Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Модель і метод обміну даними транспортного рівня мультисервісних мереж

(тема)

Виконав:
студент II курсу, групи СПзм-18-2
Кузьома І.О.
(прізвище, ініціали)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма
Системне програмування
(повна назва освітньої програми)

Керівник: проф. Каргін А.О.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри ЕОМ _____ Коваленко А.А.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

Спеціальність 123 – Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-наукова
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне програмування
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
"____" _____ 20___ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Кузьомі Іллі Олександровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Модель і метод обміну даними транспортного рівня мультисервісних мереж

затверджена наказом по університету від "30" березня 2020 р. № 43Стз

2. Термін подання студентом роботи до екзаменаційної комісії 18 травня 2020 р.

3. Вхідні дані до роботи

Реалізації протоколів TCP: Tahoe, Reno, NewReno, Vegas

Моделі мережевого трафіку

Топологія мультисервісної мережі

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз факторів, що впливають на ефективність мультисервісних комп'ютерних мереж

2. Вибір та розробка моделей, розробка методу обміну даними транспортного рівня

3. Опис інструментальних засобів моделювання і виконання моделювання запропонованих результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати - 12 арк.

6. Консультанти розділів роботи (заповнюється за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз ефективності процесів обміну даними	30.03.20-08.04.20	
2	Вибір та розробка моделей та методу	09.04.20-04.05.20	
3	Опис інструментальних засобів моделювання	05.05.20-11.05.20	
	і виконання моделювання		
4	Оформлення пояснювальної записки	28.04.20-11.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

Керівник роботи _____ проф. Каргін А.О.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 86 с., 29 рис., 1 дод., 17 джерел.

МОДЕЛЮВАННЯ, ПЕРЕВАНТАЖУВАЛЬНЕ ВІКНО, ПРОТОКОЛ, TCP, АЛГОРИТМ УПРАВЛІННЯ ПЕРЕВАНТАЖЕННЯМИ, РЕАЛІЗАЦІЯ, ЕФЕКТИВНІСТЬ.

Метою атестаційної роботи є розробка моделі і методу обміну даними транспортного рівня мультисервісних мереж, що дозволять збільшити пропускну здатність, та їх реалізації.

У ході виконання атестаційної роботи проведено огляд аспектів функціонування протоколів транспортного рівня стеку TCP/IP TCP Vegas та TCP Reno. Обране середовище моделювання та виконане моделювання з використанням мови скриптів OTcl. Результати представлені у вигляді графиків.

ABSTRACT

Master’s thesis: 86 pages, 29 figures, 1 appendix, 17 sources.

SIMULATION, CONGESTION WINDOW, PROTOCOL, TCP, CONGESTION CONTROL ALGORITHM, IMPLEMENTATION, EFFICIENCY.

The major goal of this thesis is to develop a model and method of data exchange at transport layer of multiservice networks, which will increase the throughput, as well as their implementation.

During preparation of the thesis, the aspects of operation of TCP/IP stack’s transport protocols TCP Vegas and TCP Reno were investigated. Simulation environment was chosen and simulation using OTcl scripting language was conducted. The results are presented as graphs.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACK – підтвердження отримання даних (англ., acknowledgment)

FTP – протокол передачі файлів (англ., file transfer protocol)

RTT – час передачі пакета (англ., round-trip time)

SACK – вибіркові підтвердження (англ., selective acknowledgement)

TCP – протокол управління передачею (англ., transmission control protocol)

TCP/IP – протокол управління передачею / міжмережний протокол (англ., transmission control protocol / internet protocol)

UDP – протокол дейтаграм користувача (англ., user datagram protocol)

ВСТУП

Комп’ютери та комп’ютерні технології на сучасному етапі розвитку людства відіграють дуже важливу роль для прогресу, без них неможливо уявити жодну галузь, особливо в Україні. Завдяки комп’ютерним технологіям не припиняють свій розвиток мережі передачі даних, у тому числі мультисервісних, до яких продовжує підключатися багато різноманітних користувачів і організацій, розрізнених географічно. Найбільш істотною зміною в сучасних мережах є збільшення кількості переданого ними трафіку. При збільшенні числа нових користувачів в мережі збільшується і кількість одночасно працюючих служб і додатків, що, в свою чергу, призводить до зміни якісного та кількісного вмісту трафіку.

Зростаюча необхідність використання розподілених обчислень, обробки місткої наукової інформації, проведення відеоконференцій і роботи інших сучасних додатків, для яких необхідно більш часте і активне використання мережевих ресурсів, тягне за собою необхідність внесення змін в існуючі мультисервісні мережі для збільшення їх ефективності, на різних рівнях, і , в найбільшій мірі, транспортному.

На сьогоднішній день протокол управління передачею даних (TCP) є найбільш використовуваним протоколом транспортного рівня в мережах передачі даних. Він організовує роботу таких сервісів як FTP, Telnet і інших, забезпечуючи заснований на з’єднанні надійний сервіс доставки даних поверх протоколу IP. Протокол TCP гарантує достовірну і послідовну доставку даних. Крім того, він забезпечує управління пропускною здатністю і розподіл мережевих ресурсів. Важливість протоколу TCP можна визначити, виходячи з відсотка загального трафіку, що передається з його допомогою (більше 95% всіх переданих байт даних в світі або 80-95% всіх пакетів в мережах). Цей факт привів до величезного

числа досліджень, присвячених пропускній здатності протоколу TCP.

До теперішнього часу ще недостатньо вивчені властивості ефективності протоколу TCP, а також методи обміну даними та збільшення ефективності різноманітних протоколів, що враховують вплив на неї різноманітних факторів і типів трафіку, які володіють різними властивостями.

1 АНАЛІЗ ФАКТОРІВ, ЩО ВПЛИВАЮТЬ НА ЕФЕКТИВНІСТЬ МУЛЬТИСЕРВІСНИХ КОМП'ЮТЕРНИХ МЕРЕЖ

Питання, пов'язані з ефективністю, відіграють дуже важливу роль в мультисервісних комп'ютерних мережах. Коли множина комп'ютерів пов'язані один з одним, процес їх взаємодії стає дуже складним і може привести до непередбачуваних наслідків і зниження пропускної здатності. У зв'язку з цим, аналіз факторів, що впливають на ефективність мультисервісних комп'ютерних мереж, є актуальною проблемою в сучасних мережах.

1.1 Основні фактори, що враховуються при аналізі ефективності мультисервісних комп'ютерних мереж

Класифікацію факторів, що впливають на ефективність мультисервісних комп'ютерних мереж [1] для наочності зручно представити в графічному вигляді, що показано на рисунку 1.1. Існують три принципово різні схеми комутації абонентів в мережах: комутація каналів, комутація пакетів і комутація повідомлень.

Мережі з комутацією пакетів з'явилися як результат експериментів з першими глобальними мережами. А мережі з комутацією повідомлень працюють тільки для деяких служб, причому найчастіше поверх мережі з комутацією пакетів, як служба прикладного рівня. Кожна з цих схем має свої переваги і недоліки. Наприклад, мережі з комутацією каналів добре комутують потоки даних постійної інтенсивності, але не можуть динамічно перерозподіляти пропускну здатність між одночасно існуючими потоками даних. У той же час мережі з комутацією пакетів більш ефективні в тому сенсі, що обсяг переданих даних від всіх абонентів мережі в одиницю часу більше, ніж при використанні мережі з комутацією каналів. Однак для

кожної пари абонентів пропускна здатність мережі може виявитися нижче, ніж у мережі з комутацією каналів, за рахунок черг пакетів в комутаторах.

Мережі з комутацією повідомлень призначені для організації взаємодії користувачів у фоновому режимі, коли не очікується негайної реакції на повідомлення. При цьому методі повідомлення передається через кілька транзитних вузлів, де воно може цілком буферизоване. Але, згідно з сучасними дослідженнями, найбільш перспективна технологія комутації пакетів, як більш гнучка і універсальна [2].

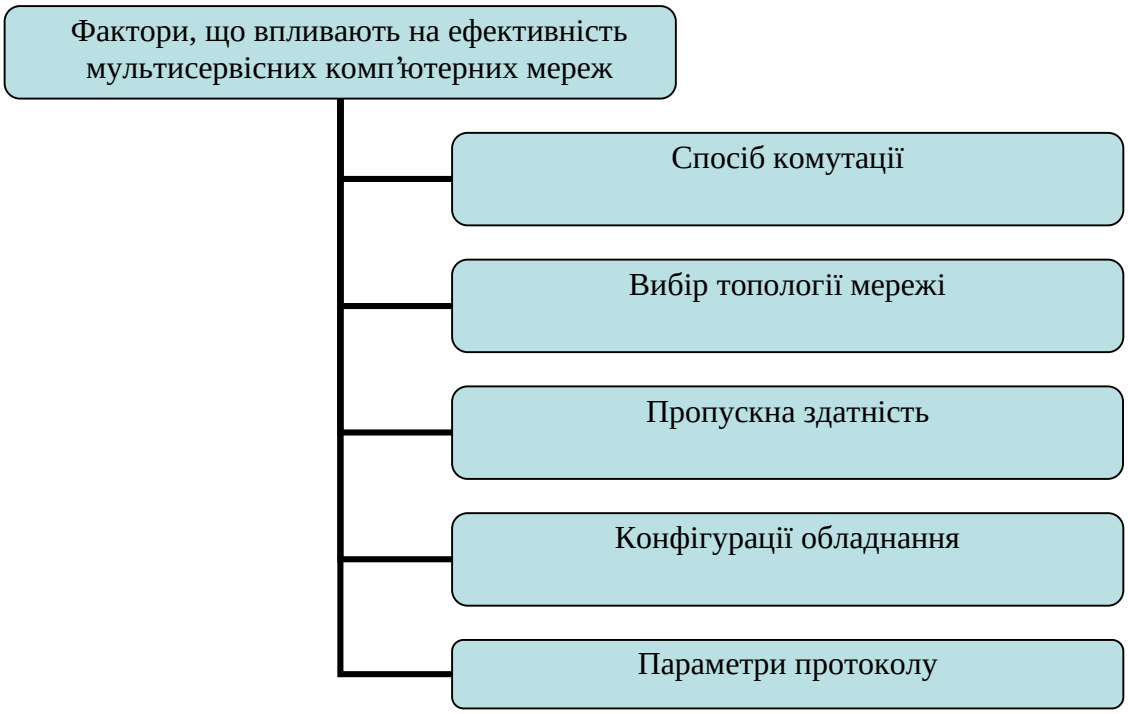

Рисунок 1.1 – Фактори, що впливають на ефективність мультисервісних комп'ютерних мереж

На рисунку 1.2 наведено графік розподілу областей застосування різних методів комутації в вигляді залежності пропускної здатності каналів мережі C_k від коефіцієнта використання каналу $K_н$.

Даний розподіл відповідає технічному рівню апаратури зв'язку та засобів обчислювальної техніки, а також їх вартості при реалізації різних сучасних способів комутації.

Топологія мережі є одним з найбільш критичних факторів, що впливають на пропускну здатність. У разі необхідності основні існуючі топології (комутовану, зіркоподібну, кільцеву і шинну) можна комбінувати довільним чином. Крім вибору однієї з топологій, слід враховувати і такі чинники, як розташування робочих станцій в будівлі, придатність і обмеження на довжину кабелю, тип і спосіб проводки [3, 4].

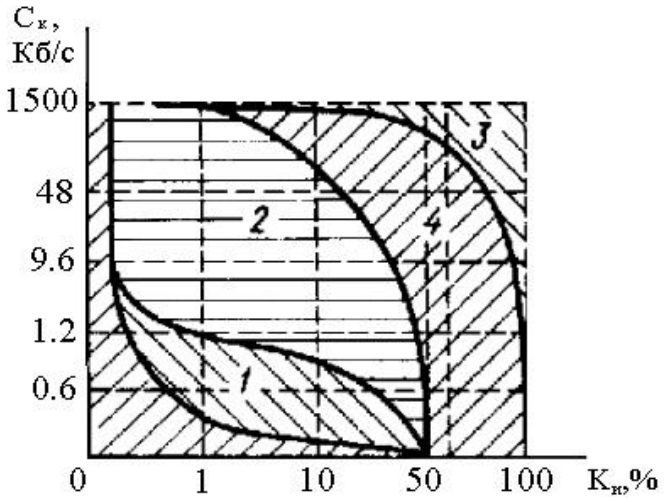

Рисунок 1.2 – Способи комутації в комп’ютерних мережах:
1 – комутація каналів; 2 –комутація пакетів;
3 – прямий зв’язок; 4 – кросова комутація

Зрештою, основним критерієм вибору топології мережі є вимоги користувачів до ефективності. Такі фактори, як вартість, передбачувана модернізація і обмеження існуючих технологій, відіграють другорядну роль.

Інтенсивність потоку повідомлень по *i*-му каналу визначається у відповідності з наступним виразом [5]:

$$\lambda_i = \sum_{j=1}^{n} \sum_{k=1}^{n} \lambda_{jk.},$$

де λ_{jk} – інтенсивність трафіку, який надходить в канал від зовнішніх

відправників для пакетів, що виникають у вузлі *j* і призначених для вузла *k*;

N – кількість вузлів.

Для *n*-вузлової мережі для визначення середнього часу передачі повідомлення в *i*-му каналі використовується такий вираз [5]:

$$T_i = \frac{1}{\nu_i - \lambda_i},$$

де ν_i - середня довжина повідомлення.

Оскільки всі канали в мережі працюють незалежно, то для обчислення середнього часу передачі повідомлення в мережі зручно використовувати такий вираз [5]:

$$T = \sum_{i=1}^{n} \frac{\lambda_i}{\gamma} \frac{1}{\nu_i - \lambda_i}.$$

Якщо результати розрахунків затримок в мережі задовільняють встановленим параметрам, то отримана топологічна структура може бути прийнята за основу для подальшої розробки, що полягає у визначенні конкретних типів апаратної частини, реалізації обраних протоколів, способів управління, комутації, в створенні програмного забезпечення і вирішенні питань технічного проектування.

Для досягнення найбільшої пропускної здатності необхідно зменшити величину затримки *T* при змінній пропускній здатності каналів *C*, і заданих обмеженнях на вартість *W* і конкретні пропускні здатності C_i:

$$W = \sum_{i=1}^{NM} W_i(C) \le W_{\max};$$

$$C_i \ge f_i.$$

Пропускна здатність мережі вимірюється за допомогою тимчасових показників (як правило, оцінюють затримку, що вноситься мережею при виконанні обміну даними) і показників пропускної здатності (що відображають кількість інформації, переданої мережею в одиницю часу). Ці два типи показників є взаємно зворотними, і, знаючи один з них, можна обчислити інший.

Вибір правильних параметрів конфігурації, які відповідають вимогам конкретної мережі, призводить до підвищення швидкості передачі даних в мережі. Кінцева мета – домогтися збільшення ефективності при обміні даними між вузлами. Завдання полягає в тому, щоб при збереженні цілісності даних передавати найбільшу їх кількість за фіксований проміжок часу. Вирішення цього завдання можливе лише в результаті використання оптимальної конфігурації. Глобальна мета полягає в забезпеченні максимально ефективного використання всіх апаратних пристроїв мережі і їх компонентів.

Однією з важливих часових характеристик пропускної здатності мережі є час реакції, який визначається як інтервал часу між виникненням запиту користувача до певної мережевої служби і отриманням відповіді на цей запит.

Час реакції на запит складається з часу підготовки запитів на клієнтській стороні, часу передачі запитів між клієнтом і сервером через сегменти мережі і проміжне мережне обладнання, часу обробки запитів на сервері, часу передачі відповідей від сервера клієнту і часу обробки одержуваних від сервера відповідей. Також цей показник залежить від поточного стану елементів мережі – завантаженості елементів мережі, через які проходить запит.

Знання складових часу реакції, обумовлених роботою мережі, дає можливість оцінити пропускну здатність окремих елементів мережі, виявити проблемні ділянки і, в разі необхідності, виконати модернізацію мережі для підвищення її загальної пропускної здатності.

При оцінці пропускної здатності мережі по відношенню до всіх вузлів в цілому, використовуються середньозважені і порогові критерії.

Середньозважений критерій є сумою часу реакції певного числа вузлів при взаємодії з певним числом серверів мережі з використанням певної служби. Граничний критерій відображає найбільший час реакції по всіх можливих поєднаннях клієнтів, серверів і сервісів [2]. Як правило, криві реального та очікуваного часу реакції збігаються тільки в межах того навантаження, з яким реально малась справа. Досвід є досить надійним тільки при повному завантаженні мережі. При подальшому збільшенні навантаження пропускна здатність мережі буде різко знижуватися.

Серед факторів, що впливають на ефективність мультисервісних комп'ютерних мереж, параметри мережних протоколів і, зокрема, протоколів транспортного рівня, займають важливе місце.

В умовах сучасних високошвидкісних мереж починають позначатися деякі обмеження, властиві протоколу TCP. Безумовно, модифікаціям і оптимізації протоколу TCP було приділено і продовжує приділятися досить багато уваги. Для цього застосовуються алгоритми Нагля, Кларка, Джейкобсона, Карна та інші. При некоректно обраних параметрах з'єднання протоколом TCP можуть мати місце перевантаження і, як наслідок, – втрата пропускної здатності і зниження ефективності.

1.2 Сучасні реалізації протоколу транспортного рівня TCP

Протокол TCP до сьогоднішнього дня зазнає безперервні модифікації. На даний момент його основними реалізаціями є Tahoe, Reno, New Reno і Vegas (рисунок 1.3).

З кожною новою модифікацією розробники прагнули позбутися певного недоліку TCP, переважно пов'язаного з характером функціонування протоколу при виникненні різного роду перевантажень в різноманітних сценаріях. Всі перераховані модифікації TCP справляються з ними по-

різному, причому ступінь ефективності часто залежить від характеру наявних перевантажень.

