Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Метод збільшення пропускної здатності
для програмних засобів протоколу TCP NewReno

(тема)

Виконав:
студент II курсу, групи СПм-18-3
Помінчук А.В.
(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. Коваленко А.А.
(посада, прізвище, ініціали)

Допускається до захисту

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

2020 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

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 Помінчук Анні Вадимівні
(прізвище, ім’я, по батькові)

1. Тема роботи Метод збільшення пропускної здатності для програмних засобів протоколу TCP NewReno

затверджена наказом по університету від “ 30 ” березня 2020 р. № 478Ст

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

3. Вхідні дані до роботи
Стек протоколів TCP/IP
Реалізації протоколів TCP: Tahoe, Reno, NewReno, Westwood
Топологія комп’ютерної мережі

4. Перелік питань, що потрібно опрацювати в роботі
1. Аналіз предметної області і постановка завдання
2. Розробка методу збільшення пропускної здатності, обґрунтування та вибір засобів моделювання
3. Моделювання запропонованих результатів

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Демонстраційні матеріали. Плакати - 15 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	Огляд структури стеку протоколів TCP/IP	30.03.20-08.04.20	
2	Розробка методу збільшення пропускної	09.04.20-04.05.20	
	здатності, вибір засобів моделювання		
3	Моделювання запропонованих результатів	05.05.20-11.05.20	
4	Оформлення пояснювальної записки	28.04.20-11.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 87 с., 25 рис., 1 табл., 1 дод., 16 джерел.

МЕРЕЖНИЙ ПРОТОКОЛ, АЛГОРИТМ, ПЕРЕВАНТАЖЕННЯ, РЕАЛІЗАЦІЯ, ФОНОВИЙ ТРАФІК, ЕФЕКТИВНІСТЬ, З'ЄДНАННЯ, ПРОПУСКНА ЗДАТНІСТЬ, МОДЕЛЮВАННЯ, ТОПОЛОГІЯ, АГЕНТ, КАНАЛ.

Метою атестаційної роботи є розробка методу збільшення пропускної здатності транспортного протоколу та її відповідної реалізації, а також її імітаційне моделювання при різноманітних сценаріях.

У ході виконання атестаційної роботи проведено огляд теоретичних аспектів функціонування транспортного протоколу TCP у його різних реалізаціях, розроблено метод збільшення пропускної здатності протоколу TCP, виконано аналіз існуючих інструментальних засобів щодо оцінки ефективності мережних протоколів, моделювання роботи реалізацій протоколу за певних умов. Розглянуто результати моделювання, дана оцінка пристосовності деяких реалізацій протоколу TCP до поставлених завдань.

ABSTRACT

Master’s thesis: 87 pages, 25 figures, 1 table, 1 appendix, 16 sources.

NETWORK PROTOCOL, ALGORITHM, CONGESTION, IMPLEMENTATION, BACKGROUND TRAFFIC, EFFICIENCY, CONNECTION, BANDWIDTH, SIMULATION, TOPOLOGY, AGENT.

The major goal of this thesis is to develop a method for increasing the throughput of transport protocol and its implementation, as well as its simulation in various scenarios.

During preparation of the thesis, theoretical aspects of TCP transport protocol operation and its various implementations were carried out, the method of increasing the throughput of the TCP protocol was developed, the analysis of existing tools for evaluating the efficiency of network protocols, modeling of the implementation of the protocol under certain conditions were performed. The simulation results are discussed, the estimation of the adaptability of some TCP implementations to the tasks is given.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ 8
ВСТУП 9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАННЯ 11
1.1 Аналіз структури стеку протоколів TCP/IP 11
1.2 Аналіз сучасних програмних засобів протоколу TCP 15
1.3 Постановка завдань атестаційної роботи 23
2 РОЗРОБКА МЕТОДУ ЗБІЛЬШЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ, ОБГРУНТУВАННЯ ТА ВИБІР ЗАСОБІВ МОДЕЛЮВАННЯ 25
2.1 Моделі сучасних реалізацій протоколу TCP 25
2.2 Метод збільшення пропускної здатності для протоколу TCP NewReno 31
2.3 Обгрунтування та вибір засобів моделювання 33
2.4 Історія розвитку Ns2 34
2.5 Архітектура Ns2 35
2.6 Мова «Object Tcl» 37
2.7 Етапи моделювання 39
2.8 Управління моделюванням 53
3 МОДЕЛЮВАННЯ ЗАПРОПОНОВАНИХ РЕЗУЛЬТАТІВ 55
3.1 Опис поставлених завдань та сценарію 55
3.2 Розробка топології 56
3.3 Створення сценаріїв моделювання 57
3.4 Результати моделювання 61
3.4.1 Моделі з одиничними з’єднаннями 61
3.4.2 Моделі з декількома з’єднаннями 65
3.4.3 Моделі зі з’єднанням TCP і одним UDP потоком 69

3.5 Застосування реалізацій протоколу TCP при наявності фонового трафіку 74
ВИСНОВКИ 76
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ 77
ДОДАТОК А Графічний матеріал атестаційної роботи 79

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACK – підтвердження отримання даних (англ., acknowledgment)

CBR – постійна бітова швидкість (англ., constant bit rate)

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

OSI – взаємодія відкритих систем (англ., open system interconnection)

RED – раннє випадкове виявлення (англ., random early detection)

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

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

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

ВСТУП

Подальше збільшення ролі інформаційних технологій робить їх невід'ємною частиною практично будь-якої сфери життя і діяльності людини. Почавши свою історію з примітивних електронно-обчислювальних машин, комп'ютери продовжують удосконалюватися, і надають все більше рішень для роботи з інформацією. Сьогодні окремий комп'ютер має дуже обмежену функціональність і не забезпечує необхідної гнучкості і масштабованості. Виникнення комп'ютерних мереж і подальша вимога можливості обміну даними між розосередженими системами відкрили нові горизонти для побудови розподілених ресурсів, їхнього адміністрування і наповнення, починаючи від розподіленого збереження інформації (мережні файлові системи, файлові архіви, інформаційні системи з віддаленим доступом), і закінчуючи мережним обчислювальним середовищем.

Робота глобальних і локальних комп'ютерних мереж забезпечується так званими протоколами – певними наборами правил, що визначають взаємодію функціональних блоків на різних рівнях. Протоколи об'єднуються в стеки, які в своєму складі містять ряд протоколів, що реалізують мінімальну сукупність необхідних функцій з організації комп'ютерної мережі. В даний час домінуючим стеком мережних протоколів є стек TCP/IP, відповідний моделі організації взаємодії відкритих систем (OSI). Всі протоколи, що входять в стек TCP/IP, є стандартизованими міжнародною організацією зі стандартизації (ISO), і їх опис визначено в численних документах серії RFC.

В даній атестаційній роботі розглядаються питання, пов'язані з роботою сучасної реалізації протоколу управління передачею даних (TCP), що відповідає транспортному рівню моделі OSI.

Перші реалізації протоколів, що складають сьогодні стек TCP/IP, пройшли довгий шлях удосконалень для забезпечення вимог одного з найбільш значущих феноменів століття – глобальної мережі Internet.

Протоколи стека TCP/IP використовуються практично в будь-якому комунікаційному середовищі, від локальних мереж на базі технології Ethernet, до надшвидкісних мереж ATM, від телефонних каналів типу «точка-точка» до трансатлантичних ліній зв’язку з пропускною здатністю в кілька сотен гігабіт на секунду.

