження пакетів відбуваються такі зміни: Кількість згенерованих пакетів зменшується, коли час надходження збільшується. Це логічно, оскільки з більшою затримкою пакети поступають рідше. Кількість оброблених пакетів найбільше при часі надходження 3 та 7 мс, але кількість втрачених пакетів, як і кількість підключень ресурсу, зменшуються з кожним збільшенням часу надходження.

Можна зробити висновок, що при збільшенні часу надходження пакетів система має менше перевантаження, що призводить до зменшення кількості втрачених пакетів і підключених ресурсів. Це може свідчити про більш стабільну роботу системи при більших інтервалах часу між пакетами, оскільки зменшується навантаження на обробку та передачу.

Висновки

Пропозиції для покращення продуктивності системи включають оптимізацію алгоритму активації прискорення, удосконалення буферизації та дослідження інтенсивності надходження пакетів. Проведені експерименти показали, що зміна параметрів активує зниження частоти підключення ресурсів, хоча деякі зміни, такі як зменшення часу обробки на одному каналі, не дають бажаного результату. Збільшення часу між надходженням пакетів може зменшити перевантаження системи, покращивши стабільність роботи та зменшивши кількість втрачених пакетів і необхідність підключення додаткових ресурсів. Модель показала свою адекватність і може бути використана для подальшої оптимізації систем передачі даних.

ВИСНОВКИ

У ході виконання даної роботи було розроблено та досліджено модель для аналізу роботи системи передачі цифрових пакетів. Модель побудована із застосуванням формалізму мереж Петрі та реалізована за допомогою інструмента PetriObjModel, що дозволило візуалізувати і дослідити динаміку роботи системи.

У процесі моделювання були враховані ключові параметри, такі як час передачі пакетів через канали, інтервал їхнього надходження, обмеження часу обробки, порогові значення знищення пакетів, а також частота активації додаткового ресурсу. Модель дозволила аналізувати вплив цих факторів на продуктивність системи та стабільність її роботи.

Проведені експерименти підтвердили коректність моделі та її здатність адекватно відображати роботу системи в різних умовах. Зокрема, було встановлено, що збільшення інтервалу надходження пакетів зменшує частоту їхнього знищення та частоту активації прискорення, оскільки система встигає обробляти більший обсяг даних без перевантаження.

Додатково виконаний регресійний аналіз показав, що швидкість передачі пакетів через канали є критично важливим параметром, який впливає на ефективність роботи системи. Результати аналізу дозволили підтвердити вагомість часу обробки для кількості знищених пакетів та активації ресурсу.

Ця робота підкреслила значення імітаційного моделювання як інструменту для оптимізації параметрів систем передачі інформації. Завдяки застосуванню моделювання вдалося виявити ключові залежності та шляхи зниження частоти збоїв і оптимізації використання ресурсів, що сприяє підвищенню загальної ефективності системи передачі даних.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

- 1) Моделювання систем. Навчальний посібник. Стеценко І.В. [Література] - <http://surl.li/ipyjpf>
- 2) Програмний засіб з імітаційного моделювання. Стеценко І.В. [Електронний ресурс] - <https://github.com/StetsenkoInna/PetriObjModelPaint>
- 3) Моделювання систем. Курсова робота. І. В. Стеценко, О. Ю. Дифучина, А. Ю. Дифучин.[Література] - <https://ela.kpi.ua/server/api/core/bitstreams/d654ae1a-b46f-46c4-a925-465846f78c27/content>
- 4) Документації мови програмування Java [Електронний ресурс] - <https://docs.oracle.com/javase>.
- 5) Регресійний аналіз. Liudmyla Vasylieva [Електронний ресурс] - <http://surl.li/kucefq>

ДОДАТКИ

Додаток А

Файл 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;
import java.util.ArrayList;