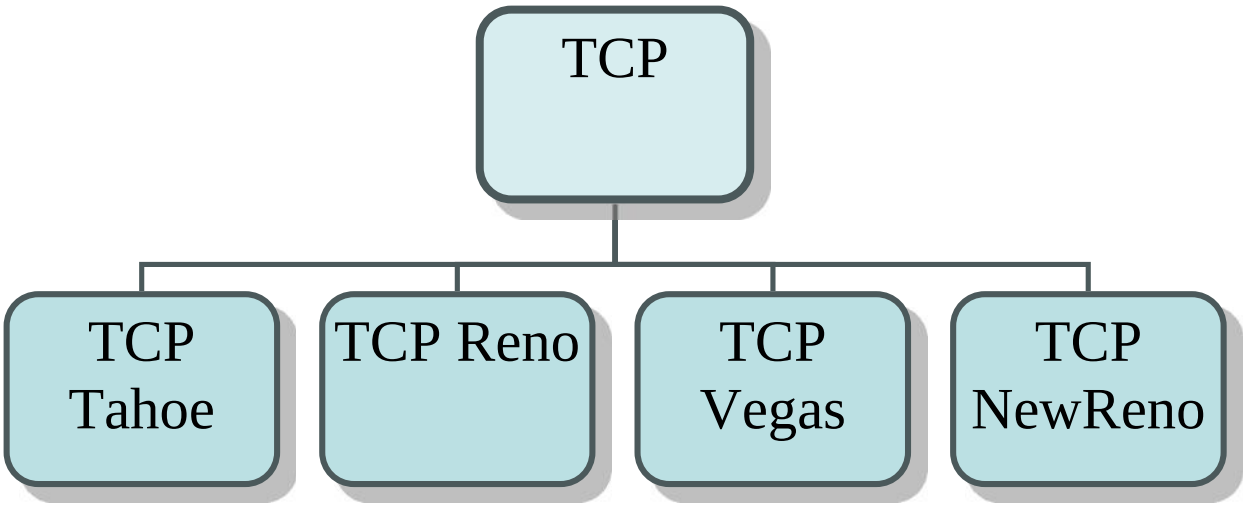

Рисунок 1.3 – Сучасні реалізації протоколу TCP

1.2.1 Реалізація TCP Tahoe

Сучасні реалізації протоколу TCP містять алгоритми управління завантаженням мережі, спрямовані на підтримку заданої ефективності. Ранні реалізації протоколу TCP використовували модель, засновану на механізмі підтверджень сегментів, потребуючу закінчення таймера повторної передачі для повторної відправки втраченого під час передачі сегмента даних. Такі протоколи лише ненабагато зменшували перевантаження в мережі.

Реалізація Tahoe TCP додала нові алгоритми і поліпшення до більш ранніх реалізацій. Ці алгоритми включають фази повільного старту, запобігання перевантаження і швидкої повторної передачі, а поліпшення – модифікації способів оцінки часу передачі пакету, що використовується для встановлення значення тайм-ауту повторної передачі. При використанні алгоритму швидкої повторної передачі, після отримання певної кількості повторюваних пакетів підтверджень для одного і того ж відправленого протоколом TCP сегмента даних, відправник робить висновок, що пакет був

втрачений і передає його повторно без очікування закінчення таймера повторної передачі, що призводить до кращого використання каналу і підвищенню пропускної здатності з’єднання.

Більш ранні реалізації протоколу TCP починали процес з’єднання з відправки в мережу багатьох сегментів відправником, з розміром вікна, який заявив одержувач. При такій схемі з’єднання все працює коректно тільки в разі, коли ці два вузли знаходяться в одній мережі, але якщо між ними будуть маршрутизатори і більш повільні з’єднання, то виникають перевантаження. Деякий проміжний маршрутизатор повинен буферизувати пакети в свою чергу, що обумовлює ймовірність перевищення максимального розміру черги пакетами, що надходять в неї. Алгоритм для запобігання цього стану називається повільним стартом. Він функціонує з метою узгодження швидкості відправки нових пакетів в мережу і швидкості повернення підтверджень одержувачем.

Повільний старт додає ще одну змінну для відправника протоколом TCP: перевантажувальне вікно (далі – s_w). Коли встановлюється нове з’єднання, розмір перевантажувального вікна встановлюється в один сегмент. Кожен раз при отриманні пакету ACK розиір перевантажувального вікна збільшується на один сегмент. Відправник може передавати дані з розміром вікна, рівним меншому зі значень поточних розмірів: перевантажувального вікна і заявленого вікна. Перевантажувальне вікно реалізується методом управління потоком на стороні відправника, в той час як заявлене вікно – методом управління потоком на стороні одержувача. Перший базується на оцінці відправником поточного перевантаження в мережі, а останній пов’язаний з кількістю вільного місця в буфері одержувача конкретного з’єднання.

Відправник починає процес передачі даних з передачі одного сегмента і очікування пакета ACK для нього. Коли пакет ACK прийнятий, розмір перевантажувального вікна збільшується з одного до двох сегментів (одночасно переданих). Коли кожен з двох відправлених сегментів

підтверджено, перевантажувальне вікно збільшується до чотирьох сегментів і так далі, що представляє собою експоненціальне зростання розміру вікна. В деякий момент часу може бути перевищена пропускна здатність ділянки мережі і проміжний маршрутизатор почне відкидання пакетів, які вже не поміщаються в його чергу. Це говорить відправнику, що його перевантажувальне вікно стало занадто великим.

Ранні реалізації протоколу TCP виконували алгоритм повільного старту, тільки якщо другий вузол знаходився в іншій мережі. Сучасні реалізації завжди виконують повільний старт. Часова діаграма фази повільного старту зображена на рисунку 1.4.

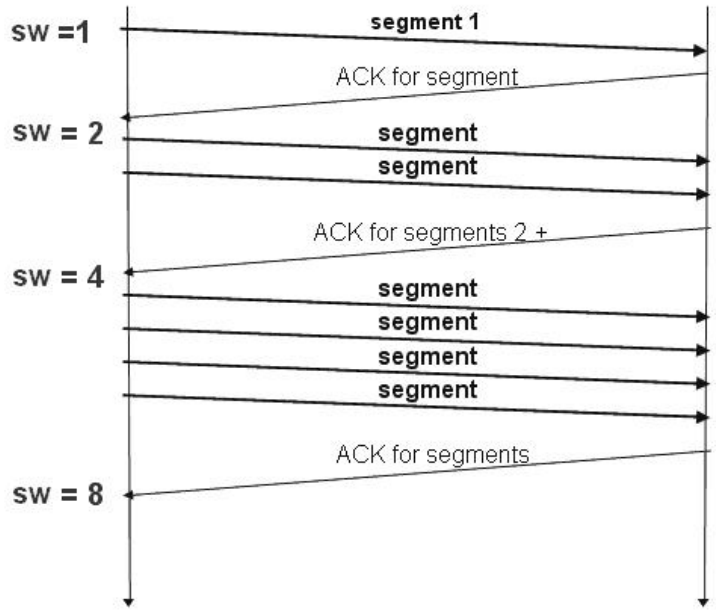

Рисунок 1.4 – Часова діаграма фази повільного старту

Найбільш часто перевантаження відбувається як наслідок прибуття даних по відносно швидкому каналу, а відправлення – по більш повільному каналу, або коли множинні вхідні потоки даних прибувають в маршрутизатор, вихідна пропускна здатність якого менша за суму пропускних здатностей вхідних каналів. Таким чином, запобігання перевантаження полягає в запобіганні втрат пакетів. Втрата пакету може

бути виявлена за допомогою одного з двох ознак: настання тайм-ауту або отримання повторюваних пакетів ACK.

Запобігання перевантаження і повільний старт є незалежними фазами, які мають різні цілі. Однак, при настанні перевантаження, протокол TCP повинен знизити свою швидкість передачі пакетів в мережу і потім викликати фазу повільного старту для відновлення попереднього стану. Таким чином, на практиці ці фази реалізовані спільно.

Запобігання перевантаження і повільний старт вимагають підтримки двох змінних для кожного з'єднання: розміру перевантажувального вікна, s_w, і порогового значення виходу з фази повільного старту, s_s. Об'єднаний алгоритм функціонує наступним чином.

Відбувається ініціалізація з'єднання; значення s_w встановлюється в один сегмент, а s_s – в 65535 байт.

У процесі відправки даних, протокол TCP ніколи не відправляє кількість даних більшу, ніж мінімальне між значеннями s_w і заявленим розміром вікна одержувача.

Коли відбувається перевантаження, за допомогою значення s_s зберігається половинне поточне значення розміру перевантажувального вікна (мінімальне зі значень s_w і заявленого одержувачем вікна, але не менше двох сегментів). Крім того, якщо перевантаження виявляється за допомогою тайм-ауту, значення s_w встановлюється в один сегмент і запускається фаза повільного старту.

Коли отримання нових даних підтверджується одержувачем, значення s_w збільшується, але спосіб його збільшення залежить від того, яка фаза виконується в даний момент протоколом TCP – повільний старт або запобігання перевантаження. Якщо значення s_w менше або дорівнює s_s, то протокол TCP знаходиться у фазі повільного старту; інакше TCP виконує алгоритм запобігання перевантаження. Повільний старт триває до досягнення TCP половинного значення пропускної здатності, яка була в момент настання перевантаження (оскільке було записано половинне значення розміру вікна),

а потім настає фаза запобігання перевантаження.

Фаза повільного старту починається зі значення змінної s_w в один сегмент і потім збільшується на один сегмент при кожному отриманні пакету ACK. Фаза запобігання перевантаження збільшує значення s_w на величину $s*s/s_w$ кожен раз при отриманні пакету ACK, де s – поточний розмір сегмента. Це є лінійним збільшенням s_w, на відміну від його експоненціального зростання під час фази повільного старту.

Рисунок 1.5 відображає механізм комбінованого функціонування фаз повільного старту і запобігання перевантаження.

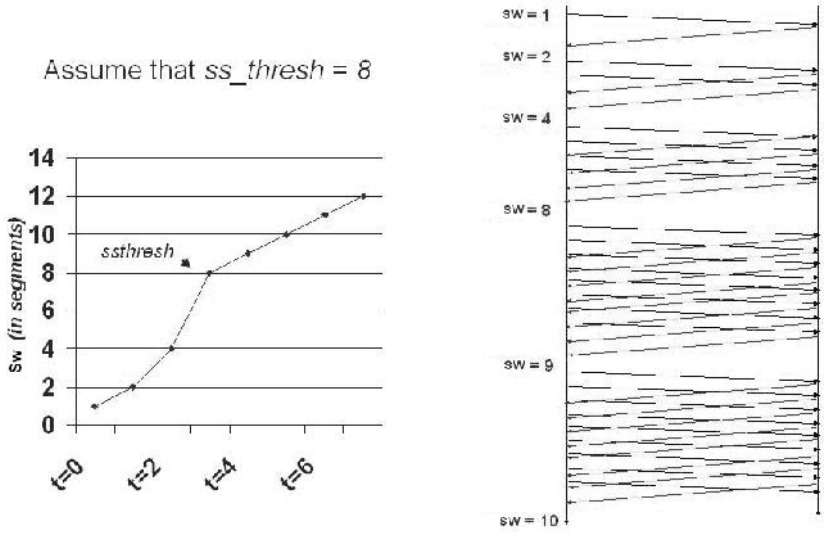

Рисунок 1.5 – Приклад спільного функціонування фаз повільного старту і запобігання перевантаження

Фаза швидкої повторної передачі вперше з’явилася в реалізації «4.3BSD Tahoe» і слідувала за повільним стартом, що є головною відмінністю між протоколами TCP Tahoe і TCP Reno.

Для виявлення втрати пакета протоколу Tahoe необхідний час, рівний значенню тайм-ауту і, фактично, це вимагає навіть більше часу внаслідок грубого масштабу часу таймера. Таким чином, кожен раз при втраті пакету протокол очікує тайм-аут і пропускна здатність каналу

недовикористовується. Це обумовлює велику вартість для з’єднань, в яких значення добутку поточних значень часу передачі пакету у встановленому з’єднанні і розміру смуги пропускання, доступної з’єднанню, досить велике.

1.2.2 Реалізація TCP Reno

Протокол TCP Reno успадкував від Tahoe його основний принцип – використання фази повільного старту і наявність грубо масштабованого таймера повторної передачі, а також вимоги негайного отримання підтвердження після успішної передачі кожного сегмента даних. Він заснований на припущенні, що отримання повторюваного пакету підтвердження свідчить про те, що сегменти досягли одержувача не в первісному порядку, або сегмент був втрачений.

Якщо надходить деяка кількість повторюваних підтверджень, то це означає, що вже пройшло чимало часу, і навіть якщо сегмент пішов по більш довгому маршруту, то він вже повинен досягти одержувача. Існує дуже велика ймовірність того, що він був втрачений. Таким чином, в TCP Reno додана нова фаза, названа «швидка повторна передача».

Якщо відправник приймає три повторювані пакети ACK, то це розцінюється як ознака втрати сегмента, і він повинен бути повторно переданий без очікування тайм-ауту. Таким чином, канал практично повністю використовується. Ще одне нововведення полягає в тому, що після втрати пакета протокол не зменшує розмір свого перевантажувального вікна до 1, оскільки це призведе до недовикористання каналу, а входить в алгоритм швидкої повторної передачі, що показано на рисунку 1.6.

Швидка повторна передача. Протокол TCP Reno може генерувати негайне підтвердження (повторюваний пакет ACK) при отриманні пошкодженого сегмента. Повторюваний пакет ACK не повинен затримуватися. Його призначення – повідомити відправнику про отримання пошкодженого сегмента і про номер очікуваної послідовності. Оскільки

протокол TCP не знає, чи є цей повторюваний ACK ознакою втрати або переупорядкування сегментів, він очікує прийому невеликої кількості повторюваних пакетів ACK. Передбачається, що якщо відбулося лише переупорядкування сегментів, з'явиться тільки один або два повторюваних пакета ACK до обробки переупорядкованого сегмента, що згенерує новий пакет ACK. Якщо поспіль отримані три або більше повторювані пакети ACK, то це є точним зазначенням втрати сегмента. Після цього протокол TCP виконує повторну передачу відсутньої частини повідомлення без очікування закінчення таймера повторної передачі.

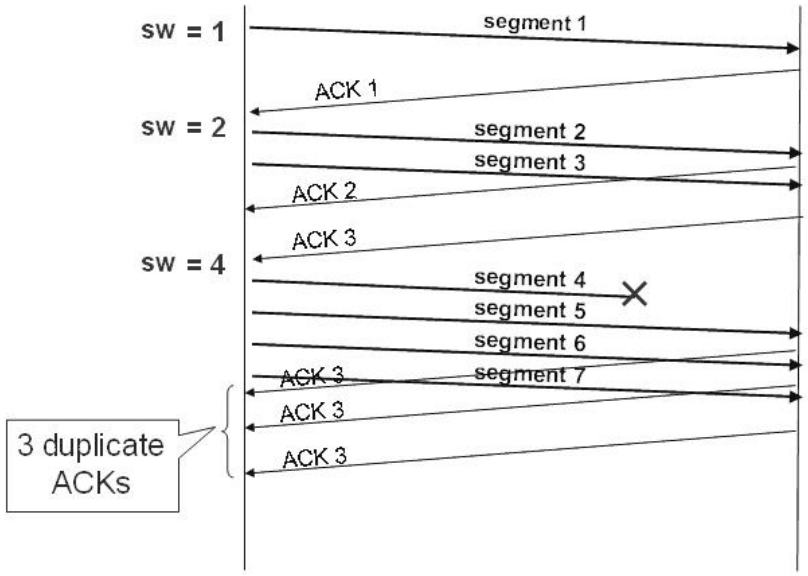

Рисунок 1.6 – Часова діаграма фази швидкої повторної передачі

Після швидкої повторної передачі відсутнього сегмента запускається фаза запобігання перевантаження, що дозволяє збільшити пропускну здатність при помірному перевантаженні, особливо при великих розмірах перевантажувального вікна. Причина, по якій немає необхідності в запуску фази повільного старту в даному випадку полягає в тому, що отримання повторюваних пакетів ACK говорить TCP не просто про втрату пакету, а про те, що одержувач прийняв інший сегмент (що свідчить про те, що сегмент

покинув мережу і знаходиться в буфері одержувача, тобто дані, як і раніше передаються в з'єднанні і протокол TCP не хоче раптово знижувати свою пропускну здатність запуском в фази повільного старту).

На рисунку 1.7 приведена залежність динаміки зміни розміру перевантажувального вікна від часу для різних модифікацій протоколу TCP.

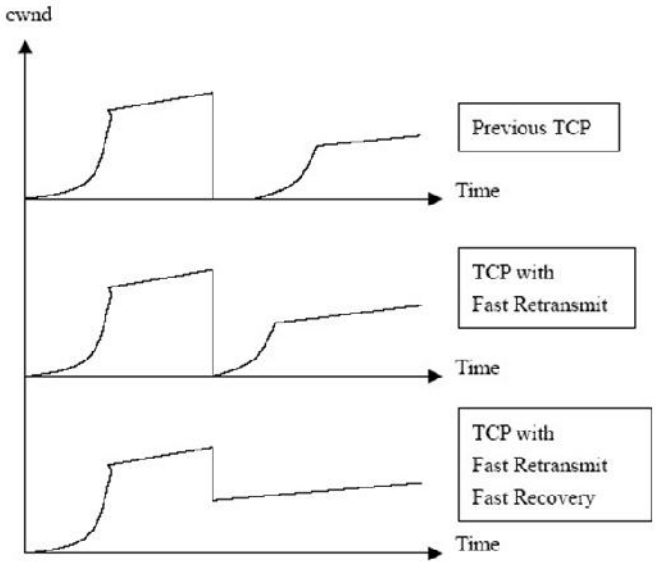

Рисунок 1.7 – Динаміка зміни розміру перевантажувального вікна для різних модифікацій протоколу TCP

Протокол TCP Reno досить ефективний при незначних втратах пакетів. Коли відбувається багато втрат пакетів в межах одного переданого вікна, ефективність протоколу TCP Reno стає сумірною з TCP Tahoe. Причина полягає в здатності TCP Reno виявляти тільки поодинокі втрати пакетів. Якщо мають місце множинні втрати пакетів, то інформація про першу втрату пакета надходить за допомогою повторюваних пакетів ACK, а інформація про другий втрачений пакет буде отримана тільки після досягнення пакетом ACK відправника для повторно переданого першого сегмента. Також можливо, що значення s_w двічі зменшиться при втраті пакетів в одному вікні.

Наступний недолік протоколу TCP Reno полягає в тому, що якщо розмір вікна невеликий в момент виникнення втрати, то достатня кількість

повторюваних підтверджень для швидкої повторної передачі ніколи не буде отримана, і необхідно буде чекати закінчення тайм-ауту. Отже, така схема не може бути ефективною для виявлення множинних втрат пакетів.