Визначальним критерієм будь-якого протоколу є його ефективність, що включає в себе такі параметри як пропускна здатність, надійність, методи боротьби з перевантаженнями, складність їх застосування та інші. В даний час розроблено багато засобів, що дозволяють оцінити ефективність різних протоколів в різних умовах. Дані засоби використовують різні методи збору, обробки і виводу інформації про різні параметри та умови роботи протоколу і дозволяють в зручній формі провести їх статистичний аналіз.

Метою даної атестаційної роботи є дослідження пропускної здатності роботи сучасного протоколу TCP при наявності в мережі фонового трафіку, а також розробка модифікації методу та відповідної реалізації протоколу TCP, що дозволяє підвищити його пропускну здатність. Актуальність даної проблеми обумовлюється тим, що в даний час в більшості комп’ютерних мереж одночасно існує множина видів трафіку (Internet, бази даних, файлові сховища, електронна пошта та інші). Аналіз впливу фонового трафіку і розробка нової модифікації протоколу TCP дозволить підвищити ефективність задач, що розв’язуються програмними засобами в сучасних комп’ютерних мережах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ

У даному розділі розглядаються основні питання, пов’язані функціонуванням протоколу TCP: його роль в загальній структурі стека протоколів TCP/IP, найбільш поширені реалізації, проблеми, а також шляхи їх усунення.

1.1 Аналіз структури стеку протоколів TCP/IP

У 1969 році Агентство досліджень (DAPRA) Міністерства Оборони США почало фінансування проекту зі створення експериментальної мережі на основі технології комутації пакетів. Ця мережа, що отримала назву «APRANET», була побудована для забезпечення надійного зв’язку між комп’ютерним устаткуванням різних виробників. Під час розвитку мережі були розроблені ряд комунікаційних протоколів для неї. Так з’явилося сімейство протоколів TCP/IP.

У 1983 році стек TCP/IP був стандартизований (у відповідності до MIL STD), в той же час агентство DAPRA почало фінансування проекту Каліфорнійського університету в Берклі з підтримки стека TCP/IP в операційній системі UNIX. Популярність UNIX-подібних систем призвела до широкого поширення протоколів стека TCP/IP. Сьогодні цей стек є також основним для сімейства операційних систем Windows, MacOS, FreeBSD, на ньому працює всесвітня мережа Internet, чий підрозділ «Internet Engineering Task Force» (IETF) вносить основний вклад в удосконалення стандартів стека, що публікуються в формі специфікацій сімейства RFC.

Структура стека протоколів TCP/IP приведена на рисунку 1.1. Стек протоколів TCP/IP поділяється на 4 рівні. Оскільки стек був розроблений до появи моделі взаємодії відкритих систем ISO/OSI, то, незважаючи на багаторівневу структуру, відповідність рівнів стека TCP/IP рівням моделі OSI

є досить умовною.

Найнижчий рівень (IV) відповідає фізичному і канальному рівням моделі OSI. Цей рівень в протоколах TCP/IP не регламентується, але підтримує всі популярні стандарти фізичного і канального рівня: для локальних мереж це Ethernet, Token Ring, FDDI, Fast Ethernet, 100VG-AnyLAN, для глобальних мереж – протоколи з'єднань типу «точка-точка» SLIP і PPP, протоколи територіальних мереж з комутацією пакетів X.25, Frame Relay. Розроблена також специфікація, яка визначає використання технології ATM в якості транспорту канального рівня. Зазвичай при появі нової технології локальних або глобальних мереж вона швидко включається в стек TCP/IP за рахунок розробки відповідних технічних RFC, що визначає метод інкапсуляції пакетів IP у її кадри.

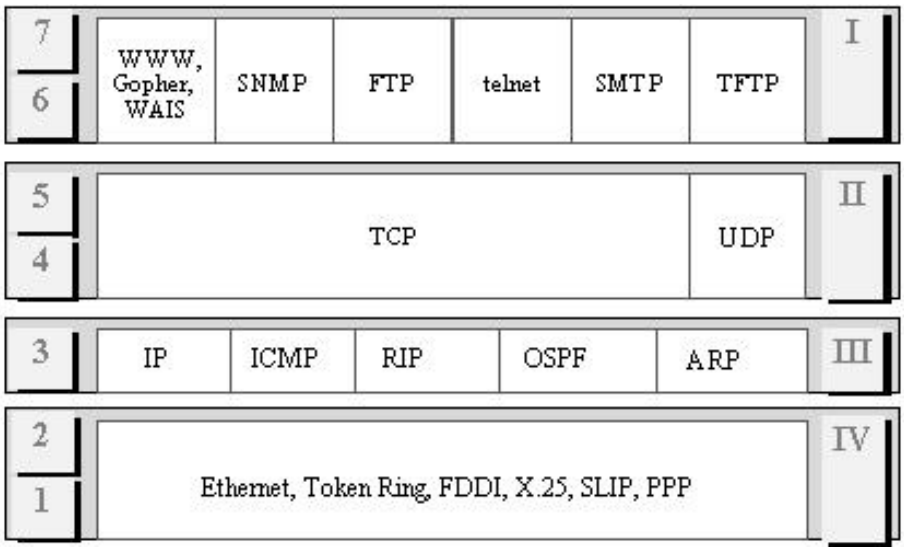

Рисунок 1.1 – Структура стеку протоколів TCP/IP

Частково популярність стека протоколів TCP/IP пояснюється можливістю його реалізації на базі великого числа каналів різних типів, а також протоколів канального рівня.

Наступний рівень (III) – це рівень міжмережної взаємодії, який займається передачею пакетів з використанням різних транспортних

технологій локальних мереж, територіальних мереж, ліній спеціального зв’язку та інше.

В якості основного протоколу мережного рівня (в термінах моделі OSI) в стеку використовується протокол IP, який спочатку проектувався як протокол передачі пакетів в складових мережах на основі великої кількості локальних мереж, об’єднаних як локальними, так і глобальними лініями зв’язку. Тому протокол IP досить добре функціонує в мережах зі складною топологією, раціонально використовуючи наявність в них підсистем і ощадливо витрачаючи пропускну здатність низькошвидкісних ліній зв’язку. Протокол IP є дейтаграмним протоколом, тобто він не гарантує доставку пакетів до вузла призначення, але намагається це зробити.

До рівня міжмережної взаємодії належать і всі протоколи, які пов’язані зі складанням і модифікацією таблиць маршрутизації, такі як протоколи збору маршрутної інформації «Routing Internet Protocol» (RIP) і «Open Shortest Path First» (OSPF), а також протокол міжмережних керуючих повідомлень «Internet Control Message Protocol» (ICMP). Останній призначений для обміну інформацією про помилки між маршрутизаторами мережі і вузлом-джерелом пакетів. За допомогою спеціальних пакетів, ICMP повідомляє про неможливість доставки пакета, про перевищення часу життя або тривалості зборки пакета з фрагментів, про аномальні величини параметрів, про зміну маршруту пересилання і типу обслуговування, про стан системи та інше.

Рівень II називається основним. На цьому рівні функціонують протокол управління передачею TCP і протокол дейтаграм користувача «User Datagram Protocol» (UDP). Протокол TCP забезпечує надійну передачу повідомлень між віддаленими прикладними процесами за рахунок утворення віртуальних з’єднань. Протокол UDP забезпечує передачу прикладних пакетів дейтаграмним способом, як і IP, і виконує тільки функції сполучної ланки між мережним протоколом і численними прикладними процесами.