/**
 *
 * @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 simulationCount = 200;

for (int i = 0; i < simulationCount; i++) {
    PetriObjModel simulationModel = getModel();
    simulationModel.setIsProtokol(false);
    double timeModeling = 1000; // Adjust the modeling time as needed

    simulationModel.go(timeModeling);

    String destroyedName;
    String processedName;
    String resourceName;
    String generatedName;

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

    destroyedName =
simulationModel.getListObj().get(0).getNet().getListP()[9].getName();

    double processedPackets =
simulationModel.getListObj().get(0).getNet().getListP()[12].getMark();

    processedName =
simulationModel.getListObj().get(0).getNet().getListP()[12].getName();

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

```

```

        resourceName =
simulationModel.getListObj().get(0).getNet().getListP()[24].getName();

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

        generatedName =
simulationModel.getListObj().get(0).getNet().getListP()[23].getName();


        destroyedPacketsList.add(destroyedPackets);
        processedPacketsList.add(processedPackets);
        resourceConnectionsList.add(resourceConnections);
        generatedPacketsList.add(generatedPackets);


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

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

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

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

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

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


    double averageDestroyed = calculateAverage(destroyedPacketsList);
    double averageProcessed = calculateAverage(processedPacketsList);
    double averageConnections = calculateAverage(resourceConnectionsList);
    double averageGenerated = calculateAverage(generatedPacketsList);


    System.out.println("Summary:");

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

```

```

System.out.println("Average Processed: " + averageProcessed);
System.out.println("Average Destroyed: " + averageDestroyed);
System.out.println("Average Connections: " + averageConnections);

```

```

    } catch (PetriObj.ExceptionInvalidTimeDelay e) {
        System.err.println("Caught an Exception: " + e.getMessage());
    }
}

```

```

private static double calculateAverage(List<Double> values){
    return
values.stream().mapToDouble(Double::doubleValue).average().orElse(0.0);
}

```

```

public static PetriObjModel getModel() throws ExceptionInvalidTimeDelay,
ExceptionInvalidNetStructure {
    ArrayList<PetriSim> list = new ArrayList<>();
    list.add(new PetriSim(NetLibrary.CreateNetMYTRY(6.0, 3, 7, 5.0, 5.0, 4.0,
4.0)));
    return new PetriObjModel(list);
}
}

```

Файл TestPetriNetsPaint.java

/*

- * To change this license header, choose License Headers in Project Properties.
- * To change this template file, choose Tools | Templates
- * and open the template in the editor.

```

*/

package LibTest;

import graphpresentation.PetriNetsFrame;
//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 innastetsenko
 */
public class TestPetriNetsPaint {
    /**
     * @param args the command line arguments
     */
}

```

```

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

```

```

    int simulationCount = 20;

```

```

    double timeModelingStep = 500;

```

```

    double timeModeling = 500;

```

```

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

```

```

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

```

```

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

```

```

    List<Double> averageConnectionsList = 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 < simulationCount; 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()[12].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;

```

```

double averageSim = calculateAverage(simTime);
double averageDestroyed = calculateAverage(destroyedPacketsList);
double averageProcessed = calculateAverage(processedPacketsList);
double averageConnections = calculateAverage(resourceConnectionsList);
double averageGenerated = calculateAverage(generatedPacketsList);

```

```

averageSimTime.add(averageSim);
averageDestroyedPacketsList.add(averageDestroyed);
averageProcessedPacketsList.add(averageProcessed);
averageConnectionsList.add(averageConnections);
averageGeneratedPacketsList.add(averageGenerated);
}

```

```

try (BufferedWriter writer = new BufferedWriter(new
FileWriter("C:\\Users\\Саша Головня\\Desktop\\CW\\output.csv"))) {

    writer.write("Average Sim,Average Generated,Average Processed,Average
Destroyed,Average Connections\n");

    int size = averageSimTime.size();

    for (int i = 0; i < size; i++) {
        double averageSim = averageSimTime.get(i).intValue();
        double averageDestroyed = averageDestroyedPacketsList.get(i);
        double averageProcessed = averageProcessedPacketsList.get(i);
        double averageConnections = averageConnectionsList.get(i);
        double averageGenerated = averageGeneratedPacketsList.get(i);

        writer.write(averageSim + ";" + averageGenerated + ";" +
averageProcessed + ";" + averageDestroyed + ";" + averageConnections + "\n");
    }