1.2.3 Реалізація TCP NewReno

Протокол TCP NewReno використовує модифікований алгоритм TCP Reno у відправнику, який полягає в обході очікування закінчення таймера повторної передачі під час множинних втрат у фазі швидкого відновлення. Коли надходять часткові пакети ACK, що підтверджують деякі, але не всі пакети з числа переданих на початку фази швидкого відновлення. У протоколі TCP NewReno часткові пакети ACK не виводять протокол з фази швидкого відновлення: якщо вони отримані в фазі швидкого відновлення, то сприймаються як ознака того, що пакет слідував відразу за підтвердженим пакетом у втраченій послідовності і повинен бути повторно переданий.

Таким чином, коли відбувається втрата багатьох пакетів з одного відправленого вікна даних, протокол TCP NewReno може відновлюватися без тайм-ауту повторної передачі, повторно передаючи один втрачений пакет за цикл передачі до тих пір, поки всі втрачені пакети з вікна не будуть повторно передані. TCP NewReno залишається у фазі швидкого відновлення до підтвердження всіх даних, відправлених в мережу до входу в цю фазу.

Параметр «maxburst» обмежує до чотирьох кількість пакетів, які можуть бути послані у відповідь на одиничний вхідний пакет ACK; таке значення дозволяє більш точно відтворювати поведінку TCP Reno під час фази швидкого відновлення. Параметр «maxburst» потрібен тільки для першого вікна посиланих даних після виходу з фази швидкого відновлення. Якщо відправник був попереджений заявленим вікном одержувача про неможливість відправки пакетів під час фази швидкого відновлення, то відправник може відправляти великий обсяг пакетів без параметра «maxburst» аж до самого виходу з фази швидкого відновлення.

Для виявлення однієї втрати пакета потрібен час, рівний одному RTT. Тільки коли отриманий пакет ACK для першого повторно переданого сегмента, стає можливим визначити номери наступних втрачених пакетів.

1.2.4 Реалізація TCP Vegas

Протокол TCP Vegas реалізує поліпшену стратегію управління передачею даних, яка полягає в спробі передбачити момент настання перевантаження, а також коригує розмір перевантажувального вікна, намагаючись зменшити швидкість відправки трафіку відправником в мережу ще до того, як пакети почнуть відкидатися проміжними вузлами. Такий підхід ґрунтується на існування ознак, що передують перевантаження в мережі.

Переваги протоколу TCP Vegas в значній мірі обумовлені меншою кількістю повторних передач пакетів.

TCP Vegas в процесі своєї роботи перебуває в наступних трьох фазах:

- повільний старт;
- запобігання перевантажень;
- поторна передача.

Для запобігання втрат пакетів в фазі повільного старту, використовується новий механізм виявлення перевантажень, а експоненціальне збільшення значення s_w відбувається в кожному парному інтервалі часу RTT. Коли справжня швидкість відправки пакетів в мережу відправником стає нижчою за очікувану, протокол TCP Vegas переходить з фази повільного старту в фазу запобігання перевантажень, якій відповідає лінійне збільшення або зменшення розміру вікна.

Після отримання кожного пакету підтвердження ACK в мережу відправляються два нових пакета даних. Отже, якщо в мережі вже виникло перевантаження, то такий алгоритм повільного старту не буде працювати коректно: відправляються пакети будуть губитися, оскільки не існує механізму зворотного зв’язку для оповіщення відправника про необхідність

зниження швидкості відправки зменшенням значення s_w .

Механізм повторних передач протоколу TCP Vegas є розширеною версією такого з TCP Reno і його більш точною моделлю, оскільки він підтримує мітки системного часу для кожного пакета, що відправляється відправником. Коли відправник TCP Vegas отримує пакет ACK, він здійснює обчислення значення *T* як різниці між поточним значенням системного часу і значенням мітки часу, що містилася в надісланому пакеті з відповідним номером. Згодом значення *T* використовується для прийняття рішення про повторну передачу, відповідно до однієї з двох схем:

- якщо прийнято повторюваний пакет ACK, протокол TCP Vegas обчислює різницю між поточним значенням часу і значенням що містилось у відповідному пакеті даних мітки часу; якщо ця різниця більше значення T_T, то пакет передається повторно, без очікування третього повторюваного пакета ACK;

- при отриманні наступного пакета ACK і якщо він є першим або другим після повторної передачі, TCP Vegas порівнює часовий проміжок, що пройшов з моменту відправки пакета зі значенням T_T і якщо він більше, то пакет передається повторно без очікування повторюваного пакета ACK.

Після успішної повторної передачі втраченого пакета, протокол TCP Vegas зменшує розмір свого вікна на 25%, що обумовлено корекцією розміру вікна на початкових стадіях перевантаження, в той час як TCP Reno зменшує його розмір у два рази.

1.3 Постановка завдань проектування

У зв’язку зі стрімким розвитком комп’ютерних мереж і, зокрема, мультисервісних комп’ютерних мереж, а також їх технологій, все більш актуальним стає завдання вивчення і вдосконалення методів, що використовуються протоколами транспортного рівня для передачі даних. В умовах сучасних високошвидкісних мереж починають позначатися деякі

обмеження, властиві протоколу TCP, а саме неоптимальний для різних сценаріїв алгоритм управління розміром перевантажувального вікна. Безумовно, розробці нових реалізацій методів, що використовуються у протоколі TCP, було приділено і зараз приділяється багато уваги.

Основними завданнями даної атестаційної роботи є наступні:

- розробка моделей сучасних реалізацій протоколу TCP та вибір моделі мережевого трафіку мультисервісної комп'ютерної мережі;
- розробка методу обміну даними транспортного рівня мультисервісної комп'ютерної мережі, що дозволяє підвищити ефективність процесу;
- програмна реалізація запропонованого методу у вигляді модифікації протоколу TCP
- створення сценарію моделювання;
- моделювання процесу втрат трафіку протоколом TCP і впливу на нього дисципліни управління перевантажувальних вікном.

2 ВИБІР ТА РОЗРОБКА МОДЕЛЕЙ, РОЗРОБКА МЕТОДУ ОБМІНУ ДАНИМИ ТРАНСПОРТНОГО РІВНЯ

2.1 Статистичні характеристики трафіку та його модель

Будь-який потік даних є випадковим процесом, а, отже, характеризується своїми статистичними властивостями. Найчастіше в якості таких властивостей використовуються щільність ймовірності надходження даних за період, щільність ймовірності інтервалів між надходженнями даних і автокореляційна функція.

Пуассонівський потік даних. Класичною моделлю трафіку в мережах передачі даних є Пуассонівський (найпростіший) потік. Він характеризується набором ймовірностей $P(k)$ надходження k повідомлень за часовий інтервал t:

$$P(k) = \frac{(\lambda t)^k}{k!},$$

де $k = 0, 1, \ldots$ – кількість повідомлень;

λ – інтенсивність потоку.

Інтервал часу вимірювання кількості повідомлень t і інтенсивність потоку λ є постійними величинами. Сімейство пуассонівських розподілів $P(k)$, в залежності від значення λ, зображено на рисунку 2.1. Більші значення λ відповідають більш широкому і симетричному графіку щільності ймовірності.

Середнє математичне очікування і дисперсія пуассонівського потоку рівні λt. Знаючи імовірність надходження даних за період, можна обчислити розподіл інтервалу τ між сусідніми подіями як

$$P(\tau) = \lambda * e^{-\lambda t}.$$

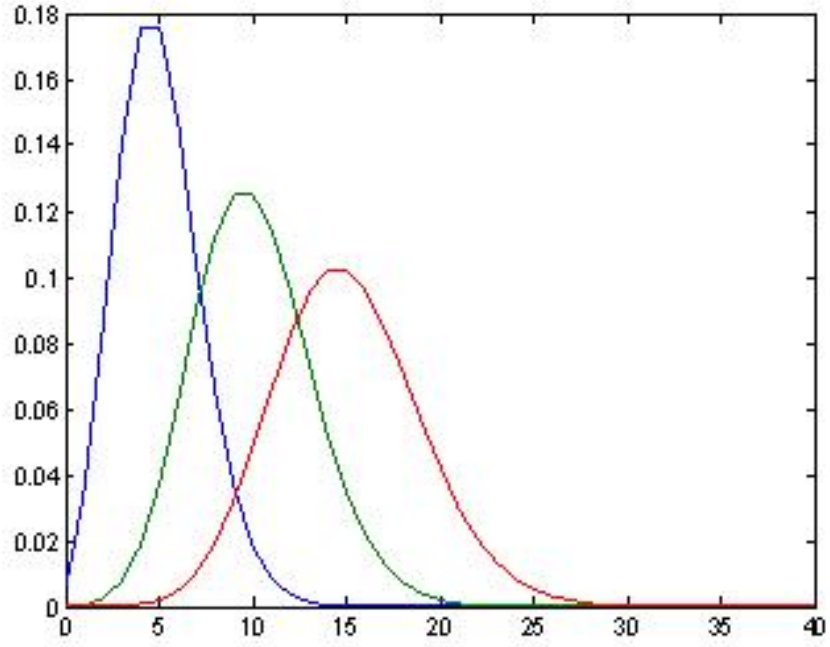

Рисунок 2.1 – Пуассонівські розподілення щільності ймовірностей

Звідси можна зробити висновок, що пуасонівський потік характеризується експоненціальним розподілом інтервалів між подіями. Основною властивістю пуассонівського потоку, що обумовлює його широке застосування при моделюванні, є властивість адитивності: результуючий потік суми пуассонівських потоків також є пуассонівським з сумарною інтенсивністю, що дорівнює

$$\lambda_{\Sigma} = \sum \lambda .$$

При моделюванні Пуассонівський потік можна отримати мультиплексуванням сукупності ON/OFF відправників, які є Марківськими процесами (рисунок 2.2).

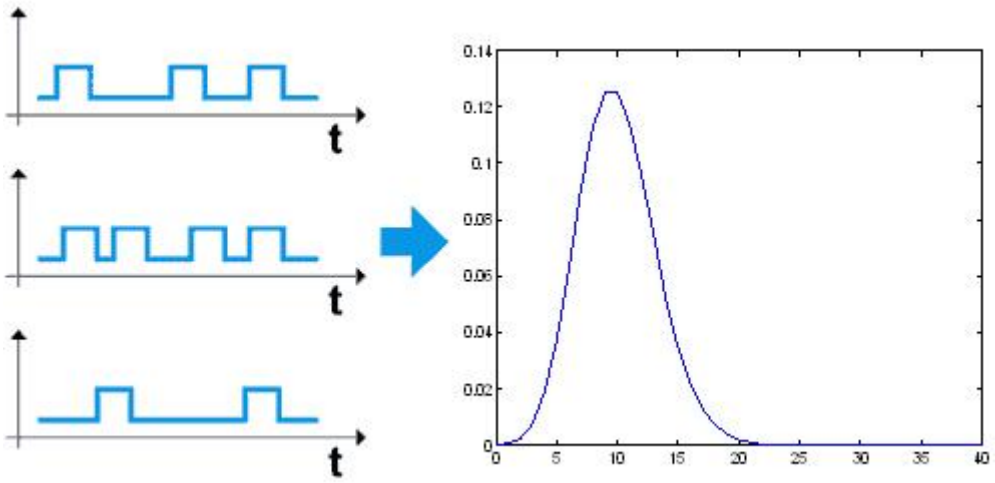

Рисунок 2.2 – Варіант отримання пуассонівського розподілу

Розподіл з важким хвостом. Вважається, що випадкова величина має розподіл з важким хвостом, якщо справедливий наступний вираз:

$$P(x) \approx x^{\alpha} \text{ при } x \to \infty .$$

Тобто хвіст розподілу загасає згідно степеневому закону, на відміну, наприклад, від Гаусівського розподілу з експоненціальним загасанням хвоста. Приклад розподілу з важким хвостом наведено на рисунку 2.3.

Сьогодні вважається, що мережний трафік у багатьох випадках найкраще описується саме розподілом з важким хвостом, наприклад розподілом Парето [5].

Автокореляційна функція (АКФ). Автокореляція – це кореляційний зв’язок між значеннями одного і того ж випадкового процесу в рознесені моменти часу. Автокореляційна функція характеризує такий зв’язок. У загальному випадку АКФ характеризує внутрішню залежність між часовим рядом і тим же рядом, але зсунутим на деякий проміжок (зрушення) часу, який називається лагом.

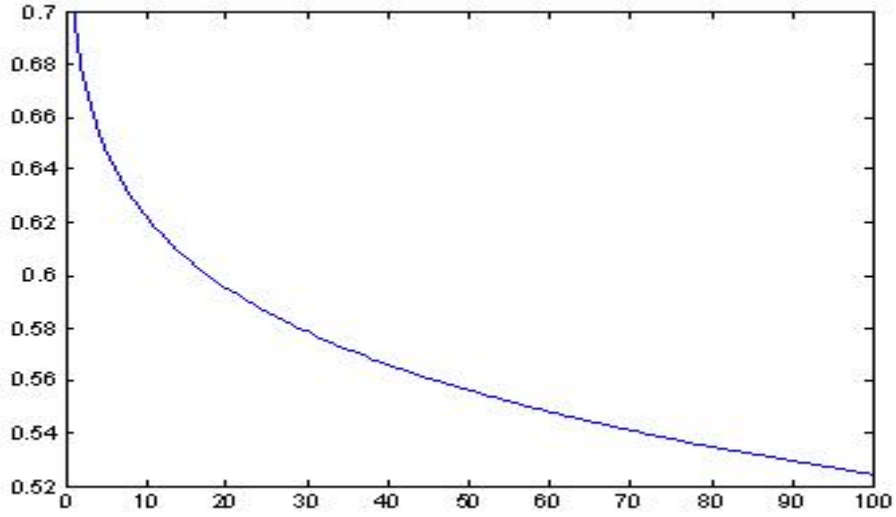

Рисунок 2.3 – Приклад розподілу з важким хвостом

Обчислення АКФ проводиться за наступною класичною формулою:

$$r(k) = \frac{\sum_i (X_i - \overline{X}) \cdot (X_{i+k} - \overline{X})}{\sum_i (X_i - \overline{X})^2},$$

де $\overline{X}$ – вибіркове середнє,

k = 0, 1, 2, … – лаг.

Повільно спадна залежність (ПСЗ). Вважається, що процес володіє повільно спадною залежністю, якщо він характеризується ПСЗ, яка зменшується гіперболічно (тобто за ступеневим законом) при збільшенні лага [5]. На противагу ПСЗ існує поняття швидко спадної залежності (ШСЗ), наведеної на рисунку 2.4.

З поняттям ПСЗ пов’язане одне з найважливіших властивостей прогнозування – тривала пам’ять, що характеризується залежністю поточних параметрів процесу від попередніх.

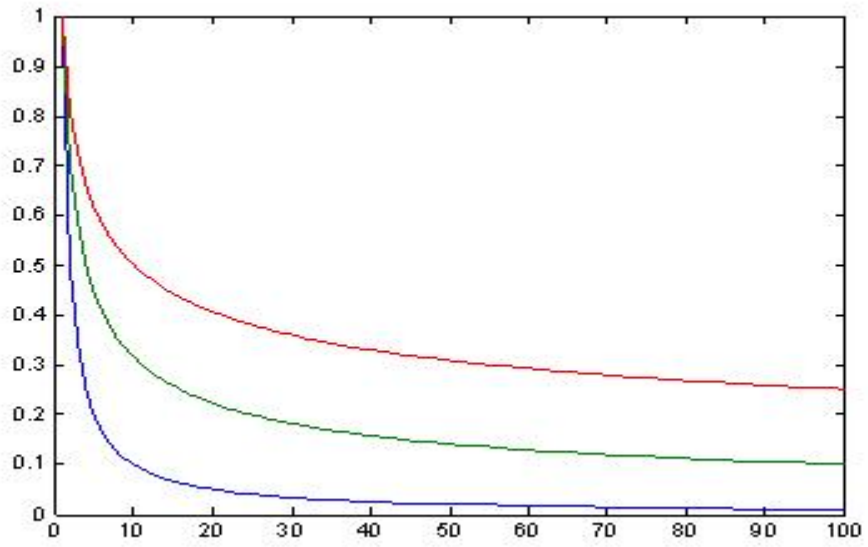

Рисунок 2.4 – Вид АКФ ШСЗ (нижня крива) і АКФ ПСЗ

Самоподібність трафіку. Численні сучасні дослідження трафіку великих (корпоративних і глобальних) комп'ютерних мереж свідчать про те, що він має властивість самоподібності [5]. Найпростішими самоподібними об'єктами є фрактали. Згідно з визначенням Мандельброта, фрактал є структурою, що складається з частин, які в якомусь сенсі подібні цілому. Тому самоподібні процеси часто називають фрактальними. Неформально самоподібний процес визначається як випадковий процес, статистичні характеристики якого виявляють властивості масштабування.

Строго самоподібний в широкому сенсі процес характеризується інваріантністю АКФ при зміні рівня агрегування за умови ПСЗ. На відміну від пуассонівських процесів, самоподібні характеризуються наявністю післядії, тобто імовірністю того, що надходження чергової події залежить не тільки від часу, але і від попередніх подій (передісторії). Це означає, що число поточних подій може залежати від числа попередніх подій у віддалені проміжки часу. Тому однією з основних властивостей самоподібного процесу є ПСЗ.

Найважливішим параметром, що характеризує ступінь самоподібності,

є параметр Херста, H, який визначається для часового ряду X_k, $k = 1, 2, K_N$ зі співвідношення

$$R / S = (aN)^H,$$

де R – розмах відхилення;

S – стандартне відхилення;

a – константа;

N – кількість членів часового ряду.

Використовуючи значення показника Херста H, можна виділити наступні типи випадкових процесів:

- $0 \le H < 0{,}5$: випадковий процес є антиперсистентним (або ергодичним рядом, який не володіє самоподібністю);

- $H = 0{,}5$: повністю випадковий ряд, аналогічний випадковим зсувам частки при класичному броунівському русі;

- $H > 0{,}5$: персистентний (або самопідтримуючий) процес, який володіє тривалою пам’яттю і є самоподібним.

Слід також зазначити, що самоподібний процес часто носить вибуховий характер, що виражається в імовірності наявності сплесків під час відносно низької швидкості надходження подій. Стосовно до трафіку, самоподобність виражається в незмінності поведінки при зміні часових масштабів спостереження і збереження схильності до сплесків при усередненні за шкалою часу.