Верхній рівень (рівень I) називається прикладним. За довгі роки

використання в мережах різних країн і організацій, стек TCP/IP нагромадив велику кількість протоколів і сервісів прикладного рівня. До них належать такі широко використовувані протоколи, як протокол передачі файлів FTP, протокол емуляції терміналу Telnet, поштовий протокол SMTP, використовуваний в електронній пошті мережі Internet, гіпертекстові сервіси доступу до вилученої інформації, такі як WWW і багато інших.

Оскільки стек TCP/IP розроблявся для глобальної мережі Internet, він має ряд особливостей, що дають йому перевагу перед іншими протоколами при побудові мереж, що включають глобальні зв'язки. Зокрема, дуже корисною властивістю, що робить можливим застосування цього протоколу у великих мережах, є його здатність фрагментувати пакети. Дійсно, велика мережа часто складається з мереж, побудованих на абсолютно різних принципах. У кожній з цих мереж може бути встановлена власна величина максимальної довжини одиниці переданих даних (кадру). У такому випадку при переході з однієї мережі, що має велику максимальну довжину, в мережу з меншою максимальною довжиною може виникнути необхідність розподілу кадру на частини. Протокол IP ефективно вирішує таку задачу.

Іншою особливістю стека протоколів TCP/IP є гнучка система адресації, що дозволяє більш просто, в порівнянні з іншими протоколами аналогічного призначення, об'єднувати мережі інших технологій. Це властивість також сприяє застосуванню стека TCP/IP для побудови великих гетерогенних мереж.

Проте платою за одержувані переваги виявляються високі вимоги до ресурсів і складність адміністрування IP-мереж. Гнучка система адресації і відмова від широкомовних розсилок приводять до наявності в IP-мережі різних централізованих служб типу DNS, DHCP та інших. Кожна з цих служб спрямована на полегшення адміністрування мережі, в тому числі і на полегшення конфігурування обладнання, але в той же час сама вимагає пильної уваги з боку адміністраторів.

1.2 Аналіз сучасних програмних засобів протоколу TCP

Завдання забезпечення надійного обміну даними між двома прикладними процесами в стеці TCP/IP вирішує протокол управління передачею TCP. Протокол TCP працює безпосередньо над протоколом IP і використовує для транспортування своїх блоків даних потенційно ненадійний протокол IP. TCP здійснює доставку дейтаграм, званих сегментами, у вигляді байтових потоків зі встановленням з'єднання. Це застосовується в тих випадках, коли потрібна гарантована доставка повідомлень. Протокол TCP використовує контрольні суми пакетів для перевірки їх цілісності та звільняє прикладні процеси від необхідності таймаутів і повторних передач для забезпечення надійності.

Структура сегмента TCP. Сегмент – це одиниця даних, що передаються між пристроями, які використовують протокол TCP. Формат сегмента TCP складається з 20-60-байтних заголовків і наступних за ними даних для прикладної програми. Заголовок – це 20 байт, якщо не має опцій, і 60 байт, якщо складається з декількох опцій (рисунок 1.2). У заголовку міститься основна інформація, яка забезпечує функціонування протоколу TCP, тому дана частина пакета є критично важливою.

Адреса порту джерела – поле довжиною 16 біт, що визначає номер порту прикладної програми вузла, в який надсилається сегмент.

Адреса порту вузла призначення – поле довжиною 16 біт, що визначає номер порту прикладної програми в хості, який отримує сегмент. У встановленому з'єднанні кожна сторона використовує випадковий номер, щоб створити початковий порядковий номер («initial sequence number», ISN), який зазвичай відрізняється в кожному напрямку.

Наприклад, якщо ISN = 2368 і перший сегмент переносить 1000 байт, то порядковий номер – 2369 (2367 і 2368 використовуються для встановлення з'єднання); другий сегмент після 1000 байтного сегмента, переносить 500 байт. Він буде мати номер 3369, і так далі. Пункт призначення може

визначити номер останнього байта для визначення розміру блоку даних.

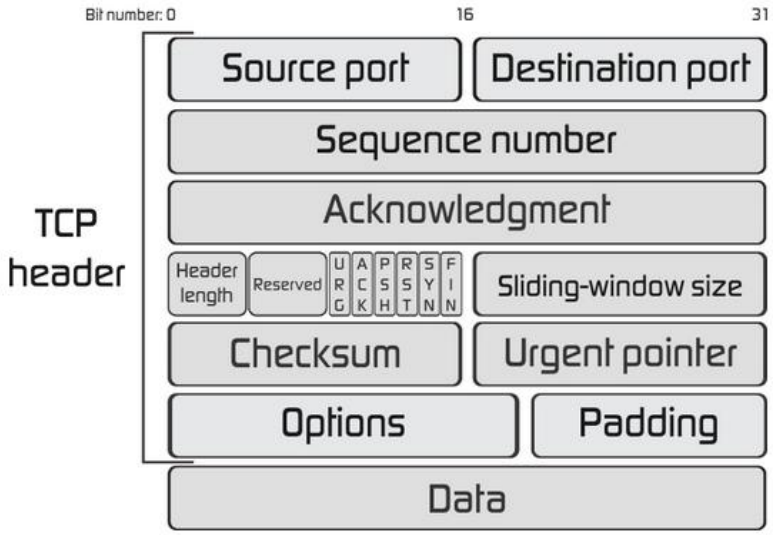

Рисунок 1.2 – Формат заголовку TCP сегмента

Порядковий номер – це 32-бітове поле визначає номер, який призначається від першого біта даних, що міститься в цьому сегменті. Щоб забезпечувати можливість взаємодії, кожен байт при передачі повинен бути пронумерований. Порядковий номер вказує вузлу призначення номер байта в послідовності в порівнянні з першим байтом сегмента.

Номер підтвердження займає 32 біта і визначає номер байта, який очікує адресат від іншої сторони. Пакети підтвердження ACK мають велике значення для продуктивності будь-якого з'єднання. По суті, вони є «квитанцією», по якій відправник дізнається про доставку пакета. У ній, зокрема, вказується порядковий номер наступного очікуваного пакета. Одержувач повинен підтвердити прихід кожного блоку пакетів. Якщо один з пакетів в пропущений, то в послідовності номерів утворюється «прогалина». Відправник буде потім повторювати посилку, починаючи з останнього порядкового номера безперервної послідовності пакетів для пакета, отриманого після «прогалини».

Довжина заголовка – поле довжиною 4 біта, вказує номер 4-байтного

слова в TCP заголовку. Така довжина заголовка може варіюватися від 20 до 60 байт. Тому значення цього поля може бути між 5 (5 * 4 = 20) та 15 (15 * 4 = 60).

Зарезервовано – поле 6 біт, зарезервовано для майбутнього використання.

Управління – поле визначає 6 різних керуючих біт, або флагів, здатних управляти потоком, встановленням з'єднання і кінцевими пристроями, а також режимом передачі TCP. Короткий опис кожного з бітів дано в таблиці 1.1.

Розмір вікна – це поле визначає розмір вікна в байтах, який може використовувати інша сторона. Довжина цього поля дорівнює 16 бітам, що відповідає максимальному розміру вікна в 65535 байт.

Таблиця 1.1 – Значення розрядів поля «флаги»

Позначення бітів (зліва направо) поля «флаги»	Значення біта, якщо він дорівнює 1
URG (терміново)	Флаг важливої інформації, поле «Вказівник важливої інформації» має значення, якщо urg=1
ACK (підтвердження)	Номер октету, який повинен прийти наступним
PSH (запуск)	Цей сегмент вимагає виконання операції «push». Одержувач повинен передати ці дані прикладній програмі якомога швидше
RST (переривання)	Переривання зв'язку
SYN (синхронізація)	Флаг для синхронізації номерів сегментів, використовуваних при встановленні зв'язку
FIN (кінцеве)	Відправник закінчив посилку байтів

Контрольна сума – поле розміром 16 біт, яке дійсно, тільки якщо встановлений строковий флаг, і використовується, коли сегмент містить термінові дані. Воно визначає, що повинно бути доповнено до порядкового

номера, щоб отримати номер останнього і термінового байта в секції даних сегмента.