    System.out.println("Data written to file: output.csv");
} catch (IOException e) {
    e.printStackTrace();
}

private static double calculateAverage(List<Double> values){
    return
values.stream().mapToDouble(Double::doubleValue).average().orElse(0.0);
}

```

```

    public static PetriObjModel getModel() throws ExceptionInvalidTimeDelay,
ExceptionInvalidNetStructure {
        ArrayList<PetriSim> list = new ArrayList<>();
        list.add(new PetriSim(NetLibrary.CreateNetMYTRY(6.0, 3, 7, 5.0, 5.0, 4.0,
4.0)));
        return new PetriObjModel(list);
    }
}

```

Файл NetLibrary.java //Сеть

```

public static PetriNet CreateNetMYTRY(double GenTime, int Package, int
lostPackage, double normal1, double normal2, double fast1, double fast2) 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("Buffer 1 channel",0));
    d_P.add(new PetriP("Total number of packets processed by 1 channel",0));
    d_P.add(new PetriP("Pakages without delay",0));
    d_P.add(new PetriP("Number of acceleration shutdowns 1 channel",0));
    d_P.add(new PetriP("Channel is free",1));
    d_P.add(new PetriP("trash",0));
    d_P.add(new PetriP("Pakages w delay",0));
    d_P.add(new PetriP("Packages for destruction from the first channel",0));
    d_P.add(new PetriP("Number of accelerations 1 channel",0));
    d_P.add(new PetriP("total destruvtive packages",0)); //9
    d_P.add(new PetriP("Total number of packets processed by 1 channel",0));
    d_P.add(new PetriP("Fast channel is free and active",0));
    d_P.add(new PetriP("Total number of packets processed by 2
channel",0)); //12
}

```

```

d_P.add(new PetriP("Number of acceleration shutdowns 2 channel",0));
d_P.add(new PetriP("",0));
d_P.add(new PetriP("",1));
d_P.add(new PetriP("Packages for destruction from the second channel",0));
d_P.add(new PetriP("Channel is free",1));
d_P.add(new PetriP("Pakages without delay",0));
d_P.add(new PetriP("Number of accelerations 2 channel",0));
d_P.add(new PetriP("Pakages w delay",0));
d_P.add(new PetriP("Buffer2",0));
d_P.add(new PetriP("Fast channel is free and active",0));
d_P.add(new PetriP("total generated",0));//23
d_P.add(new PetriP("Total number of resource connections",0));//24
d_P.add(new PetriP("total accelerations",0));
d_P.add(new PetriP("total shutdowns",0));
d_P.add(new PetriP("total shutdowns",0));
d_P.add(new PetriP("total accelerations",0));
d_P.add(new PetriP("P1",0));
d_T.add(new PetriT("Processing through a normal channel",normal1));
d_T.add(new PetriT("Turn off acceleration",0.0));
d_T.get(1).setPriority(5);
d_T.add(new PetriT("Processing through a normal channel w
delay",normal1));
d_T.get(2).setPriority(2);
d_T.add(new PetriT("",0.0));
d_T.get(3).setPriority(6);
d_T.add(new PetriT("Turn on acceleration",0.0));
d_T.get(4).setPriority(2);
d_T.add(new PetriT("calculating packages from 2 channel",0.0));
d_T.add(new PetriT("Processing through a fast channel",fast1));

```