Модель трафіку подається у вигляді ON/OFF-послідовності, яка складається з ON- і OFF-інтервалів, що чергуються. Значення ON-інтервалів відповідають тривалості періодів безперервної передачі даних в мережу відправником, а значення OFF-інтервалів відповідають тривалості періодів неактивності відправника. Побудова такої моделі полягає в знаходженні функцій щільності розподілу імовірностей, найбільш точно відповідних емпіричним розподілом тривалостей ON- і OFF-інтервалів.

2.2 Моделі реалізацій протоколу TCP

Протокол TCP Reno містить дві фази: фазу повільного старту і фазу запобігання перевантажень. При виявленні втрати пакету швидкість відправки даних відправником сповільнюється; передача кожного пакету вважається успішною лише при отриманні пакету підтвердження. Отримання повторюваного пакету підтвердження свідчить про те, що або прийняті сегменти переупорядковані, або сегмент втрачений. Якщо надходить деяка кількість повторюваних підтверджень, то це означає, що вже пройшло чимало часу, і навіть якщо сегмент був відправлений по більш довгому маршруту, то він вже повинен досягти одержувача: отже, існує велика імовірність його втрати.

В TCP Reno розмір вікна змінюється циклічно: він збільшується до тих пір, поки не відбудеться втрата пакета. При отриманні відправником пакету підтвердження в момент часу *t*, значення розміру перевантажувального вікна в наступний момент часу $s_w(t+t_A)$ обчислюється на основі поточного значення $s_w(t)$ відповідно до наступного виразу:

$$s_w(t+t_A)=\begin{cases} s_w(t)+1, & \text{якщо } s_w(t)<S(t); \\ s_w(t)+\dfrac{1}{s_w(t)}, & \text{якщо } s_w(t)\geq S(t), \end{cases}$$

де $S(t)$ – граничне значення розміру вікна (в пакетах), при досягненні якого протокол TCP переходить з фази повільного старту в фазу запобігання перевантаження.

Коли в результаті тайм-ауту виявляється втрата пакета, значення $s_w(t)$ та $S(t)$ обчислюються наступним чином:

$$s_w(t) = 1;$$

$$S(t) = (s_w(t))/2.$$

Коли протокол TCP виявляє втрату пакету згідно з алгоритмом швидкої повторної передачі, значення $s_w(t)$ та $S(t)$ змінюються інакше:

$$S(t) = \frac{s_w(t)}{2};$$

$$s_w(t) = S(t).$$

Після цього протокол TCP Reno переходить в фазу швидкого відновлення. Розмір вікна збільшується на один пакет при отриманні повторюваного підтвердження. При отриманні підтвердження для повторно відправленого пакета, значення $s_w(t)$ та $S(t)$ рівні. У разі тайм-ауту $S(t) = s_w(t)/2;\ s_w = 1$.

Фази швидкої повторної передачі і швидкого відновлення реалізовані спільно наступним чином.

При отриманні третього поспіль повторюваного пакету ACK, значення S встановлюється в половину поточного розміру перевантажувального вікна s_w, але не менше ніж в два сегменти. Потім повторно передається втрачений пакет. Значення s_w встановлюється в S плюс 3 розміри сегмента.

При отриманні іншого повторюваного пакету ACK, значення s_w збільшується на розмір сегмента і передається черговий пакет, якщо це можливо при новому значенні s_w.

Коли прибуває наступний пакет ACK, що підтверджує нові дані, значення s_w встановлюється в S (значення, встановлене на кроці 1). Цей ACK повинен бути підтвердженням повторної передачі з кроку 1 і повинен підтверджувати всі проміжні сегменти, відправлені після втраченого пакета і

перед отриманням першого повторюваного пакету ACK. Це дозволяє запобігти перевантаженню, оскільки протокол TCP знижує швидкість відправки пакетів в два рази після втрати пакета.

Протокол TCP Vegas використовує схему оцінки доступного з'єднанню розміру смуги пропускання, для чого постійно обчислює різницю між очікуваною і реальною швидкостями відправки пакетів в мережу. Таким чином, фазу запобігання перевантажень протоколу TCP Vegas можна записати так:

$$\nu_o = s_w(t)/t_m ,$$

де ν_o – очіувана швидкість відправки пакетів;

$s_w(t)$ – поточний розмір перевантажувального вікна;

t_m – мінімальне значення RTT даного з'єднання;

$$\nu_r = s_w(t)/t ,$$

де ν_r – реальна швидкість відправки пакетів;

t – поточне значення параметра RTT.

Для кожного прийнятого пакета підтвердження відправник проводить оцінку кількості пакетів, що знаходяться в черзі маршрутизатора як різницю

$$\delta = (\nu_o - \nu_r) \cdot t_m .$$

Знаючи значення параметра δ, розмір перевантажувального вікна s_w змінюється в такий спосіб:

$$s_w(t+1) = \begin{cases} s_w(t)+1, & \textit{якщо } \delta < \alpha; \\ s_w(t)-1, & \textit{якщо } \delta > \beta; \\ s_w(t), & \textit{інакше}, \end{cases}$$

де α – згладжувальний коефіцієнт у процедурі обчислення тривалості циклу таймера повторної передачі;

β – коефіцієнт дисперсії часу передачі пакету у встановленому з'єднанні.

2.3 Метод обміну даними

Метод обміну даними, що пропонується, грунтується на базовому методі протоколу TCP Vegas та стосується фази запобігання перевантажень. Послідовність єтапів розробленого методу наведено на рисунку 2.5.

Модифікація заснована на тому факті, що при передачах довгих файлів завантаження мережі в часі відносно стабільна і відправники здатні визначати свої оптимальні швидкості передачі. Оскільки кожен з відправників намагається підтримувати кількість пакетів в інтервалі значень від α до β в чергах на маршруті, то, при збільшенні кількості відправників в мережі, буде відбуватися лінійне збільшення часу постановки пакетів в чергу.

У методі, що використовується в протоколі TCP Vegas, значення α та β є константами, а завданням є таке управління розміром плаваючого вікна, при якому кількість пакетів, що знаходяться в буферах на маршруті з'єднання, буде знаходитися між значеннями α та β. Керуючись цим принципом, у пропонованій модифікації організована динамічна зміна значень параметрів.

Для збереження значення дійсної швидкості, досягнутої за попередній час RTT, і обчислення нової протоколом TCP Vegas кожен раз при успішній передачі пакетів, в метод введена одна нова змінна.

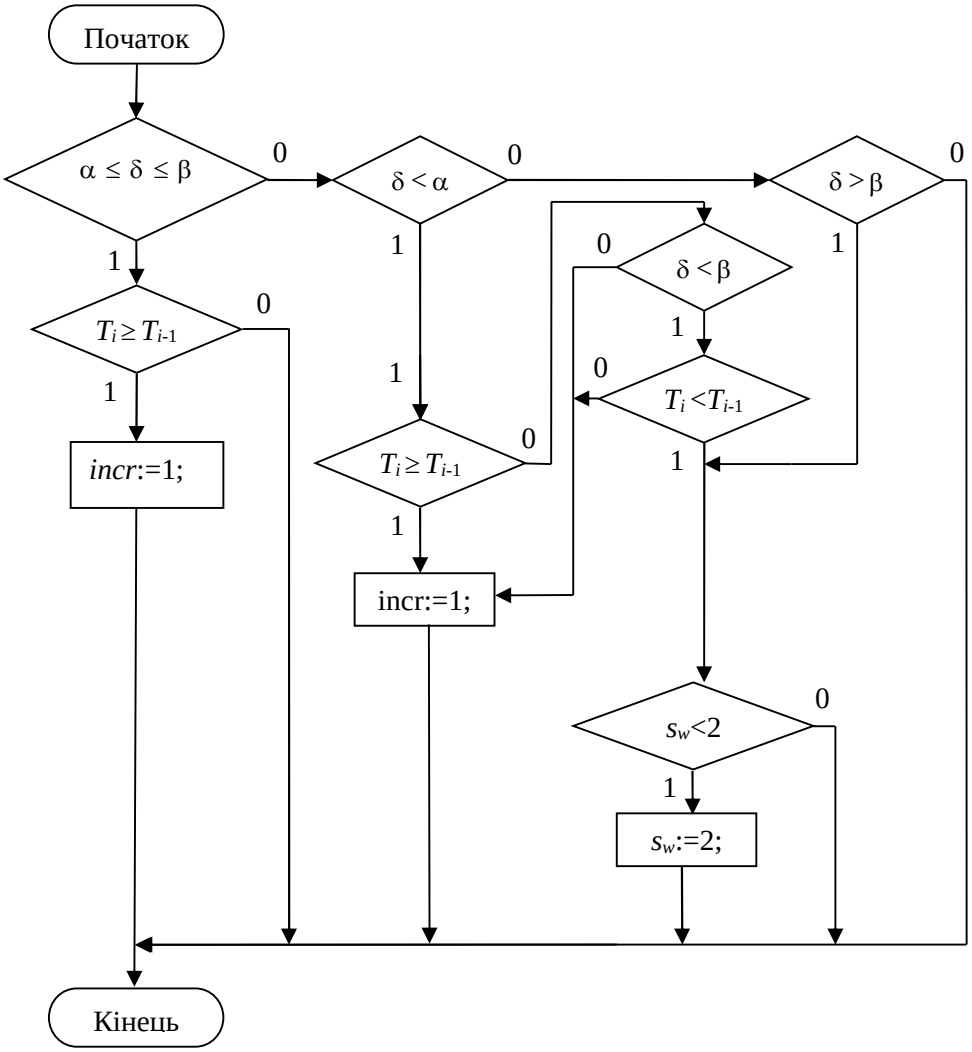

Рисунок 2.5 – Метод обміну даними

Роботу методу можна описати наступним чином. При відкритті з'єднання параметрам α та β надано початкові значення, α_0 та β_0, відповідно. Після успішної передачі кожного сегмента даних проводиться динамічна зміна значень α та β на одиницю відповідно до таких умов:

- якщо $T(t) > T(t\text{-}1)$ та $s_W(t) \leq s_W(t\text{-}1)$, то виконується інкремент значень α та β;
- якщо $T(t) \leq T(t\text{-}1)$ та $s_W(t) > s_W(t\text{-}1)$, то проводиться декремент значень α та β.

У разі виникнення втрат пакетів значення параметрів α та β скидаються в їх початкові, і виконання етапів проводиться заново.

Модифікований метод має наступні переваги:

- зміни потрібні тільки для відправника (не тягне змін в структурі мережі і її елементах);

- не проводиться додавання нових змінних (не виникає завдання їх коректної установки);

- результат роботи практично не відрізняється від роботи протоколу TCP Vegas в разі однорідного мережного середовища.

3 ОПИС ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ МОДЕЛЮВАННЯ І ВИКОНАННЯ МОДЕЛЮВАННЯ ЗАПРОПОНОВАНИХ РЕЗУЛЬТАТІВ

3.1 Опис інструментальних засобів, що використовуються для моделювання

Ускладнення сучасних телекомунікаційних технологій створює труднощі застосування аналітичних методів для оцінки характеристик проектованих систем і мереж, що обумовлює широке застосування імітаційного моделювання.

Системи імітаційного моделювання – це важливий інструмент для розробки комп’ютерних мереж. Вони відкривають розробникам можливості віртуальної побудови і перевірки мережі в роботі, а також швидко і ефективно створювати і налагоджувати нові протоколи і алгоритми роботи мережі. Це особливо важливо, оскільки витрати на розгортання реальних мереж залишаються досить високими.

В останні роки широкий розвиток отримали спеціалізовані системи імітаційного моделювання телекомунікаційних мереж, такі як GTNeS, OPNET, NISTNET, DummyNet, ModelNet, Ohio Network Emulator, ENDE, Emulab, EMPOWER, NSE, Vint/NS, NETWARS. Одне з головних місць серед них займає система моделювання «Network Simulator» (Ns2), яка лідирує за кількістю застосувань в реальних проектах.

NS представляє нове покоління систем моделювання, створених як розширення об’єктно-орієнтованої мови програмування. Застосування мови програмування дозволяє описувати заголовки пакетів і особливості взаємодії компонентів з достатнім ступенем деталізації і відповідно до стандартів, що підвищує реалістичність моделей.

Альтернативним є підхід, заснований на використанні мови кольорових мереж Петрі для представлення моделей, успішно застосованих для

дослідження мереж Ethernet, TCP/IP, MPLS і Bluetooth. З причини її широкого застосування, достовірності і підтвердженої адекватності побудованих моделей реальним телекомунікаційним процесам, ця система була обрана для моделювання процесів, досліджуваних в рамках даної атестаційної роботи.

3.2 Система моделювання комп'ютерної мережі Ns2

Ns2 є об'єктно-орієнтованим програмним хабезпеченням (ПЗ), ядро якого реалізовано на мові C++ [15, 16, 17]. Мова скриптів (або сценаріїв) Object-oriented Tool Command Language (OTcl) використовується в якості інтерпретатора. Ns2 повністю підтримує ієрархію класів C++ (звану компільованою ієрархією) і подібну ієрархію класів інтерпретатора OTcl (звану інтерпретованою ієрархією). Обидві ієрархії мають ідентичну структуру, тобто існує однозначна відповідність між класом однієї ієрархії і аналогічним класом іншої.

Використання двох мов програмування має певне підгрунтя [6, 7, 8]. З одного боку, для детального моделювання протоколів необхідно використовувати системну мову програмування, що забезпечує високу швидкість і здатність маніпулювати досить великими обсягами даних. З іншого боку, для зручності користувача і швидкості реалізації і модифікації різних сценаріїв моделювання, привабливіше використовувати мову програмування більш високого рівня абстракції [9]. Такий підхід є компромісом між зручністю використання і швидкістю. В Ns2 в якості системної мови використовується C++, що дозволяє забезпечити:

- високу продуктивність;
- роботу з пакетами потоку на низькому рівні абстракції моделі;
- модифікацію ядра Ns2 з метою підтримки нових функцій і протоколів.

В якості мови програмування високого рівня абстракції

використовується мова скриптів OTcl, що дозволяє забезпечити наступний ряд позитивних властивостей, притаманних мові Tcl/Tk (оскільки OTcl є об’єктно-орієнтованим розширенням мови Tcl):

- простота синтаксису;
- простота побудови сценаріїв моделювання;
- можливість з’єднання докупи блоків, виконаних на системних мовах програмування і досить просту маніпуляцію ними.

Об’єднання для спільного функціонування мов C++ і OTcl проводиться за допомогою TclCl (Classes Tcl). TclCl є інтерфейсом між об’єктами C++ та OTcl, якими користуються Ns2 та аніматор (Nam) [10].

Забезпечення сильної математичної підтримки дозволяє генерувати різні види трафіку, починаючи від найпростішого, що підкорюється пуассонівському закону, закінчуючи самоподібним трафіком. Гнучкість архітектури Ns2 дозволяє користувачеві реалізовувати власні математичні функції на C++. Реалізація моделі помилок дозволяє моделювати виникнення помилок на канальному рівні, тобто спотворення або втрату інформації. Моделювання виникнення помилки можливо як на рівні бітів, так і на рівні пакетів. Також можлива реалізація користувальницької моделі помилок.

Ns2 містить засіб анімації результатів моделювання Nam, реалізований з використанням Tcl/Tk, який забезпечує графічне відтворення проведеного експерименту: відображення топології мережі, анімація пакетів, вузлів, черг, а також різні можливості аналізу даних. В якості вхідних даних для Nam використовуються файли, записані в процесі функціонування Ns2 (моделювання мережі).

3.3 Мова OTcl

Для створення конкретного сценарію моделювання необхідно ознайомитися з синтаксисом і основними командами мови OTcl, які використовуються системою Ns2, а також розглянути основні елементи

створюваної мережі, такі як вузли та зв’язки. За допомогою описаних компонент відбувається створення заданої топології досліджуваної мережі.

У Tcl значення можуть бути збережені в змінних з метою подальшого використання в командах (приклад 3.1).

```
set a 5
set b [expr $a/5]
```

Приклад 3.1 – Присвоєння значень змінним

У першому рядку змінної *a* було присвоєно значення «5». У другому рядку результат команди «[expr $ a / 5]», яка обчислює вираз і повертає результат (1), використовується як аргумент для іншої команди, яка присвоює це значення змінній *b*. Знак «$» використовується для отримання значення, що міститься у змінній, а квадратні дужки служать вказівником підстановки команди.

Нові процедури можна визначати, використовуючи команду «proc». Перший аргумент визначає ім’я процедури, другий містить список аргументів, які їй передаються. Як приклад можна привести процедуру, яка обчислює суму двох чисел (приклад 3.2).

```
proc sum {a b}{
expr $a + $b
}
```

Приклад 3.2 – Визначення процедури

Наступна процедура дозволяє обчислити факторіал деякого числа (приклад 3.3).

Також, як список аргументів, можливо передавати порожній рядок. Однак в цьому випадку змінні, використовувані процедурою, повинні бути визначені як глобальні (приклад 3.4).

```
proc factorial a {
if {$a<=1}{
      return 1
}
expr $x*[factorial [expr $x-1]]
}
```

Приклад 3.3 – Приклад процедури

```
proc sum {}{
global a b
expr $a + $b
}
```

Приклад 3.4 – Визначення змінних

У Tcl файл може бути відкритий для читання за допомогою команди, що наведена на прикладі 3.5.

```
set testfile [open test.dat r]
```

Приклад 3.5 – Відкриття файлу для читання

Перший рядок файлу може бути збережено як список командою у відповідності до прикладу 3.6.

```
gets $testfile list
```

Приклад 3.6 – Збереження рядку файлу

Тепер можливо отримувати елементи списку за допомогою наступних команд; нумерація елементів починається з 0 (приклад 3.7).

```
set first [lindex $list 0]
set second [lindex $list 1]
```

Приклад 3.7 – Отримання елементів списку

Аналогічно, файл може бути записаний командою «puts» (приклад 3.8).

```
set testfile [open test.dat w]
puts $testfile “testi”
```

Приклад 3.8 – Запис файлу

Команда «exec» створює підпроцес і очікує його виконання. Використання такої команди схоже на передачу виконуваного рядка командному інтерпретатору («shell»). Наприклад, для видалення файлу можна додати рядок у відповідності до приклада 3.9.

```
exec rm $testfile
```

Приклад 3.9 – Видалення файлу

Команда «exec» зазвичай використовується, коли необхідно викликати один tcl-скрипт з іншого tcl-скрипта. Наприклад, для запуску скрипта «example.tcl» безліч разів із значенням параметра «test» що змінюється від 1 до 10, можливо використовувати рядки в іншому tcl-скрипті у відповідності до приклада 3.10.

```
for {set ind 1}{$ind<=10}{incr ind}{
set test $ind
exec ns example.tcl test
}
```

Приклад 3.10 – Запуск скрипта «example.tcl»

3.3.1 Створення топології

Для запуску сценарію моделювання необхідно створити мережну топологію. У Ns2 топологія складається з набору вузлів («nodes») і зв’язків між ними («links»).