Поле «тема» може містити до 40 байт інформації опцій. Опції передають додаткову інформацію для адресата або вирівнюють інші опції. Визначено дві категорії опцій: однобайтове і багатобайтове. Перша категорія має два типи опцій: закінчена опція або немає операції. Друга категорія містить три типи опцій: розмір максимального сегмента, коефіцієнт масштабу, а також мітку часу.

Кінець опції – це опція в 1 байт використовується для заповнення поля «опції», щоб вирівняти формат. Вона може використовуватися тільки як остання опція і тільки з одного кінця. Після цієї опції адресат шукає корисне навантаження. Іноді поле опції може містити більш ніж один байт. Це означає, що необхідно вирівняти поле опції, використовуючи кілька опцій типу «немає операції» (або нічого не робити), після чого поставити кінець опції. Кінець опції несе три значення інформації для кінцевого пункту, вона вказує, що:

- немає більше опцій в заголовку;
- залишок від останнього 32-бітового слова можна не обробляти;
- дані від прикладної програми стартують з початку наступного 32-бітового слова.

Немає операцій – ця опція в один байт використовується як роздільник між опціями. Наприклад, вона може застосовуватися для вирівнювання наступної опції до 32-бітової границі.

Максимальний розмір сегмента (MSS) – ця опція визначає розмір найбільшої частини даних, яка може бути отримана пунктом призначення сегмента TCP. Всупереч своїй назві вона визначає максимальний розмір даних, але не максимальний розмір сегмента. У той же час, поле довжиною 16 біт може бути значенням 65535 байт. Значення за замовчуванням – 536.

Максимальний розмір даних визначається протягом фази встановлення з’єднання адресатом сегмента. Так, сторона «1» визначає MSS сторони «2».

Сторона «2» визначає, що повинно бути MSS, посланим зі сторони «1». Якщо ні та, ні інша сторона не визначає розмір, він вибирається за замовчуванням. Ця опція застосовується тільки в сегментах, які здійснюють з'єднання. Вона не може використовуватися в сегментах протягом передачі.

Коефіцієнт масштабування вікна – поле розміру вікна в заголовку визначає розмір ковзаючого вікна. Це поле має довжину 16 біт, таким чином, вікно може мати розмір від 0 до 65535. Хоча такий розмір вікна здається дуже великим, він ще може не бути ефективним, зокрема, якщо дані проходять середовище з високою пропускною здатністю з малими затримками передачі. Коефіцієнт масштабування вікна може бути визначений тільки під час фази встановлення з'єднання. Під час передачі даних розмір вікна (заданий заголовком) може змінюватися, але він може бути помножений на той же самий коефіцієнт масштабування. Формат опції коефіцієнта масштабування показаний на рисунку 1.3. Він містить код опції, загальну довжину опції і значення коефіцієнта масштабування.

Рисунок 1.3 – Опція «масштабний коефіцієнт вікна»

Мітка часу – це 10-бітова опція, заповнюється джерелом коли сегмент залишає джерело. Адресат отримує сегмент і накопичує значення мітки часу. Коли адресат посилає підтвердження байтів в цьому сегменті, він вводить накопичене попереднє значення в поле «відлуння відповіді». Джерело, коли отримує підтвердження, перевіряє поточний час в порівнянні з цим значенням. Різниця – це час проходження «туди і назад». Значення часу «туди і назад» може бути використано протоколом TCP динамічно для визначення часу повторної передачі.

Основні проблеми роботи протоколу TCP і методи їх вирішення.

Транспортний протокол TCP був розроблений, коли швидкості передачі даних і пропускні здатності комп’ютерних мереж були набагато нижче сучасних. У разі застосування його в високошвидкісних мережах, проявилися такі властиві протоколу недоліки, як неефективне використання смуги пропускання і фрактальна природа [1-3].

При передачі виникають ситуації, коли необхідно управляти швидкістю надходження пакетів для того, щоб передавальний вузол не переповнив приймальний надмірною кількістю даних. Цей контроль здійснюється за допомогою управління потоком даних [4].

Щоб не допустити переповнення буферів приймача, необхідно регулювати потік даних від передавача. Найпростіший механізм управління відомий під назвою управління потоком з зупинками. Він працює таким чином: вузол-відправник передає пакет, після, його адресат посилає підтвердження, що служить сигналом готовності прийняти наступний, тобто перш ніж джерело передасть черговий пакет, воно повинне отримати підтвердження прийому попереднього. Отже, адресат може зупинити потік даних, утримавшись від відправки підтвердження прийому.

Основний недолік цього методу полягає в низькій ефективності використання каналу при великих швидкостях передачі. Він викликаний тим, що в кожен момент часу в дорозі може перебувати тільки один пакет, і якщо довжина каналу зв’язку істотно перевищує довжину пакета, то канал виявляється незавантаженим.

Другою ключовою проблемою є перевантаження в мережах передачі даних. Перевантаження виникає в разі, коли кількість пакетів, що передаються по мережі, починає наближатися до максимально можливого значення, яке в змозі обробити мережа.

З ростом навантаження на мережу збільшується довжина черг в різних вузлах. Загалом, досягається точка максимальної продуктивності мережі, після цього її пропускна здатність різко знижується при збільшенні навантаження, що надходить. Це викликається тим, що розміри буферів на

вузлах обмежені. При їх заповненні вузли не в змозі приймати пакети. Внаслідок цього відправники змушені, крім нових пакетів, знову передавати відхилені. Навіть пакети, доставлені успішно, можуть передаватися знову, оскільки час їх підтвердження перевищує допустимий час очікування відправника, визначений в таймері повторної передачі («Retransmission Timeout», RTO), і той посилає їх знову. Процес лавиноподібно наростає, і в підсумку пропускна здатність мережі знижується до нуля.

Окремою проблемою необхідно виділити фрактальну природу мережного трафіку [5]. Під нею передбачається повторюваність розподілу навантаження в часі при його різних масштабах. Це означає, що якщо зобразити графік залежності щільності інформаційного потоку від часу, взявши за одиницю виміру секунду, хвилину, годину і так далі, то кожен раз вийдуть практично однакові діаграми.

Дослідження показали, що спочатку потоки даних які не виявляють властивостей самоподібності, пройшовши обробку на вузлових серверах і активних мережевих елементах, починають подавати яскраво виражені ознаки самокореляції. Особливо сильно цей ефект помітний в високошвидкісних мережах. Перші експерименти проводилися з використанням ESnet (Energy Sciences network) – потужної мережі, що з’єднує національні лабораторії.

Мережі, побудовані на стеку TCP/IP, схильні до прояву потужних пікових сплесків передачі пакетів. Такі локалізовані в часі «товкотнечі» викликають значні втрати пакетів, навіть коли сумарна потреба всіх потоків далека від максимально допустимих значень [6].

Всі ці проблеми частково вирішує управління потоком методом ковзаючого вікна [7]. При такому методі обидва вузла, які беруть участь в обміні даними, використовують для кожного з’єднання вікно, яке встановлює обмеження на буфер доставки даних від прикладних програм і готовність до посилки. Вікно виділяє невелику частину буфера, що містить байти, які буфер може передати, перш ніж очікувати підтвердження від іншого вузла.