```

d_T.get(6).setPriority(3);
d_T.add(new PetriT("Processing through a fast channel w delay",fast1));
d_T.get(7).setPriority(4);
d_T.add(new PetriT("Processing through a fast channel",fast2));
d_T.get(8).setPriority(3);
d_T.add(new PetriT("",0.0));
d_T.get(9).setPriority(6);
d_T.add(new PetriT("Processing through a fast channel w delay",fast2));
d_T.get(10).setPriority(4);
d_T.add(new PetriT("Generation",GenTime));
d_T.get(11).setDistribution("norm", d_T.get(11).getTimeServ());
d_T.get(11).setParamDeviation(3.0);
d_T.add(new PetriT("Turn off acceleration",0.0));
d_T.get(12).setPriority(5);
d_T.add(new PetriT("Processing through a normal channel",normal2));
d_T.add(new PetriT("Processing through a normal channel w
delay",normal2));
d_T.get(14).setPriority(2);
d_T.add(new PetriT("Turn on acceleration",0.0));
d_T.get(15).setPriority(2);
d_T.add(new PetriT("calculating packages from 1 channel",0.0));
d_T.add(new PetriT("calculating",0.0));
d_T.add(new PetriT("calculating",0.0));
d_T.add(new PetriT("In Second channel",0.0));
d_T.add(new PetriT("First Channel",0.0));
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(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(17),d_T.get(14),1));
d_In.add(new ArcIn(d_P.get(21),d_T.get(14),2));
d_In.get(5).setInf(true);
d_In.add(new ArcIn(d_P.get(21),d_T.get(14),1));
d_In.add(new ArcIn(d_P.get(10),d_T.get(19),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(16),d_T.get(5),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(6),d_T.get(4),1));
d_In.add(new ArcIn(d_P.get(20),d_T.get(15),1));
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(16).setInf(true);
d_In.add(new ArcIn(d_P.get(0),d_T.get(2),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(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(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(24).setInf(true);
d_In.add(new ArcIn(d_P.get(21),d_T.get(10),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(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(11),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(0),d_T.get(7),3));
d_In.get(31).setInf(true);
d_In.add(new ArcIn(d_P.get(0),d_T.get(7),1));
d_In.add(new ArcIn(d_P.get(15),d_T.get(11),1));
d_In.add(new ArcIn(d_P.get(7),d_T.get(16),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_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_Out.add(new ArcOut(d_T.get(0),d_P.get(2),Package)); //gen
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(6),d_P.get(11),1));
d_Out.add(new ArcOut(d_T.get(6),d_P.get(2),Package)); //gen
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(14),d_P.get(17),1));
d_Out.add(new ArcOut(d_T.get(14),d_P.get(20),lostPackage)); //lost
d_Out.add(new ArcOut(d_T.get(14),d_P.get(16),1));
d_Out.add(new ArcOut(d_T.get(19),d_P.get(21),1));
d_Out.add(new ArcOut(d_T.get(3),d_P.get(5),1));
d_Out.add(new ArcOut(d_T.get(5),d_P.get(9),1));
d_Out.add(new ArcOut(d_T.get(13),d_P.get(18),Package)); //gen
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(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(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(2),d_P.get(4),1));
d_Out.add(new ArcOut(d_T.get(2),d_P.get(6),lostPackage)); // lost
d_Out.add(new ArcOut(d_T.get(2),d_P.get(7),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),Package)); //gen
d_Out.add(new ArcOut(d_T.get(8),d_P.get(12),1));
d_Out.add(new ArcOut(d_T.get(9),d_P.get(14),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),lostPackage));//lost
d_Out.add(new ArcOut(d_T.get(18),d_P.get(24),1));
d_Out.add(new ArcOut(d_T.get(17),d_P.get(24),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),lostPackage));//lost
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(16),d_P.get(9),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));

```



```
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));  
PetriNet d_Net = new PetriNet("MYTRY",d_P,d_T,d_In,d_Out);  
PetriP.initNext();  
PetriT.initNext();  
ArcIn.initNext();  
ArcOut.initNext();  
  
return d_Net;  
}
```