Перш ніж поставити топологію, необхідно на початку скрипта створити новий об’єкт, використовуючи команду у відповідності до приклада 3.11.

```
set ns [new Simulator]
```

Приклад 3.11 – Створення нового об’єкту

Об’єкт моделювання має компоненти-функції, які можуть створювати вузли, зв’язки, агенти та інше. Такі основні функції можуть бути отримані через клас «Simulator». Коли використовуються функції, що належать цьому класу, команда починається з «$ ns», так як ns визначено в якості описателя («handle») даного об’єкта «Simulator».

Нові об’єкти вузлів створюються командою у відповідності до приклада 3.12

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
```

Приклад 3.12 – Створення вузлів

Метод класу «Simulator», званий «node» створює чотири вузли і призначає їм описувачі, відповідно, «n0», «n1», «n2», і «n3». Ці описувачі в подальшому можуть бути використані для посилань на вузли. Якщо вузлом є не маршрутизатор, а кінцевий вузол, то можуть бути встановлені агенти трафіку (TCP, UDP) і тип трафіку (FTP, CBR), що використовуються при моделюванні. Таким чином, відправники повинні бути послідовно прикріплені до агентів, а агенти до вузлів.

Найбільш часто використовуваними агентами в Ns2 є TCP і UDP. У першому випадку доступні такі реалізації протоколу:

- «Agent/TCP» – реалізація TCP Tahoe;
- «Agent/TCP/Reno» – реалізація TCP Reno;

- «Agent/TCP/Sack1» – реалізація TCP з функцією вибіркового підтвердження;

- «Agent/TCP/Vegas» – реалізація TCP Vegas.

У Ns2 реалізовані наступні типи додатків, які генерують відповідні типи трафіку:

- «Application/FTP» – генерує FTP трафік;

- «Application/Traffic/CBR» – генерує трафік з постійним бітрейтом;

- «Application/Traffic/Exponential» – генерує ON / OFF трафік (під час періодів неактивності трафік не надсилається, періодів активності – пакети генеруються з постійною інтенсивністю) з експоненційної тривалістю періодів;

- «Application/Traffic/Trace» – генерує трафік на основі трейс-файлу, в якому визначені розміри і часи генерації пакетів.

Також існує можливість генерувати трафік, використовуючи методи, що надаються класом «Agent». Наприклад, якщо необхідно надіслати дані через UDP, метод може бути використаний на рівні Tcl, якщо udp-агент був спочатку встановлений і призначений вузлу (приклад 3.13).

```
send (int nbytes)
```

Приклад 3.13 – Приклад генерації трафіку

Нижче наведено приклад створення CBR-трафіку і його прив’язки до вузла «n0», використовуючи UDP в якості транспортного протоколу (приклад 3.14).

```
set udp0 [new Agent/UDP]
$ns attach-agent $n0 $udp0
set cbr0 [new Application/Traffic/CBR]
$cbr0 attach-agent $udp0
$cbr0 set packet_size_ 1000
$udp0 set packet_size_ 1000
$cbr0 set rate_ 1000000
```

Приклад 3.14 – Створення CBR-трафіку

Застосунок FTP, що використовує протокол TCP як транспорт, може бути створений і прив’язаний до вузла «n1» вельми схожим шляхом (приклад 3.15).

```
set tcp1 [new Agent/TCP]
$ns attach-agent $n1 $tcp1
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$tcp1 set packet_size_ 1000
```

Приклад 3.15 – Створення застосунку FTP

Класи «UDP» і «TCP» є нащадками класу «Agent». Використовуючи вирази «[new Agent / TCP]» і «[new Agent / UDP]», властивості цих класів можуть бути передані новим об’єктам «udp0» і «tcp1», відповідно. Ці об’єкти можуть бути прикріплені до вузлів «n0» і «n1». Потім визначається застосунок і прив’язується до транспортного протоколу. Нарешті, встановлюються параметри трафіку.

У разі CBR, трафік може бути визначений параметрами «rate_» (або рівнозначним «interval_», визначальним проміжки між часом появи пакетів), «packetSize_» (що встановлює розмір пакета) і «random_». За допомогою параметра «random_» можна додати певну випадковість під час появи пакетів. Значення за замовчуванням – 0, позначає, що випадковість не вводиться.

Якщо потоки даних повинні завершуватися без додаткової обробки, відправники «udp» і «tcp» повинні бути з’єднані з одержувачами трафіку. Одержувач TCP визначений в класі «Agent/TCPSink», а одержувач UDP – в класі «Agent/Null».

Одержувач UDP може бути прив’язаний до вузла «n2» і з’єднаний з агентом «udp0» наступним чином (приклад 3.16).

```
set null [new Agent/Null]
$ns attach-agent $n2 $null
$ns connect $udp0 $null
```

Приклад 3.16 – Приєднання одержувача

Стандартний одержувач TCP трафіку, який відправляє один пакет підтвердження на один отриманий пакет даних, може бути прикріплений до вузла «n3» і з’єднаний з агентом «tcp1» наступними командами у відповідності до приклада 3.17.

```
set sink [new Agent/Sink]
$ns attach-agent $n3 $sink
$ns connect $tcp1 $sink
```

Приклад 3.17 – Стандартний одержувач TCP трафіку

Також існує більш короткий шлях завдання зв’язку між відправником і отримувачем, за допомогою команди у відповідності до приклада 3.18.

```
$ns create-connection <srctype> <src> <dsttype> <dst> <pktclass>
```

Приклад 3.18 – Зв’язок між відправником і отримувачем

Наприклад, для створення стандартного з’єднання TCP між вузлами «n1» і «n3» з ідентифікатором класу пакета «1», можна використовувати наступний код, у відповідності до приклада 3.19.

```
$ns create-connection TCP $n1 TCPSink $n3 1
```

Приклад 3.19 – Створення з’єднання TCP з ідентифікатором класу

Можна досить легко створювати кілька з’єднань протоколом TCP, використовуючи цю команду всередині циклу.

Для завершення створення топології необхідно задати з’єднання. У Ns2 вихідна черга («queue») вузла реалізована як частина зв’язку, тому, коли створюються з’єднання, від користувача вимагається визначити тип черги.

На рисунку 3.1 зображена структура симплексного зв’язку («simplex link») в Ns2. Для створення дуплексного зв’язку необхідно задати два симплексних, по одній на кожен напрямок. Усередині зв’язку пакет спочатку ставиться в чергу. Після цього він або скидається (передається агенту «Agent/Null»), або вибирається з черги і передається об’єкту затримки («Delay»), який моделює затримку в реальному з’єднанні. Нарешті, обчислюється і оновлюється параметр TTL.

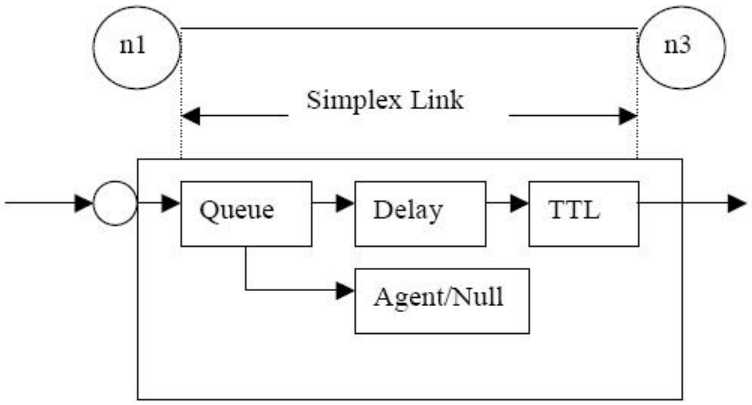

Рисунок 3.1 – Модель з’єднання в Ns2

Зв’язки задаються за допомогою наступної команди у відповідності до приклада 3.20.

```
$ns duplex/simplex-link <endpoint1> <endpoint2> <bandwidth> <delay> <queue-type>
```

Приклад 3.20 – Створення зв’язку

Наприклад, щоб створити дуплексний зв’язок з дисципліною управління чергою типу «DropTail» між вузлами «n0» і «n2», можна

використовувати наступний синтаксис у відповідності до приклада 3.21.

```
$ns duplex-link $n0 $n2 15Mb 10ms DropTail
```

Приклад 3.21 – Створення дуплексного зв’язку

Для створення симплексного зв’язку з дисципліною управління чергою типу «RED» між вузлами «n1» і «n2», використовується наступний синтаксис у відповідності до приклада 3.22.

```
$ns simplex-link $n1 $n3 10Mb 5ms RED
```

Приклад 3.22 – Створення симплексного зв’язку з дисципліною управління чергою типу «RED»

Значення розміру смуги пропускання вказуються безрозмірним числом або, використовуючи приставки «k» (кіло), «M» (мега), «b» (біт) і «B» (байт). Затримка також може бути виражена схожим чином, використовуючи приставки «m» (мілі) і «u» (мікро).

В системі Ns2 реалізовані різні алгоритми з управління чергою, однак, для створення топології, достатньою для досягнення поставлених цілей, будуть потрібні лише «DropTail» і «RED».

В ОС Linux, при обробці пакетів опції «SACK», стан кожного пакета в черзі повторної передачі може описуватися одним з чотирьох значень: «InFlight», «Lost», «Retrans» або «SACKed». Машина станів такої черги представлена на рисунку 3.2.

Пакет даних, який був відправлений у перший раз, ще не підтверджений і не втрачений, має стан «InFlight». При селективному підтвердженні за допомогою опції «SACK», він переходить в стан «SACK»; в стан «Lost» – при настанні тайм-ауту повторної передачі, або якщо наступний пакет зі станом «SACK» вже був переданий до нього за більш ніж

3 пакети (згідно з механізмом «FACK»). Якщо втрачений пакет був переданий повторно, він отримує стан «Retrans». Перехід зі стану «Retrans» в «Lost» відбувається при втраті повторно переданого пакета.

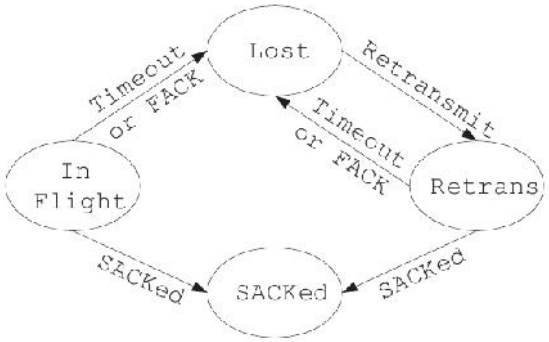

Рисунок 3.2 – Машина станів черги повторної передачі

3.3.2 Управління процесом моделювання

Після створення топології, конфігурації агентів і зв’язків, повинен бути заданий графік для початку, зупинки та інших подій при моделюванні.

Моделювання може бути розпочато і зупинено наступними командами у відповідності до приклада 3.23.

```
$ns at $simtime “finish”
$ns run
```

Приклад 3.23 – Створення графіку моделювання

Перша команда запускає в кінці моделювання процедуру «finish», а друга – безпосередньо починає моделювання. Процедура «finish» повинна бути визначена для скидання буфера трасування, закриття трейс-файлів і завершення програми за допомогою підпрограми виходу. Також вона може додатково запускати графічний аніматор мережі («Nam»), проводити обробку даних і будувати до них необхідні графіки. Така процедура повинна містити

наступні елементи у відповідності до приклада 3.24.

```
proc finish {}{
global ns trace_all
$ns flush-trace
close $trace_all
exit 0
}
```

Приклад 3.24 – Процедура «finish»

Решта подій, такі як час початку і зупинки роботи клієнтів, можна задати наступним чином (приклад 3.25).

```
$ns at 0.0 “cbr0 start”
$ns at 50.0 “ftp1 start”
$ns at $simtime “cbr0 stop”
$ns at $simtime “ftp1 stop”
```

Приклад 3.25 – Створення графіку роботи клієнтів

Якщо визначені власні процедури, то можна зробити так, щоб процедура запускалася, наприклад, кожні 5 секунд (приклад 3.26).

```
proc example {}{
global ns
set interval 5
…
$ns at [expr $now + $interval] “example”
}
```

Приклад 3.26 – Запуск процедури за розкладом

3.4 Використання генератора випадкових значень

Для створення генератора випадкових значень в Ns-2 існує клас під назвою «RNG», який містить в собі виконання багаторазового рекурсивного генератора.

Генератор забезпечує $1,8*10^{19}$ незалежних потоків випадкових значень, кожен з якого складається з $2,3*10^{15}$ підпотоків. Кожен підпотік має період в $7,6*10^{22}$ випадкових значень (до їх повторення). Повний період генератора дорівнює $3,1*10^{57}$ значень [6]. На рисунку 3.3 зображена структура потоків генератора випадкових значень.

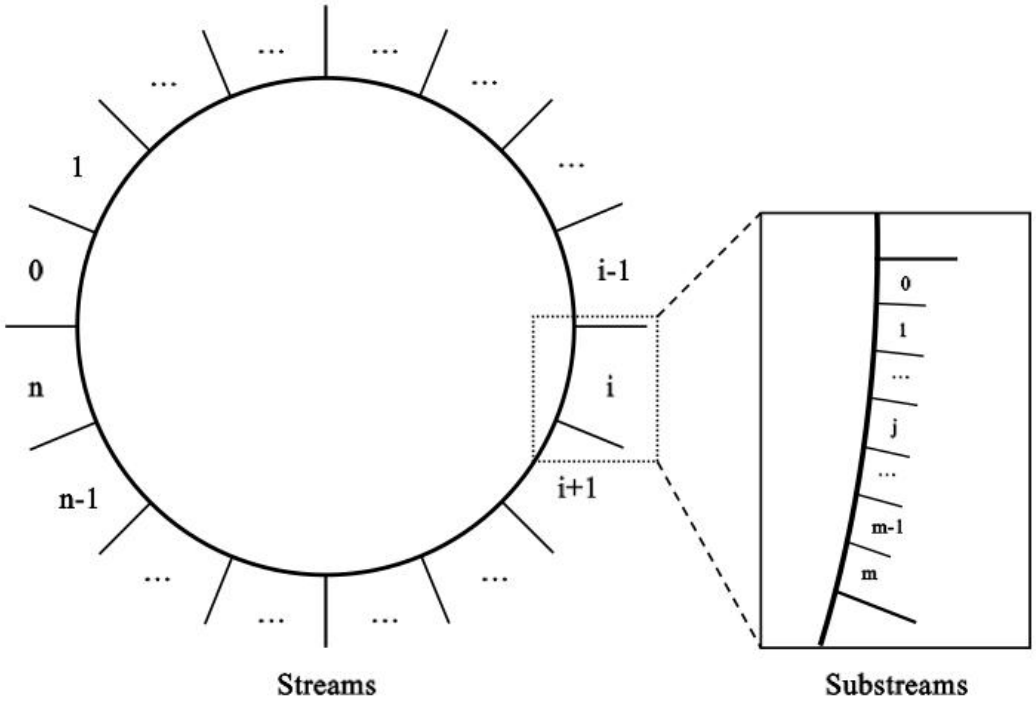

Рисунок 3.3 – Структура потоків і підпотоків генератора випадкових значень

Значення параметра «RNG» в момент запуску процесу моделювання, як правило, відомо. Якщо при моделюванні використовується кілька випадкових змінних, то кожна з них повинна використовувати окремий об’єкт «RNG». При створенні нового об’єкта «RNG», він автоматично встановлюється в початок наступного залежного потоку випадкових чисел. Подібним чином ця структура підтримується для $1,8*10^{19}$ випадкових змінних.

Нижче наведено перелік основних команд, які використовуються при моделюванні (приклади 3.27-3.34).

```
set rng [new RNG]
```

Приклад 3.27 – Створення нового генератора випадкових чисел;

```
$rng seed <0 or n>
```

Приклад 3.28 – Встановлення значення RNG: 0 відповідає евристичній установці RNG, інакше RNG встановлюється в значення <n>

```
$rng next-random
```

Приклад 3.29 – Повернення наступного випадкового значення з RNG

```
$rng uniform <a> <b>
```

Приклад 3.30 – Повернення значення для рівномірного розподілу в інтервалі [a, b]

```
$rng integer <k>
```

Приклад 3.31 – Повернення цілого числа з рівномірного розподілу в інтервалі [0, k-1]

```
$rng exponential
```

Приклад 3.32 – Повернення значення, яке має експоненціальне поширення із середньою величиною 1

```
set rv [new Randomvariable/<type of random-variable>]
```

Приклад 3.33 – Створення об’єкта генерації випадкової змінної з певним розподілом

```
$rv use-rng <rng>
```

Приклад 3.34 – Метод зв’язку «RandomVariable» з «RNG»

Нижче наведено приклад створення генератора випадкових чисел з експоненціальним розподілом (приклад 3.35).