Вікно називається ковзаючим, тому що воно може ковзати як по буферу даних, так і по буферу підтвердження – змінюючи при цьому число байтів, які можна одночасно посилати і приймати без прийняття або посилки сигналу підтвердження. Для того щоб утримувати кількість пакетів в мережі нижче рівня, при якому різко зменшується її продуктивність, застосовують різні механізми управління перевантаженням.

Потрібно звернути увагу на основні фази управління вікном, що використовуються практично у всіх сучасних реалізаціях протоколу TCP – повільний старт, запобігання перевантаження і швидка повторна передача.

Чим більше розмір встановленого в протоколі TCP вікна передачі, тим більше сегментів в змозі відправити в мережу відправник, перш ніж включити таймер очікування підтверджень [8, 9]. Це може створити перевантаження. Тому була запропонована процедура, відома під назвою повільного старту (рисунок 1.4). Тут використовується ковзаюче вікно, *s*, розмір якого вимірюється в сегментах і управляється відправником. При відкритті нового з'єднання об'єкт TCP привласнює *s* початкове значення 1. В цей момент TCP має право передати лише один сегмент і потім повинен очікувати підтвердження. Після кожного підтвердження значення *s* збільшується на 1 до деякої максимальної величини.

З приходом підтверджень TCP збільшує розмір ковзаючого вікна, поки потік даних не стане контролюватися вже не розміром *s*, а надходженими підтвердженнями. Насправді значення *s* зростає експоненціально, тобто розмір вікна збільшується на 1 після кожного отримання відправником підтвердження.

Експоненціальне зростання ковзаючого вікна при повільному старті може виявитися занадто швидким і призвести до перевантаження. Тому було запропоновано спочатку використовувати повільний старт, а потім збільшувати розмір *s* лінійно. Для цього встановлюється граничне значення ковзаючого вікна, *S*, що дорівнює половині того, при якому сталася втрата пакета. На ділянці до порогового значення виконується процедура повільного

старту, а після цього збільшення вікна відбувається лінійно.

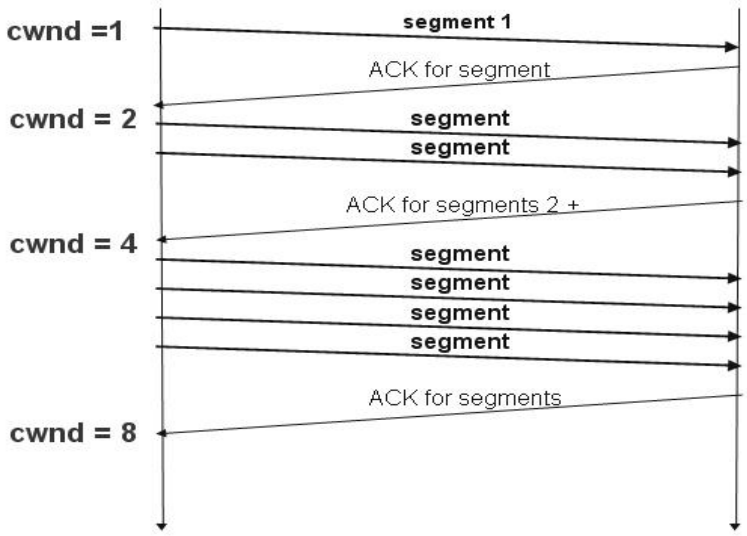

Рисунок 1.4 – Ілюстрація алгоритму повільного старту

При використанні алгоритму швидкої повторної передачі, після отримання невеликої кількості дубльованих підтверджень для одного і того ж сегмента TCP, відправник даних робить висновок, що пакет був втрачений і передає його повторно без очікування закінчення таймера повторної передачі, що призводить до кращого використання каналу та підвищення пропускної здатності з’єднання.

1.3 Постановка завдань атестаційної роботи

В даному розділі був проведений аналіз протоколів транспортного рівня стека протоколів TCP/IP, розглянуті основні проблемні області протоколу управління передачею TCP, а також способи їх усунення. Слід зазначити, що всі розглянуті вище реалізації протоколу спрямовані на усунення різного роду перевантажень і підвищення ефективності роботи мережі. Однак навіть очевидна перевага більш сучасних реалізацій над своїми попередниками в стаціонарних режимах роботи не дозволяє їм бути

універсальним засобом усунення всіх недоліків протоколу TCP у загальному випадку, основним з яких є зниження пропускної здатності при наявності в каналі фонового трафіку.

Таким чином, в даній атестаційній роботі поставлено завдання розробки моделей сучасних реалізацій протоколу TCP, методу та відповідних програмних засобів протоколу TCP, що дозволяє підвищити його пропускну здатність при наявності в мережі фонового трафіку, а також обґрунтування вибору програмно-апаратної частини, що найбільш задовольняє умовам запропонованої реалізації. За допомогою середовища моделювання комп’ютерних мереж Ns2 буде проведена оцінка різних параметрів протоколу при різних умовах його функціонування. Аналіз отриманих експериментальних даних може послужити основою для розробки рекомендацій з використання конкретних реалізацій протоколу і вибору програмного забезпечення мережі.

2 РОЗРОБКА МЕТОДУ ЗБІЛЬШЕННЯ ПРОПУСКНОЇ ЗДАТНОСТІ, ОБГРУНТУВАННЯ ТА ВИБІР ЗАСОБІВ МОДЕЛЮВАННЯ

2.1 Моделі сучасних реалізацій протоколу TCP

TCP Tahoe. Протокол TCP Tahoe є найбільш ранньою і широко поширеною версією. Він спочатку був сформульований Джейкобсоном в 1988 році, а деякі модифікації були внесені в нього пізніше [10].

Якщо буфер адресата переповнюється, то якусь кількість сегментів буде втрачено. При цьому може бути запущено декілька сценаріїв. Основний варіант – повільний старт, запускається в рамках класичного алгоритму TCP Tahoe при втраті сегмента і зв’язаним з ним таймаутом у відправника, оскільки відправник не отримає сигналу підтвердження ACK для втраченого сегмента. Повільний старт передбачає установку розміру ковзаючого вікна, s, в 1, а порога повільного старту, S, в половинну значення s, при якому стався таймаут. Зменшення значення s до одиниці відбувається тому, що відправник не має ніякої інформації про стан мережі. Далі, після отримання кожного пакету підтвердження, розмір вікна встановлюється відповідно до наступного виразу:

$$s_{i+1} = s_i + 1. \tag{2.1}$$

Вираз (2.1) справедливий до тих пір, поки розмір ковзаючого вікна не стане рівним значенню порога повільного старту. Після цього зростання ковзаючого вікна стає лінійним. Сенс цього алгоритму полягає в утриманні значення вікна в області максимально можливих значень. По суті ця оптимізація здійснюється з допомогою втрати пакетів. Якщо втрати пакетів немає, значення вікна досягає значення вікна за замовчуванням, що задається при конфігурації TCP-драйвера. На рисунку 2.1 показана залежність розміру

ковзаючого вікна від часу при втраті пакетів.

Значення таймаута може бути обчислено відповідно до наступного виразу:

$$T_O = T + 4\sigma_T , \qquad (2.2)$$

де σ – середньоквадратичне відхилення середнього значення часу передачі пакету (*T*) на встановленому маршруті.

Втрачений пакет і усі передані після нього (незалежно від того, підтверджено їх отримання чи ні) пересилаються повторно. При високій імовірності втрати це суттєво зменшує пропускну здатність і збільшує і без того високе завантаження каналу.

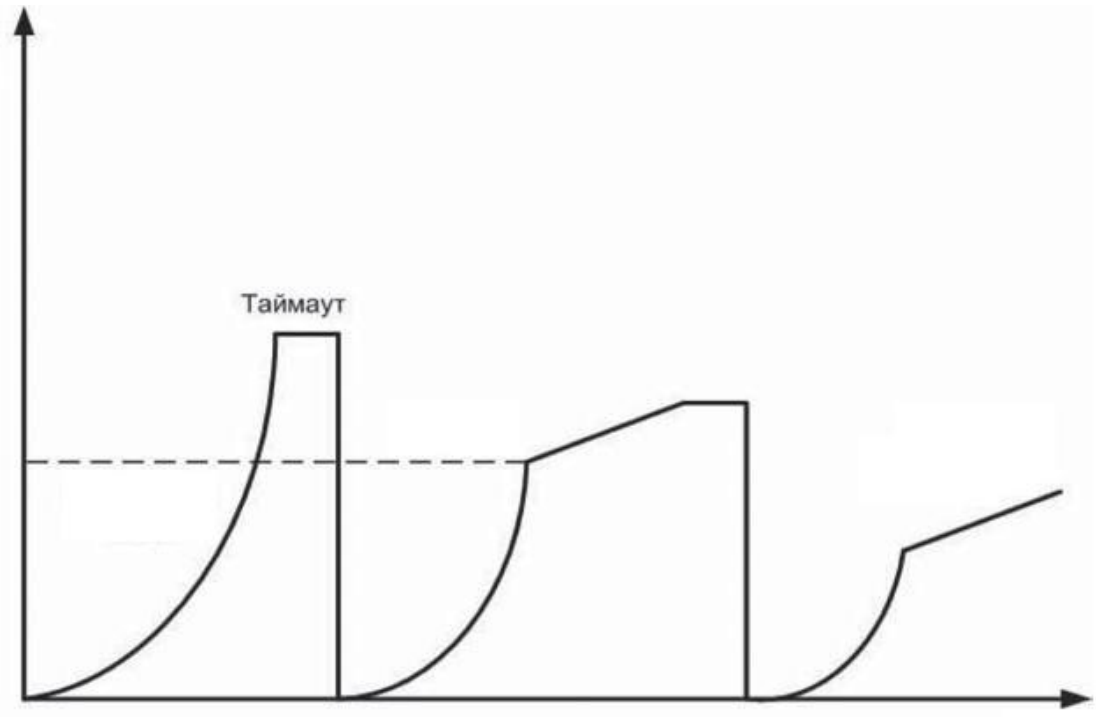

Рисунок 2.1 – Залежність розміру ковзаючого вікна від часу
в протоколі TCP Tahoe

TCP Reno. У протоколі TCP Reno при відсутності втрат розмір вікна

змінюється циклічно [11]. Розмір вікна збільшується до тих пір, поки не відбудеться втрата сегмента. TCP Reno має дві фази зміни розміру ковзаючого вікна: фаза повільного старту і фаза запобігання перевантаження. При отриманні відправником підтвердження доставки в момент часу $t+\Delta t$, поточне значення розміру ковзаючого вікна $s(t)$ обчислюється згідно з такими виразами:

$$s(t+\Delta t) = s(t) + 1, \text{ при } s < S,$$

$$s(t+\Delta t) = s(t) + 1/s(t), \text{ при } s \geq S.$$

Коли в результаті таймаута визначається факт втрати пакету, значення розміру ковзаючого вікна і порогового значення запуску фази запобігання перевантаження обчислюється таким чином:

$$s(t) = 1,$$

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

З іншого боку, коли протокол TCP Reno визначає факт втрати пакета відповідно до алгоритму швидкої повторної передачі, дані параметри обчислюються інакше:

$$S(t) = s(t)/2,$$

$$s(t) = S(t).$$

Після цього TCP Reno переходить до фази швидкого відновлення. У цій фазі розмір вікна збільшується на один пакет, коли приходить повторюване підтвердження. З іншого боку, $s(t)$ приймає значення $S(t)$, коли приходить не

повторюване підтвердження для пакета, посланого повторно.

У разі таймаута:

$$S(t) = s(t)/2,$$

$$s(t) = 1.$$

TCP NewReno. В даний час все більшого поширення набуває протокол TCP NewReno, описаний в документі RFC-3782, і використовує алгоритм швидкої повторної передачі і швидкого відновлення. У разі, коли доступна опція вибіркового підтвердження (SACK), відправник знає, які пакети слід переслати повторно у фазі швидкого відновлення. У відсутності опції SACK немає достатньої інформації щодо пакетів, які потрібно надіслати повторно [12, 13]. При отриманні трьох повторюваних підтверджень, відправник вважає відправлений пакет втраченим і посилає його повторно. TCP NewReno використовує змінну «recover», початкове значення якої дорівнює вихідному порядковому номеру пакета.

Протокол TCP NewReno використовує модифікований алгоритм TCP Reno у відправника, який полягає в обході очікування закінчення таймера повторної передачі під час множинних втрат у фазі швидкого відновлення. Це реалізовано за допомогою алгоритму швидкої повторної передачі і швидкого відновлення.

При доступності опції вибіркового підтвердження, відправник може повторно передавати пакети у фазі швидкого відновлення. У разі отримання трьох повторюваних пакетів підтверджень для одного відправленого пакета даних, такий пакет даних вважається втраченим і повинен бути відправлений повторно. Якщо втрачений один пакет з одного відправленого вікна даних і не відбулося переупорядковування пакетів, то підтвердження цього пакета буде означати успішну доставку всіх попередніх пакетів до переходу в фазу швидкої повторної передачі. Однак якщо втрачено кілька пакетів з одного

відправленого вікна даних, то підтвердження повторно посланого пакета підтверджує доставку деяких пакетів (але не всіх), відправлених до переходу в фазу швидкої повторної передачі і такі підтвердження називаються частковими.

Усунення очікування протоколом закінчення таймера повторної передачі під час множинних втрат у вікні досягнуто за рахунок фази швидкого відновлення, і опції вибіркових підтверджень [14-16]. У протоколі TCP Reno селективні пакети ACK виводять TCP з фази швидкого відновлення за допомогою зменшення розміру використовуваного вікна до розміру ковзаючого вікна. У протоколі TCP NewReno селективні пакети ACK не виводять TCP з фази швидкого відновлення, а сприймаються як ознака того, що пакет слідував відразу за підтвердженим пакетом у втраченій послідовності і повинен бути повторно переданий.

Таким чином, при множинній втраті пакетів з одного вікна даних, протокол TCP NewReno може відновлюватися без очікування таймаута, повторно передаючи один втрачений пакет до тих пір, поки всі втрачені пакети з вікна не будуть повторно передані. TCP NewReno залишається у фазі швидкого відновлення до повного підтвердження всіх ненадісланих даних на момент входу в цю фазу. Існує параметр, що обмежує кількість пакетів, які можуть бути послані у відповідь на один пакет ACK, незважаючи на розмір ковзаючого вікна відправника. Для протоколу TCP NewReno такий параметр має значення, рівне чотирьом поза фазою швидкого відновлення і двом у фазі швидкого відновлення (для відтворення поведінки протоколу TCP Reno).

TCP NewReno використовує нову змінну для повернення порядкового номера втраченого пакета (recover). Нижче представлені основні етапи реалізації такого алгоритму.