```
set rng [new RNG]
$rng seed 0
set e [new RandomVariable/Exponential]
$e use-rng $rng
```

Приклад 3.35 – Створення генератора випадкових чисел з експоненціальним розподілом

3.5 Алгоритми управління перевантаженнями, вбудовані в ядрі ОС Linux

Для запуску алгоритму управління перевантаженням протоколу TCP, що знаходиться безпосередньо в ядрі ОС Linux, існує спеціальний агент, наприклад «Agent/TCP/Linux» (відправник TCP з підтримкою опції «SACK»).

Існує можливість поновлення або імпорту користувачами нових алгоритмів управління перевантаженнями у вихідний код ядра ОС Linux. Модулі протоколів TCP ОС Linux представлені у вихідних кодах Ns-2. Користувачі можуть вибрати різні алгоритми управління перевантаженнями і різні параметри TCP Linux для цього агента. Для роботи з агентом «TCP/Linux» необхідно, щоб одержувач підтримував опцію «SACK» [10]. Реалізація такого агента викликає схеми управління перевантаженнями ядра ОС Linux для зміни їх параметрів (розмір вікна, поріг повільного старту та інше).

Для досягнення результатів моделювання, близьких до ОС Linux, агент змінює значення за замовчуванням наступних параметрів:

- «Agent/TCP/Linux set maxrto_ 120»;

- «Agent/TCP/Linux set ts_resetRTO_ true»;
- «Agent/TCP/Linux set delay_growth_ false».

На рисунку 3.4 показана структура «TCP/Linux» в Ns2, де: жовті блоки від існуючих кодів (один від Ns2, інший від ядер ОС Linux); «LinuxTcpAgent» є підкласом «TcpAgent» в Ns2; «ScoreBoard1» – незалежний клас для виявлення втрат; «Ns-linux-util.h» і «.cc» - інтерфейси між NS-2 і ОС Linux; «Ns-linux-c.h» – спрощене середовище для кодів Linux.

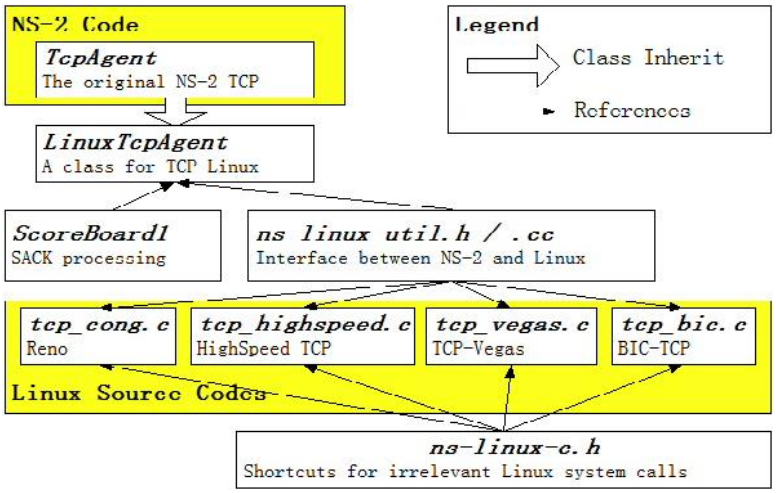

Рисунок 3.4 – Структура TCP/Linux в Ns2

3.6 Опис завдань моделювання

Для вирішення завдань, поставлених у цій атестаційній роботі, необхідно провести моделювання різних реалізацій протоколу TCP, включаючи запропоновану модифікацію, і вивчити різні алгоритми управління перевантаженнями. Для проведення моделювання та отримання результатів необхідно побудувати топологію, при якій сумарна пропускна здатність з'єднань між відправниками і маршрутизаторами на вході каналу перевищувала б пропускну здатність каналу [11, 12]. Пропускна здатність такого каналу буде визначальною для кожного з'єднання що проходить через нього, отже, такий канал є критичною ділянкою.

Наступним етапом є створення сценарію моделювання, що описує потрібну топологію і дозволяє отримати необхідні залежності: залежності розміру вікна від часу моделювання і залежності ефективності (в термінах пропускної здатності) від часу моделювання. Сценарій повинен бути написаний таким чином щоб легко можна було вносити зміни такі як: збільшення числа відправників, заміна реалізації протоколу, зміна пропускної здатності каналів, затримок, механізмів обслуговування черг, максимального розміру переданих пакетів даних та інше.

Необхідно проаналізувати отримані результати і зробити висновки щодо ефективності запропонованої модифікації протоколу TCP, а також застосування різних сучасних реалізацій протоколу TCP для конкретних сценаріїв.

3.7 Розробка топології

Для досягнення поставлених завдань моделювання, необхідно створити топологію, в якій би був єдиний маршрут від відправника до одержувача, що включає критичну ділянку і використовує маршрутизатори.

Топологія, яка задовольнить необхідним умовам, зображена на рисунку 3.5. За допомогою «s1», …, «sn» позначені вузли, які є відправниками трафіку, а «n2» та «n3» – маршрутизатори. Максимальна швидкість передачі даних між вузлами і маршрутизатором дорівнює 1000 Мбіт/с. Час передачі пакета даних між вузлами і маршрутизатором буде вибиратися випадковим чином з інтервалу від 1 до 3 мс.

Канал між маршрутизаторами («n2»-«n3») є загальним для всіх з’єднань критичною ділянкою та має смугу пропускання 100 Мбіт/с, що набагато менше сумарної пропускної здатності каналів, що з’єднують маршрутизатор з відправниками. Затримка в цій лінії зв’язку дорівнює 2 мс. У маршрутизаторах, як механізм обслуговування, застосовується механізм «DropTail», при якому задається максимальний розмір черги, а знову

прибуваючий пакет поміщається в кінець черги, і якщо довжина черги досягла певного значення – пакет відкидається. Передача трафіку здійснюється від відправників трафіку в підмережу маршрутизатора «n3». Передбачається, що файли передаються протоколом FTP.

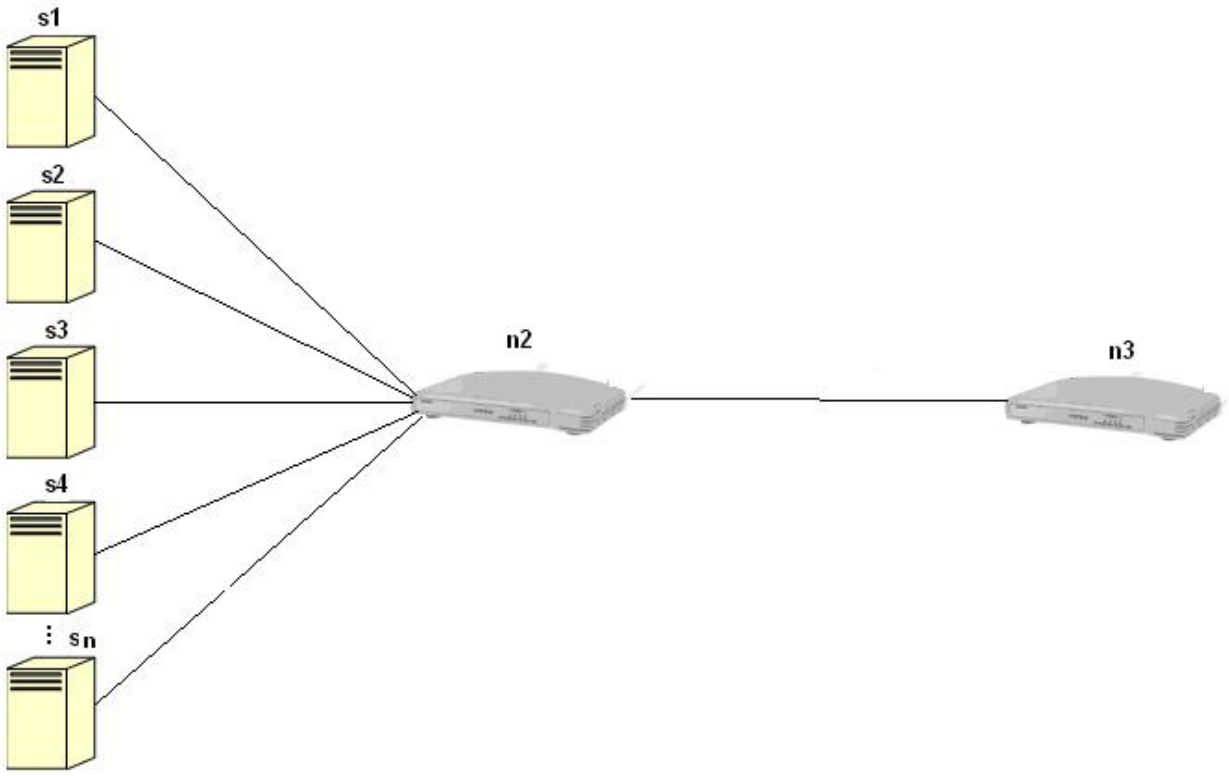

Рисунок 3.5 – Топологія мережі з однією критичною ділянкою

3.8 Опис сценарію моделювання.

Опис сценарію моделювання містить наступну послідовність дій.

Необхідно створити файл (наприклад, «random.tcl»), який буде містити сценарій моделювання, заснований на методах, розглянутих в попередньому розділі.

Створюємо новий об’єкт моделювання (приклад 3.36).

```
set ns [new Simulator]
```

Приклад 3.36 – Створення нового об’єкту моделювання

Описуємо змінні, які будуть використовуватися протягом усього моделювання (приклад 3.37).

```
set NumbSrc 5
set Duration 20
```

Приклад 3.37 – Опис змінних

Потім визначаємо процедуру «finish», в якій описуються дії після закінчення моделювання, такі як закриття файлів з даними та скидання буфера трасування. Текст процедури наведено нижче (приклад 3.38).

```
proc finish {} {
    global ns nf tf
    $ns flush-trace
    close $nf
    close $tf
    exit
}
```

Приклад 3.38 – Процедура «finish»

Тепер необхідно створити задану топологію. Критична ділянка складається з двох маршрутизаторів («n2» і «n3») і з’єднуючого їх каналу. Вузли («s1», «s2», «s3», «s4», «s5») є відправниками трафіку. Нижче наведено сценарій створення вузлів і з’єднань між ними (приклад 3.39).

Також визначаємо максимальний розмір черги маршрутизатора (приклад 3.40).

Наступним кроком є оголошення генератора випадкових чисел і визначення випадкових змінних. Випадковими змінними буде затримка «RVdly», момент часу початку передачі даних «RVstart» і припинення

передачі «RVdur» (приклад 3.41).

```
set n2 [$ns node]
set n3 [$ns node]
$ns duplex-link $n2 $n3 100Mb 2ms DropTail
#Source nodes
for {set j 1} {$j<=$NumbSrc} { incr j } {
     set S($j) [$ns node]
}
#Links between source and bottleneck
for {set j 1} {$j<=$NumbSrc} { incr j }  {
    $ns duplex-link $S($j) $n2  100Mb $dly($j)ms DropTail
    $ns queue-limit $S($j) $n2 20
}
```

Приклад 3.39 – Створення вузлів і з’єднань між ними

```
#Set Queue Size of link (n1-n2) to 20
$ns queue-limit $n1 $n2 20
```

Приклад 3.40 – Визначення максимального розміру черги

```
set rng [new  RNG]
$rng seed 0
# parameters for random variables for delays
set RVdly [new RandomVariable/Uniform]
$RVdly set min_ 1
$RVdly set max_ 3
$RVdly use-rng $rng
# parameters for random variables for beginning of ftp connection
set RVstart [new RandomVariable/Uniform]
$RVstart set min_ 0
$RVstart set max_ 7
$RVstart use-rng $rng
set RVdur [new RandomVariable/Uniform]
$RVdur set min_ 8
$RVdur set max_ 20
$RVdur use-rng $rng
```

Приклад 3.41 – Оголошення генератора випадкових чисел

Нижче наведено фрагмент скрипта (приклад 3.42), що обчислює випадкові величини для кожного з потоків: затримки (від 1 до 3 мс), початку передачі (від 0 до 7 с), і припинення передачі (від 8 до 20 с), а також здійснює їх подальший запис в певні змінні.

```
for {set i 1} {$i<=$NumbSrc} { incr i }  {
   set  startT($i)     [expr   [$RVstart value]]
   set dly($i)   [expr [$RVdly value]]
   set dur($i)   [expr [$RVdur value]]
   puts  $param "dly($i) $dly($i) ms"
   puts  $param "startT($i) $startT($i) sec"
   puts  $param "dur($i) $dur($i) sec"
}
```

Приклад 3.42 – Обчислення випадкових величин

Для створення з’єднання протоколом TCP необхідно задати агенти і одержувачі трафіку. Приклад створення зв’язування вузла з агентом (відправник трафіку) і вузла з одержувачем (одержувач трафіку) наведено нижче (приклад 3.43).

```
#TCP Source
for {set j 1}  {$j<=$NumbSrc} { incr j }  {
    set  tcp_src($j) [new Agent/TCP/Reno]
}
for {set j 1} {$j<=$NumbSrc} { incr j  }  {
    set   tcp_snk($j)  [new Agent/TCPSink]
}
for {set j 1} {$j<=$NumbSrc} { incr j } {
    $ns attach-agent $S($j) $tcp_src($j)
    $ns attach-agent $n3 $tcp_snk($j)
    $ns connect $tcp_src($j) $tcp_snk($j)
}
```

Приклад 3.43 – Створення зв’язування вузлів

Подальшим етапом є опис процесу відправки даних, для чого створений агент приєднується до вже існуючого відправника (приклад 3.44).

```
for {set j 1} {$j<=$NumbSrc} { incr j } {
    set e$i [new Application/Traffic/Exponential]
$e set packetSize_ 210
$e set burst_time_ 500ms
$e set idle_time_ 500ms
$e set rate_ 100k
```

Приклад 3.44 – Приєднання агента

Наступна процедура «plotWindow» (приклад 3.45) записує значення

модельованих параметрів до файлу з періодом дискретизації, рівним 0,03 с. Для цього необхідно визначити поточний час моделювання змінної «set now [$ns now]» і розмір вікна змінної «set cwnd [$tcpSource set cwnd_]». Після цього проводиться запис в файл значень «час – розмір вікна». Наступний раз процедура запускається через заданий період дискретизації і так до кінця моделювання.

```
proc plotWindow {tcpSource file k} {
    global ns
    set time 0.03
    set now [$ns now]
    set cwnd [$tcpSource set cwnd_]
    puts $file "$now $cwnd"
    $ns at [expr $now+$time] "plotWindow $tcpSource $file $k"
}
```

Приклад 3.45 – Процедура «plotWindow»

Фрагмент сценарію, представлений нижче (приклад 3.46), здійснює запуск процедури «plotWindow», а також запуск і припинення передачі даних відправником в випадкові моменти часу.

```
for {set i 1} {$i<=$NumbSrc} { incr i } {
$ns at 0.1 "plotWindow $tcp_src($i) $window($i) $j"
    $ns at $startT($i) "$e($i) start"
    $ns at $dur($i) "$e($i) stop"
}
$ns at [expr  $Duration] "finish"
```

Приклад 3.46 – Запуск процедури «plotWindow»

Запуск моделювання здійснюється командою у відповідності до прикладу 3.47.

```
$ns run
```

Приклад 3.47 – Запуск моделювання

Запуск створеного сценарію здійснюється командою у відповідності до прикладу 3.48.

```
ns2 random.tcl
```

Приклад 3.48 – Запуск сценарію

Для побудови графіка залежності розміру перевантажувального вікна від часу використовується команда у відповідності до прикладу 3.49.