Коли відправник приймає три однакових пакета ACK і при цьому він не знаходиться у фазі швидкого відновлення, проводиться перевірка перекриття полем «Cumulative Acknowledgement» більшого діапазону, ніж

заданий змінною recover. Якщо це так, то запускається фаза швидкої повторної передачі, в якій значення *S* визначається відповідно до наступного виразу:

$$S = max\,(F/2,\ 2 \cdot SMSS),$$

де *SMSS* – максимальний розмір переданого сегмента даних;

F – обсяг відправлених, але ще не підтверджених даних.

Змінній «recover» надається порядковий номер останнього вдало переданого пакета і здійснюється перехід до п. 2.

Інакше (якщо такого перекриття немає), фаза швидкої повторної передачі не починається (тобто не змінюється значення *S*), не виконується перехід до п.2 (щоб передати втрачений сегмент), і згодом не робиться перехід до п.3 (при наступних повторюваних пакетах ACK).

Здійснюється запуск фази швидкої повторної передачі для передачі втраченого сегмента, після чого значення *s* встановлюється у відповідності з наступним виразом:

$$s = S + 2 \cdot SMSS\ .$$

Це дозволяє штучно збільшити розмір ковзаючого вікна на кілька сегментів (три), які покинули мережу і були буферизовані одержувачем.

Здійснюється запуск фази швидкого відновлення і, для кожного одержуваного нового дублюючого пакету ACK, значення *s* збільшується на величину *SMSS*, що штучно збільшує ковзаюче вікно, для обліку сегмента, який покинув мережу.

Продовження виконання фази швидкого відновлення, в результаті чого сегмент передається, якщо це дозволено новим значенням *s* і розміром вікна, заявленим одержувачем.

Кожен одержуваний пакет ACK для нових даних є підтвердженням

повторної передачі (див. п. 2).

Якщо пакет ACK підтверджує всі дані аж до позиції, зазначеної в змінній «recover», то він є підтвердженням всіх проміжних сегментів від втраченого сегмента і отримання трьох дубльованих ACK. Значення *s* встановлюється або в значення $min(S, F + SMSS)$ (опція 1), або в *S* (опція 2), де *S* дорівнює значенню, встановленому на етапі 1.

Необхідно зауважити, що значення *F* на етапі 1 належить до обсягу даних, недоставлених на етапі 1, коли відбувся перехід до фази швидкого відновлення, в той час як значення *F* на етапі 5 належить до обсягу даних недоставлених на етапі 5, після того як стався вихід з фази швидкого відновлення. Якщо вибрано 2, дозволяється вжити заходів щодо запобігання можливого сплеску трафіку в разі, якщо обсяг недоставлених даних в мережі набагато менше, ніж допускає ковзаюче вікно. Простим механізмом такого запобігання є обмеження числа пакетів, які можуть бути послані у відповідь на одне підтвердження.

2.2 Метод збільшення пропускної здатності для протоколу TCP NewReno

На рисунку 2.2 наведено послідовність етапів запропонованого методу збільшення пропускної здатності, у вигляді алгоритму. Такий метод може використовуватися для модифікації протоколу TCP NewReno. Суть модифікації полягає в динамічній зміні параметра *S*, який в оригінальній версії протоколу не змінюється. Така динаміка дозволяє реалізувати управління часом перебування протоколу в фазі повільного старту, коли він має найбільшу швидкість росту розміру вікна.

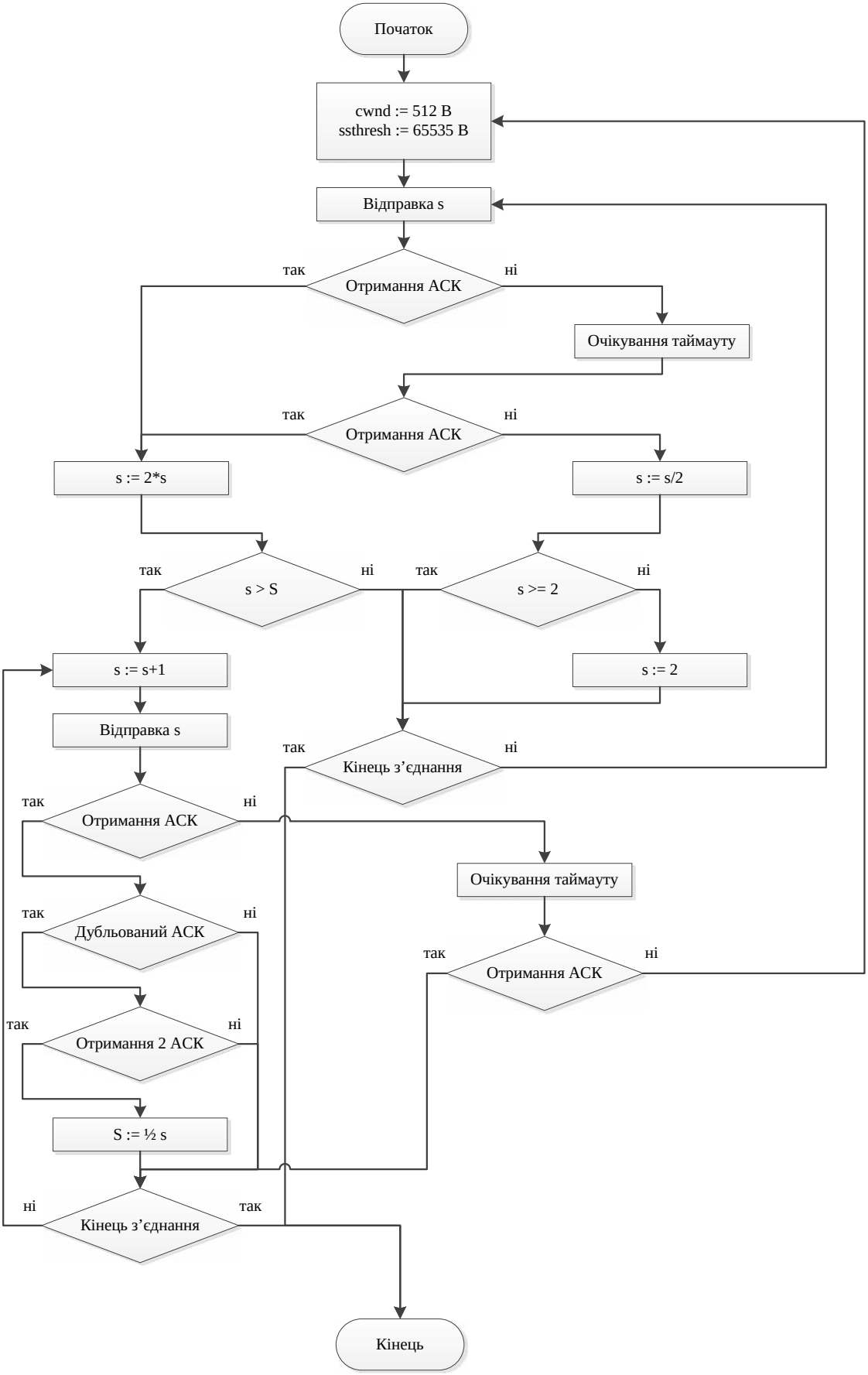

Рисунок 2.2 – Послідовність етапів методу збільшення пропускної здатності

Назва фази повільного старту протоколу TCP NewReno в дійсності не відповідає її реалізації, оскільки, перебуваючи в ній, протокол може виконувати експоненційне збільшення розміру свого ковзаючого вікна кожного разу при отриманні пакету ACK. Завданням фази повільного старту є якнайшвидше заповнення доступного з'єднанню розміру смуги пропускання каналу трафіком з'єднання, що особливо важливо при наявності в каналі фонового трафіку.