```
xgraph -nb -bg white -lw 1 -zg black -x t -y wnd -t TCP_Model win1 win2 win3 win4 win5
```

Приклад 3.49 – Побудова графіка

3.9 Результати моделювання

Моделювання почнемо зі сценарію, коли одночасно існує 5 потоків протоколом TCP Vegas. Смуга пропускання критичної ділянки – 100 Мбіт/с, механізм обслуговування черги маршрутизатора – «DropTail», розмір черги – 20 сегментів. З рисунка 3.6 видно що мережа, що складається тільки з відправників TCP Vegas, може функціонувати без періодичних втрат пакетів, оскільки кожен з відправників намагається підтримувати кількість пакетів в черзі в інтервалі значень від α до β.

Протокол TCP Vegas реалізує дисципліну управління розміром вікна, засновану на використанні часових характеристик передачі пакета в установленому з’єднанні, і забезпечує рівномірність розподілу доступних ресурсів мережі і відсутність значних флуктуацій розміру перевантажувального вікна.

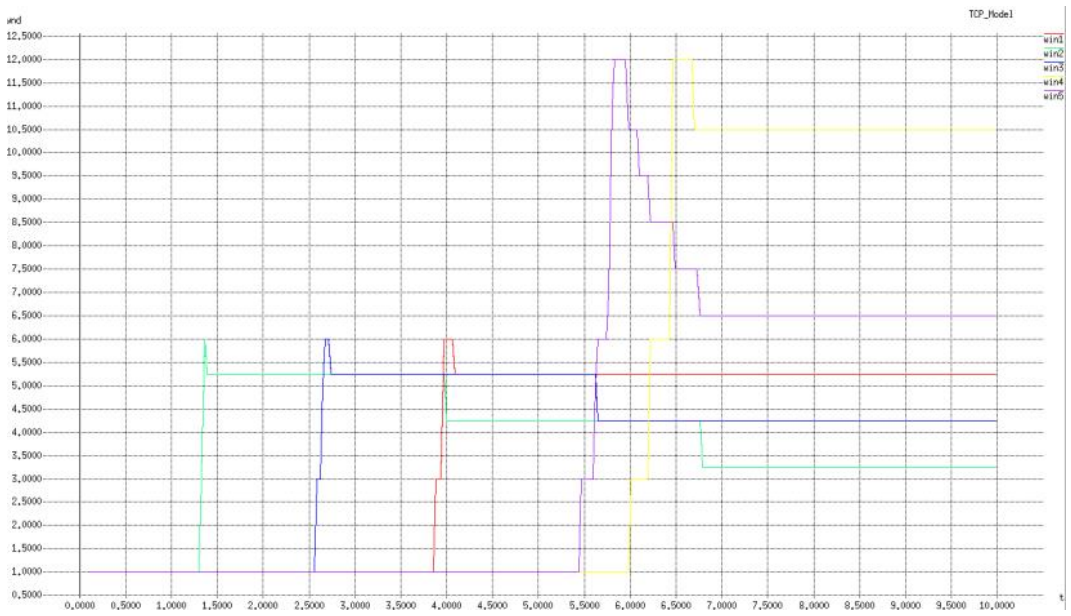

Рисунок 3.6 – Графік залежності розміру перевантажувального вікна від часу при одночасному існуванні 5 потоків протоколом TCP Vegas

На рисунку 3.7 наведені результати моделювання для сценарію, коли існує 5 з’єднань протоколом TCP Reno, смуга пропускання критичної ділянки дорівнює 100 Мбіт/с, механізм обслуговування черги маршрутизатора – «DropTail» і розмір такої черги дорівнює 20 сегментам.

Як видно з рисунку, після запуску 5 потоків у випадковий час в інтервалі від 0 до 7 секунд, протокол TCP Reno починає передачу в режимі повільного старту і, оскільки в каналі знаходяться тільки з’єднання TCP Reno, розмір перевантажувальних вікон з’єднань збільшується до практично максимального значення, займаючи весь доступний розмір смуги пропускання критичної ділянки. Лише після досягнення практично максимального розміру смуги пропускання, протокол переходить в режим запобігання перевантаження. Емпіричним шляхом встановлено, що для такого сценарію, якщо розмір черги буде менше 12 сегментів, то протокол TCP Reno не досягне таких великих значень пропускної здатності, що обумовлено алгоритмом управління перевантаженнями і його реакцією на втрату пакетів. Це показано на рисунку 3.8.

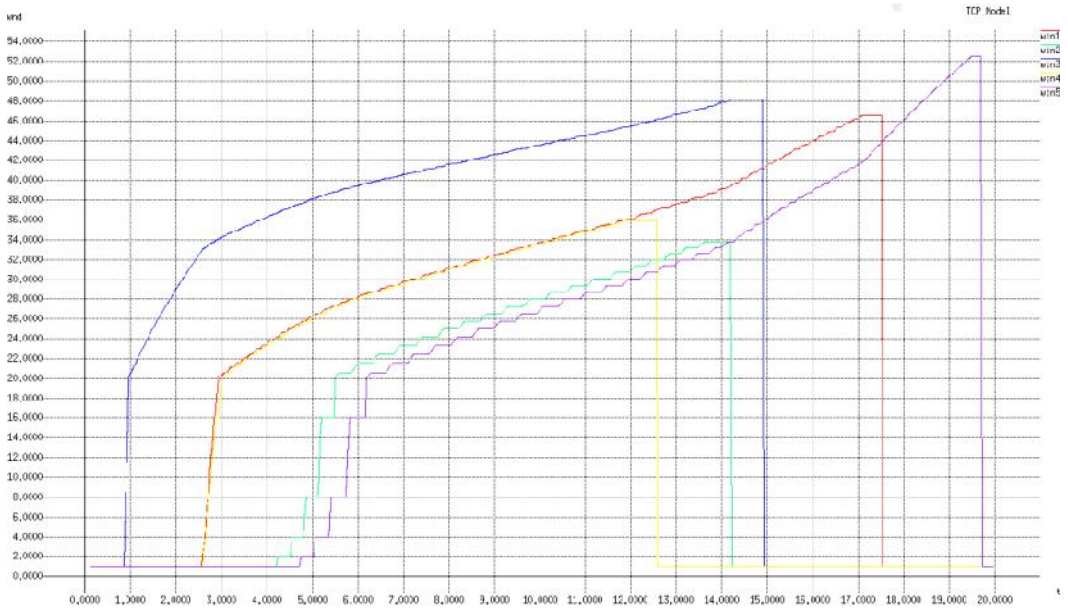

Рисунок 3.7 – Графік залежності розміру перевантажувального вікна від часу при існуванні 5 потоків протоколом TCP Reno

Розглянемо випадок, коли одночасно існують з'єднання протоколами TCP Reno і TCP Vegas при смузі пропускання критичної ділянки в 100 Мбіт/с, механізмі обслуговування черги маршрутизатора «DropTail» і розмірі його черги в 20 сегментів.

З рисунку 3.9 видно зниження пропускної здатності протоколу TCP Vegas, що пояснюється низьким рівнем агресивності протоколу, що використовує динаміку зміни значення RTT в якості запобіжного перевантаження в мережі. В інтервалі часу 1-5 секунд, коли в мережі не присутні потоки трафіку іншими протоколами і мережне середовище є однорідним, не спостерігається різких флуктуацій розміру перевантажувального вікна.

При досить невеликих розмірах черг (порядку декількох сегментів) протокол TCP Reno починає поводитися інакше, що показано на рисунку 3.10. Як тільки з'являється потік TCP Vegas, він збільшує своє перевантажувальне вікно, не даючи шансів для збільшення вікна TCP Reno. Можна зробити висновок, що TCP Reno потрібно певний розмір буфера, для

того щоб оцінити доступну пропускну здатність. Без мінімально необхідного розміру буфера його робота помітно погіршується.

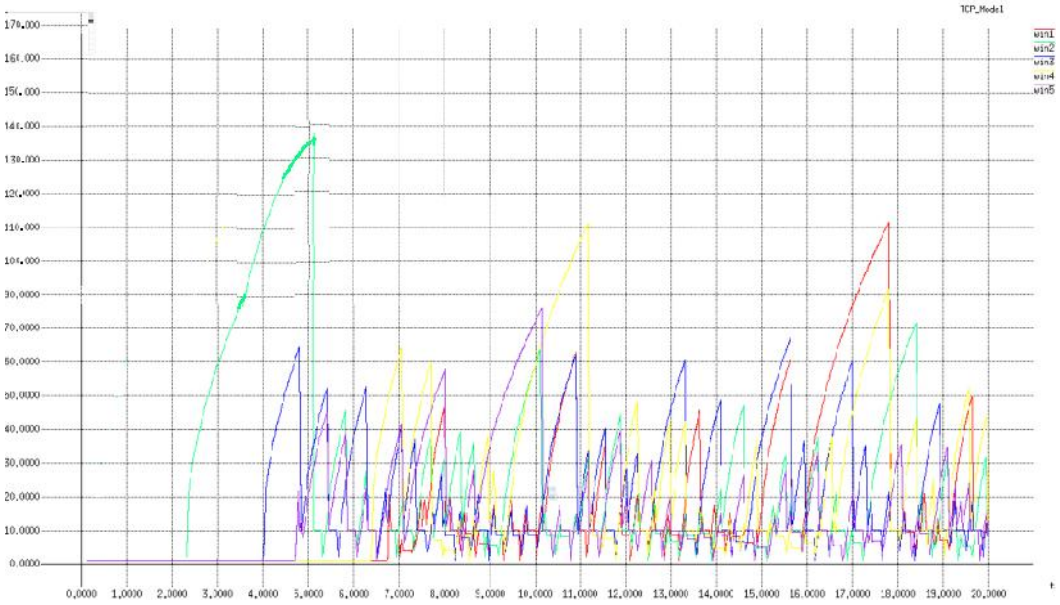

Рисунок 3.8 – Графік залежності розміру перевантажувального вікна від часу при розмірі черги в 10 сегментів і існування 5 потоків протоколом TCP Reno

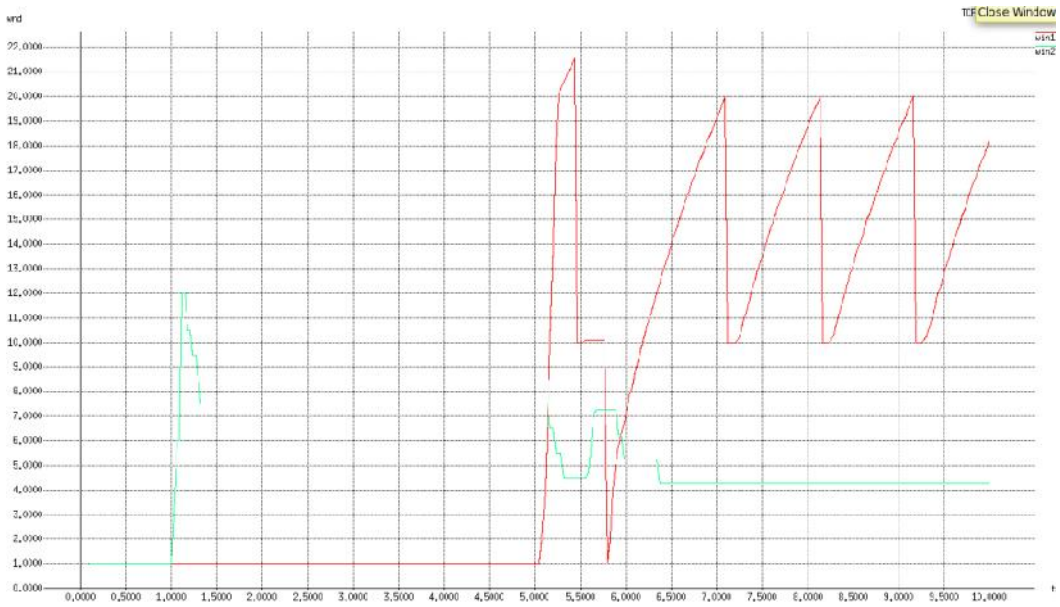

Рисунок 3.9 – Графік залежності розміру перевантажувального вікна від часу при існуванні 2 потоків (TCP Reno та TCP Vegas)

На рисунку 3.11 показаний графік залежності розміру перевантажувального вікна від часу при існуванні в каналі 5 потоків, з яких 4 потоки протоколом TCP Reno і один потік протоколом TCP Vegas. Смуга пропускання критичного ділянки – 100 Мбіт/с, розмір черги – 20 сегментів.

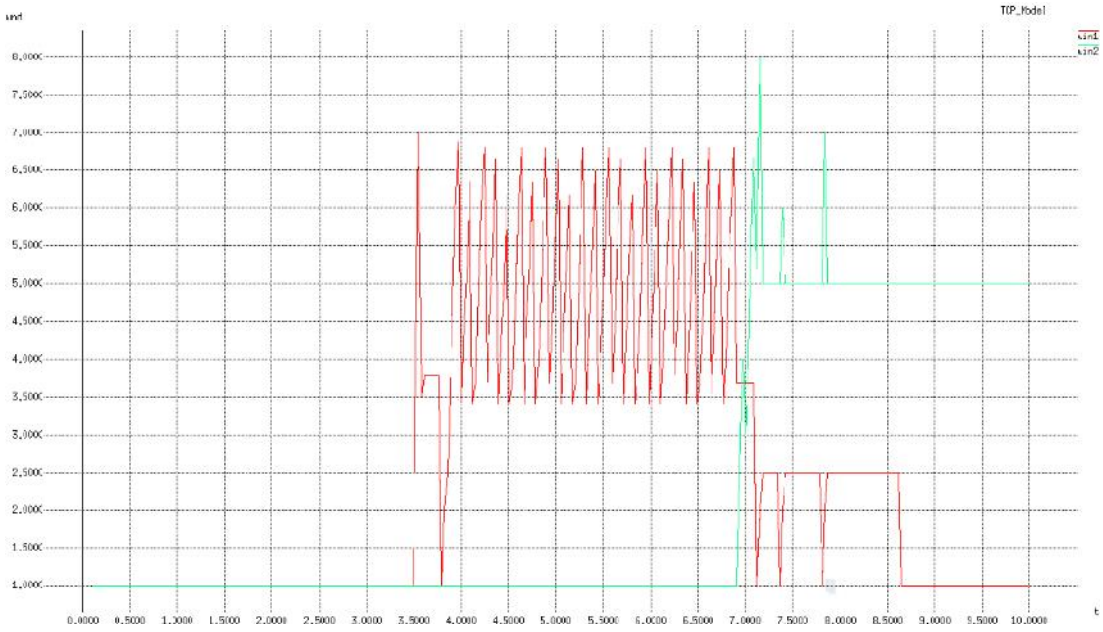

Рисунок 3.10 – Графік залежності розміру перевантажувального вікна від часу при невеликому розмірі черги і існуванні 2 потоків (Reno та Vegas)

Можна зробити висновок, що велика кількість фонового трафіку (з'єднання протоколом TCP Reno по відношенню до TCP Vegas), який виявляє значно більшу агресивність на початкових стадіях свого існування, негативним чином позначається на якісних показниках протоколу TCP Vegas, що також підтверджується авторами в джерелах [13, 14].

Аналогічна ситуація проявляється і на рисунку 3.12, де існують 4 відправника TCP Vegas і 1 відправник TCP Reno.

Далі наведені результати моделювання сценарію, коли на критичній ділянці встановлені два з'єднання протоколом TCP Reno, а також в інтервалі часу 0-200 секунд присутній фоновий трафік (рисунок 3.13). З рисунка видно, що миттєве значення пропускної здатності кожного з протоколів нестабільно

навіть при відсутності фонового трафіку, що можна сказати і про значні флуктуації розміру плаваючого вікна на рисунку 3.8. Рівномірність розподілу ресурсів каналу знаходиться на достатньому рівні. Поведінка обох з'єднань протягом всього часу моделювання ідентична.

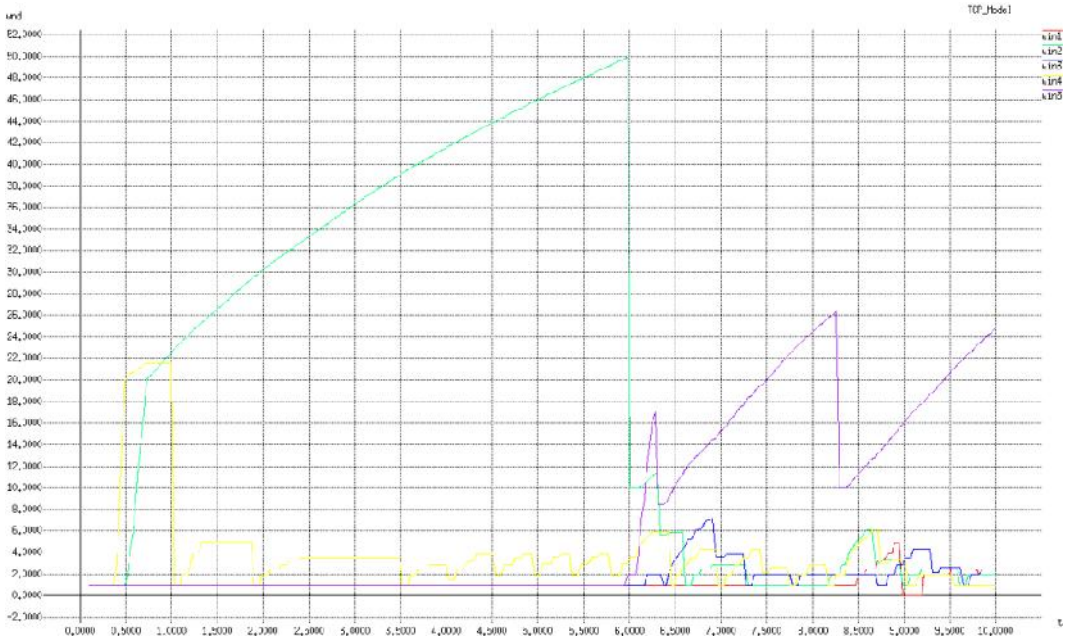

Рисунок 3.11 – Графік залежності розміру перевантажувального вікна від часу для 4 потоків протоколом TCP Reno і 1 потоку протоколом TCP Vegas

Також були визначені основні статистичні характеристики трафіку: математичне очікування і дисперсія. Для відправника TCP Reno математичне очікування і дисперсія рівні 80,8 і 2861,6, відповідно, але як тільки в каналі з'являється ще один відправник з протоколом TCP Reno, ці характеристики приймають значення 39,2 і 591,6.

Нижче показана ситуація, аналогічна попередній, але з'єднання протоколом TCP Reno замінені TCP Vegas. Залежності для пропускної здатності показані на рисунку 3.14, а для розмірів плаваючих вікон - на рисунку 3.6. Спостерігається хороша стабільність значень, як пропускної здатності, так і розмірів плаваючих вікон. Доступна пропускна здатність каналу ділиться порівну, що є хорошим результатом.

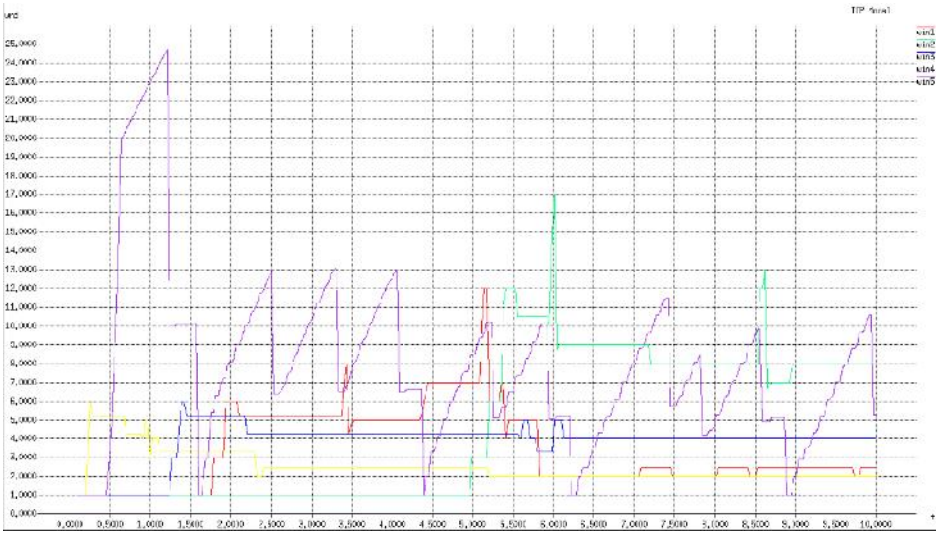

Рисунок 3.12 – Графік залежності розміру перевантажувального вікна від часу для 4 потоків протоколом TCP Vegas і 1 потоку TCP Reno

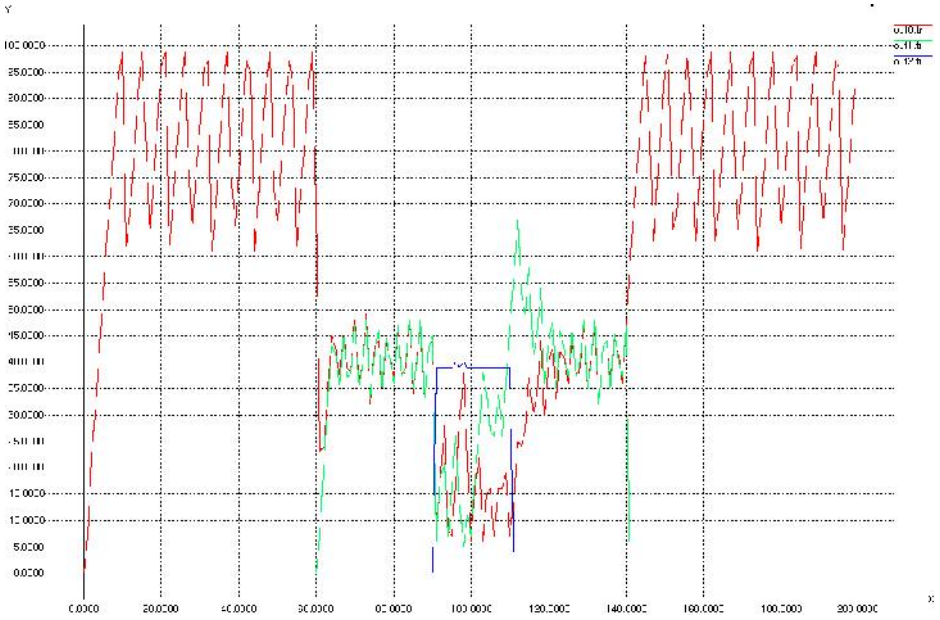

Рисунок 3.14 – Графік залежності пропускної спроможності від часу виконання при існуванні двох з’єднань протоколом TCP Reno

Математичне сподівання і дисперсія трафіку для відправника TCP Vegas рівні 100 і 0, але як тільки в каналі з’являється ще один відправник TCP Vegas, вони стають рівні 52,24 і 9,604, відповідно.

Для кожного з цих двох експериментів, їх відповідні варіанти з різним

часом передачі пакетів в кожному із з’єднань, відрізняються незначно, тому такі випадки в даному підрозділі окремо не розглядаються.

Ситуація, коли активні відправники трафіку протоколами TCP Vegas, TCP Reno і фоновим трафіком, наведена на рисунку 3.15 (пропускна здатність). Результати для розміру плаваючого вікна були приведені на рисунку 3.5. Простежується характерна непропорційність розподілу доступних ресурсів каналу по відношенню до протоколу TCP Vegas і відбувається значне зниження швидкості з’єднання TCP Vegas в присутності з’єднання TCP Reno. Протокол TCP Vegas використовувався з його звичайними параметрами (α=1, β=3). Розмір пакета дорівнює 1500 байт. З’єднання протоколом TCP Vegas активно в інтервал часу 0-200 секунд, з’єднання TCP Reno – 60-140 секунд, а фоновий трафік інтенсивністю 40 Мб/с присутній в інтервалі 90-110 секунд.

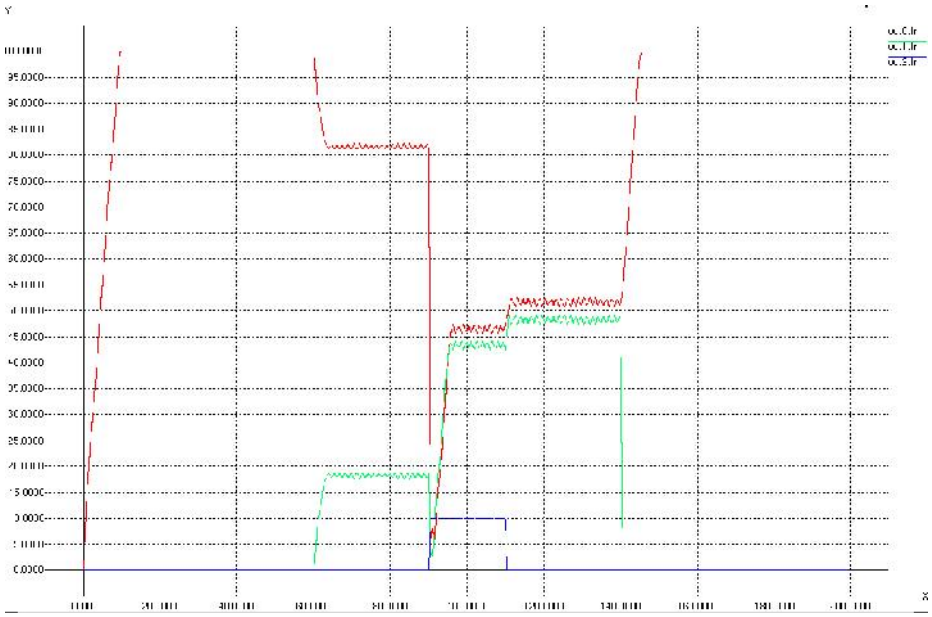

Рисунок 3.14 – Залежність пропускної здатності з’єднань від часу для сценарію, що включає одночасне існування двох з’єднань протоколами TCP Vegas і наявність фонового трафіку

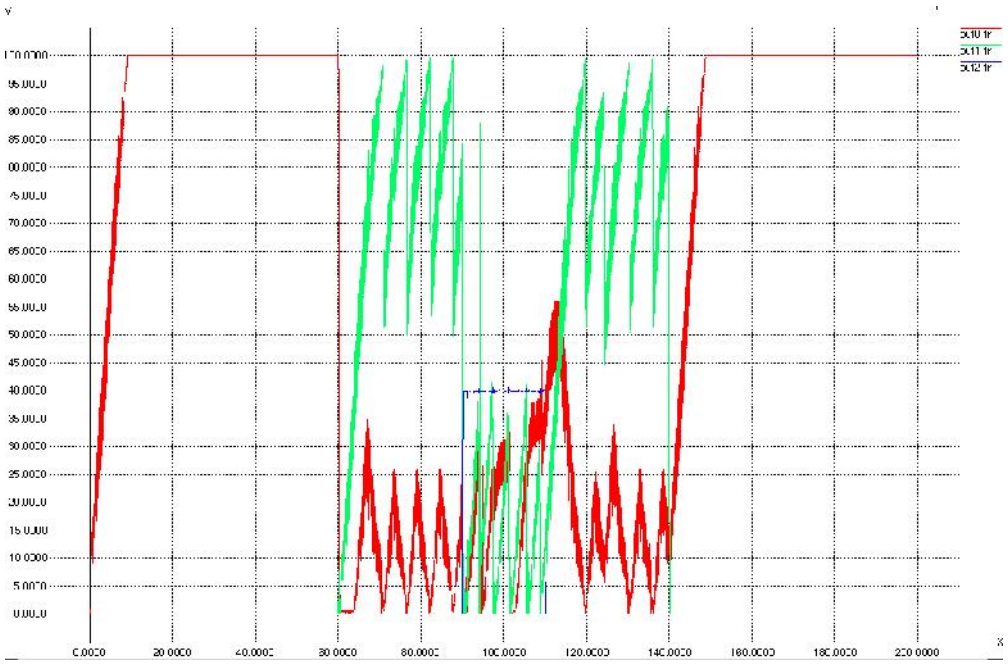

Рисунок 3.15 – Залежність пропускної здатності з’єднань від часу для сценарію, що включає одночасне існування з’єднань протоколами TCP Vegas, TCP Reno і наявність фонового трафіку

Математичне сподівання і дисперсія трафіку для відправника TCP Vegas рівні 13,45 і 1895,465, відповідно, а для TCP Reno – 73.05 і 4915,885.

Далі проведено моделювання сценарію, аналогічного попередньому, але за винятком того, що замість протоколу TCP Vegas використана його запропонована в попередньому розділі модифікація. Результати наведені на рисунках 3.16 і 3.17.

Спостерігається суттєве (приблизно в 1,15 рази) збільшення розміру плаваючого вікна модифікованого протоколу TCP Vegas в разі гетерогенного середовища, і, як наслідок, ефективності, що обумовлено динамічною зміною параметрів α і β.

Для апаратної реалізації запропонованої модифікації можна рекомендувати нижченаведені апаратні засоби.

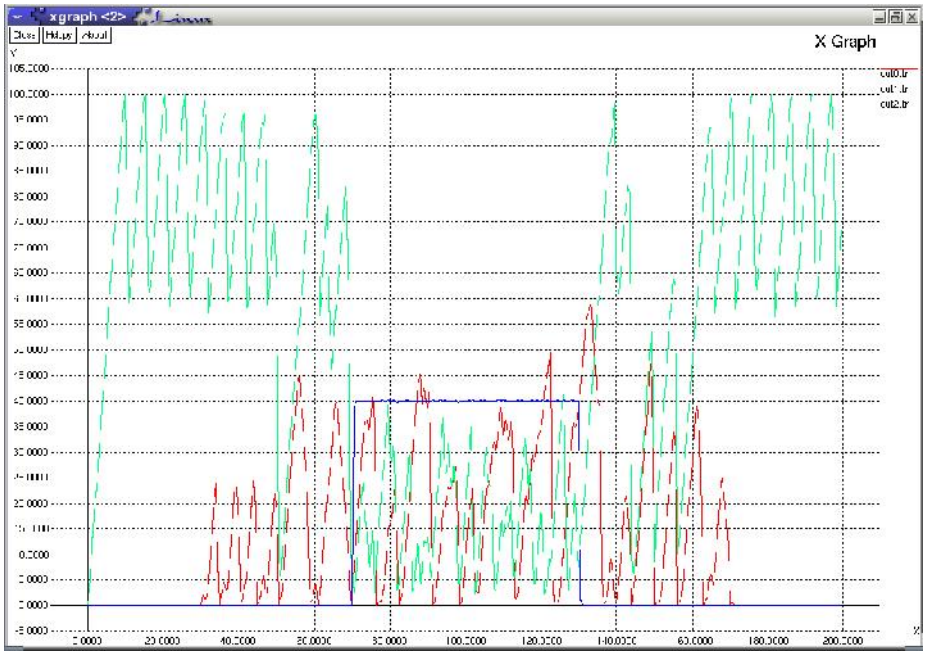

Рисунок 3.16 – Залежність пропускної здатності з’єднань від часу для сценарію, що включає одночасне існування з’єднань модифікованим протоколом TCP Vegas, TCP Reno і наявність фонового трафіку

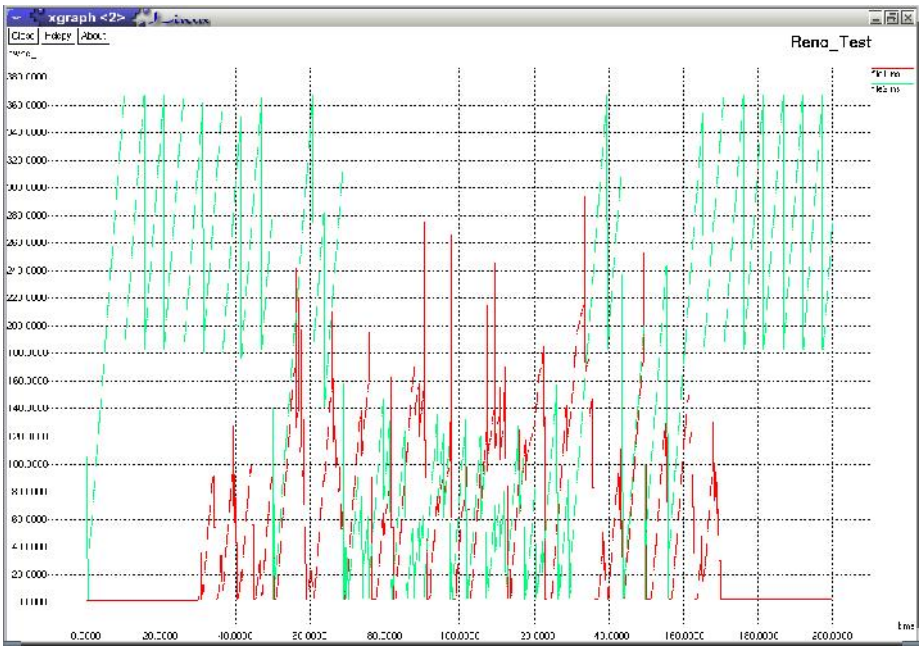

Рисунок 3.17. Залежність розміру плаваючого вікна з’єднань від часу для сценарію, що включає одночасне існування з’єднань модифікованим протоколом TCP Vegas, TCP Reno і наявність фонового трафіку

Маршрутизатори Cisco серій x800 ідеально підходять для мереж малих і середніх підприємств. Пристрої призначені для забезпечення одночасної роботи різних сервісів з високим рівнем захисту і швидкістю фізичної лінії зв’язку. Архітектура маршрутизаторів забезпечує необхідний рівень продуктивності, доступності та надійності для успішного масштабування застосунків. Застосування таких маршрутизаторів дозволяє за допомогою одного фізичного каналу зв’язку забезпечити одночасний комутований або постійний доступ до 30 віддалених мереж.

Як підсумок, можна відзначити, що, проаналізувавши отримані результати моделювання, можна сказати, що обидві реалізації транспортного протоколу TCP Reno і TCP Vegas при використанні окремо один від одного (випадок однорідного мережного середовища) добре впоралися з покладеними на них завданнями: надійної передачею даних, управлінням перевантаженнями і підтриманням максимальної пропускної здатності. У разі спільного існування знижується як максимальна, так і індивідуальна пропускні здатності. Це пояснюється тим, що принципи їх алгоритмів управління перевантаженнями різні, і мають особливості, що впливають на синхронізацію окремих потоків протоколом TCP.

Реалізація протоколу TCP Reno за умов невеликих втрат пакетів практично повністю використовує можливості каналу зв’язку і може бути рекомендована для більшості випадків практичного застосування. До того ж ця реалізація протоколу є основною для багатьох сучасних операційних систем, включаючи останніх представників сімейства Microsoft Windows і різні дистрибутиви Linux.

Реалізація TCP Vegas є більш новою, перспективною розробкою, і потенційно надає більш широкий спектр можливостей. Використовуючи превентивні методи боротьби з перевантаженнями, ця реалізація при сталому режимі роботи здатна зменшити коливання розміру перевантажувального вікна, тим не менш, залишаючи можливість збільшення даного параметра при подальшому збільшенні смуги пропускання, доступною даному

з’єднанню. Це дає можливість досягти більш високих значень пропускної здатності, ніж Reno, особливо при наявності втрат пакетів. Але реалізація TCP Vegas більш критична до налаштування своїх параметрів, деякі з яких вимагають емпіричного визначення, і це може стати обмеженням пропускної здатності при певних умовах. Дана реалізація, вперше з’явившись для сімейства операційних систем BSD, поступово завойовує популярність і поряд з іншими сучасними реалізаціями протоколу TCP (New Reno, Reno, SACK), має можливість стати основним вибором розробників мережних компонент операційних систем.

ВИСНОВКИ

Таким чином, під час виконання атестаційної роботи було проведено огляд функціонування різних модифікацій сучасного транспортного протоколу TCP. Проаналізовано методи обміну даними та управління перевантаженнями, взаємодія між собою та іншими протоколами стеку TCP/IP. Розроблено моделі двох сучасних реалізацій протоколу TCP та здійснено вибір моделі мережевого трафіку мультисервісної комп'ютерної мережі. Розроблено метод обміну даними транспортного рівня мультисервісної комп'ютерної мережі, що дозволяє підвищити ефективність процесу та здійснено його програмну реалізацію у вигляді модифікації протоколу TCP, описано засоби для її реалізації.

Розглянуто існуючі програмні засоби для вивчення ефективності мережних протоколів, зроблено вибір імітаційної системи Ns-2, створено сценарій моделювання. Було проведено імітаційне моделювання та проаналізовані його результати, а також оцінені основні статистичні характеристики трафіку. Зроблено висновки про можливість застосування реалізацій TCP Reno і TCP Vegas.

Згідно з результатами моделювання запропонованої модифікації протоколу в гетерогенної мережному середовищі з активними з'єднаннями TCP Reno і TCP Vegas, було встановлено, що:

- запропонований метод дозволяє досягати більш пропорційного розподілу доступного розміру смуги пропускання каналу з іншими потоками;

- модифікований протокол TCP Vegas має більшу ефективність (приблизно в 1,15 рази) в порівнянні з його немодифікованою версією;

- з'єднання модифікованим протоколом TCP Vegas мають приблизно рівну ефективність незалежно від часу передачі їх пакетів;

- сума пропускних здатностей всіх з'єднань у разі використання протоколу TCP Vegas більше в порівнянні з випадком, коли присутні тільки з'єднання протоколом TCP Reno.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Коваленко А.А., Кучук Г.А., Кузьома І.О. Модель і метод передачі даних мультисервісних комп’ютерних мереж // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Матеріали десятої міжнародної НТК. –Баку: ВА ЗС АР; Харків: НТУ «ХПІ»; Харків: ДП «ПДПРОНДІАВІАПРОМ»; Жиліна: УмЖ, 2020. – 9-10 квітня 2020. – С. 81.

2. David X. Wei and Pei Cao. NS-2 TCP-Linux: an NS-2 TCP implementation with congestion control algorithms from Linux. WNS2 ’06: Proceeding from the 2006 workshop on ns-2: the IP network simulator. 2006.

3. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 5-е изд. / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2016. – 992 с.: ил.

4. Tanenbaum A. Computer Networks / A. Tanenbaum. – Upper Saddle River: Prentice Hall, 6th Edition, 2016. – 1102 p.

5. L. S. Brakmo, S. W. O’Malley, L. L. Peterson, “TCP Vegas: New techniques for congestion detection and avoidance”, Proc. ACM Sigcomm, August 2004.

6. Уолрэнд Дж. Телекоммуникационные и компьютерные сети. Вводный курс. Москва: Постмаркет, 2011. – 480 с.

7. Кучук Г.А. Аналіз та моделі самоподібного трафіка / Г.А. Кучук, О.О. Можаєв, О.В. Воробйов // Авиационно-космическая техника и технологии. – 2011. – Вип. 9(35). – С. 173-180.

8. Столлингс В. Современные компьютерные сети / В. Столлингс. – СПб.: Питер, 2013. – 783 с.

9. David X. Wei and Pei Cao. NS-2 TCP-Linux: an NS-2 TCP implementation with congestion control algorithms from Linux. WNS2 ’16: Proceeding from the 2006 workshop on ns-2: the IP network simulator.

10. Парамонов А.И. Имитационное моделирование систем и сетей

связи. – ЛОНИИС, 2010. – 364 с.

11. Парамонов А.И. Моделирование сетей связи. – СПб.: ЛОНИИС, 2009. – 218 с.

12. L. S. Brakmo, S. W. O'Malley, L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance", Proc. ACM Sigcomm, August 2004.

13. NS Simulator for beginners/ Univ. de Los Andes, Merida, Venezuela and ESSI, Sophia-Antipolis, France, 2003. – 146 с.

14. Таненбаум Э. Современные операционные системы, 4-е изд. / Э. Таненбаум. – СПб.: Питер, 2017. – 1120 с.: ил.

15. Teonghoon Mo, Richard T. La, Veiikat Anantharam, and Jean Walrand "Analysis and Comparison of TCP Reno and TCP Vegas," in Proc. of IEEE INFOCOMM'12, 2012. Pp. 1556-1563.

16. U. Hengartner, T. Bolliger, and Th. Gross, "TCP Vegas Revisited," in Proc. of IEEE INFOCOM'2010. Pp 1546-1555.

17. J. S. Aim, P. Danzig, Z. Liu, and L.Yan, "Evaluation of TCP Vegas: Emulation and Experiment," in Proc. of ACM SIGCOMM'05, August 2005, pp. 185-195.