Перехід у фазу запобігання перевантаження з фази повільного старту виконується при досягненні розміром ковзаючого вікна *s* деякого порогового значення *S*. Фаза запобігання перевантаження передбачає лише лінійне збільшення розміру ковзаючого вікна, а також наявність таймера повторної передачі для кожного пакета даних, що надсилається.

Можна відзначити наступні ключові моменти в роботі запропонованого методу. При відкритті з'єднання параметрам *s* (початкове значення ковзаючого вікна) і *S* (порогове значення переходу з фази повільного старту в фазу запобігання перевантаження) присвоюються їх значення за замовчуванням. Після успішної передачі кожного вікна даних проводиться інкремент параметра *S* на одиницю (пакет). У разі ж втрати хоча б одного пакету з відправленого вікна даних, проводиться декремент значення *S*.

Таким чином, запропонована модифікація протоколу TCP NewReno має наступні переваги:

- на відміну від протоколу TCP Reno, дозволяє коректно обробляти події множинних втрат з одного вікна відправлених даних;
- має збільшену, в порівнянні з немодифікованою версією, пропускну здатність, за рахунок більш тривалого перебування в фазі повільного старту.

2.3 Обґрунтування та вибір засобів моделювання

Для вирішення сучасних завдань імітаційного моделювання мереж передачі даних існує досить широкий спектр програмних засобів: від

бібліотек функцій для стандартних компіляторів до спеціалізованих мов програмування. В даній атестаційній роботі моделювання буде проводитися за допомогою системи моделювання Ns2. Ця система має високу ефективність використання, гнучкість, а також широкий спектр підтримуваних протоколів.

Ns2 – це керована подіями, об’єктно-орієнтована система моделювання мережі, що дозволяє моделювати роботу локальних, глобальних, дротових і бездротових мереж [9]. У Ns2 на рівні ядра реалізовані всі сучасні мережні протоколи, дисципліни обслуговування черг (RED, WFQ, CBQ, SFQ), протоколи маршрутизації, джерела трафіку та інше.

2.4 Історія розвитку Ns2

У 1996 році почалася робота над проектом «Virtual InterNetwork Testbed» (VINT), організованим агентством «DARPA» під керівництвом цілої низки наукових організацій і центрів. Головною метою проекту VINT було створення програмного продукту, що дозволяє здійснювати імітаційне моделювання мереж зв’язку і володіє цілим рядом характеристик, серед яких висока продуктивність, масштабованість, візуалізація результатів і гнучкість. Як основа програмної реалізації, було обрано розроблюваний Університетом Каліфорнії в Берклі з 1989 року пакет «network simulator» (до 1995 року відомий як «REAL»). Для програмного продукту було вибрано ім’я «network simulator 2». Аналогічно попередникам, Ns2 розроблявся як програмне забезпечення (ПЗ) з відкритим вихідним кодом. Таке ПЗ поширювалось безкоштовно, без будь-яких обмежень на право використання, модифікації та розповсюдження третіми особами, що сприяло широкому поширенню, доповненню та підвищенню конкурентоздатності Ns2.

2.5 Архітектура Ns2

Ns2 є об’єктно-орієнтованим ПЗ, ядро якого реалізовано на мові C++. Мова скриптів «Object oriented Tool Command Language» (OTcl) використовується в якості інтерпретатора. Ns2 повністю підтримує ієрархію класів C++ (звану в термінах Ns2 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 таким же класом іншої. Використання двох мов програмування в Ns2 пояснюється наступними факторами. З одного боку, для детального моделювання протоколів необхідно використовувати системну мову програмування, що забезпечує високу швидкість виконання, гнучкість і здатність маніпулювати досить великими обсягами даних. З іншого боку, для зручності користувача, швидкості реалізації і модифікації різних сценаріїв моделювання, раціонально використовувати мову програмування більш високого рівня абстракції. Такий підхід є компромісом між зручністю використання і швидкістю. У Ns2 у якості системної мови використовується C++, що дозволяє забезпечити наступні аспекти:

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

У якості мови програмування високого рівня абстракції використовується мова скриптів OTcl, що дозволяє забезпечити ряд позитивних властивостей, притаманних мові Tcl/Tk (OTcl є об’єктно-орієнтованим розширенням мови Tcl):

- простота синтаксису;
- простота побудови сценарію моделювання;
- можливість з’єднання блоків, виконаних на системних мовах

програмування і просту маніпуляцію ними.

На рисунку 2.3 представлена загальна схема етапів моделювання в Ns2.

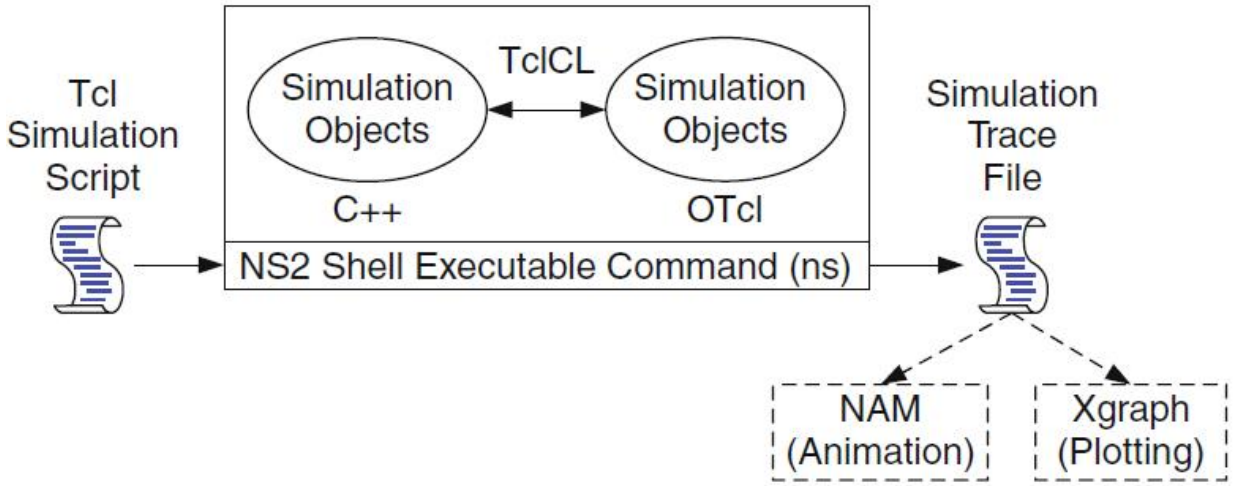

Рисунок 2.3 – Етапи моделювання в Ns2

Поєднання для спільного функціонування C++ і OTcl проводиться за допомогою TclCl (Classes Tcl). TclCl є інтерфейсом між об’єктами C++ і OTcl. Загальна архітектура Ns2, відповідна представленим вище підходам, наведена на рисунку 2.4.

Ns2 – керована подіями система моделювання. Будь-яка модель в Ns2 є сукупність мережних компонентів (вузлів, агентів, додатків, зв’язків), які обмінюються пакетами. Передача пакетів організована за допомогою планувальника подій. В даний час Ns2 підтримує чотири планувальника:

- проста таблиця вказівників (simple linked list);
- планувальник типу «heap»;
- календарна черга (calendar queue);
- планувальник реального часу (real-time).

У таблицю планувальника заносяться пакети, що містять адресу відправника і одержувача, а також очікуваний момент приходу пакету, і події, призначені для управління загальним ходом моделювання. Планувальник в хронологічному порядку обробляє кожен рядок таблиці.

Виняток становлять планувальники реального часу, що синхронізують свій модельний час з реальним. За замовчуванням створюється планувальник типу «календарна черга».

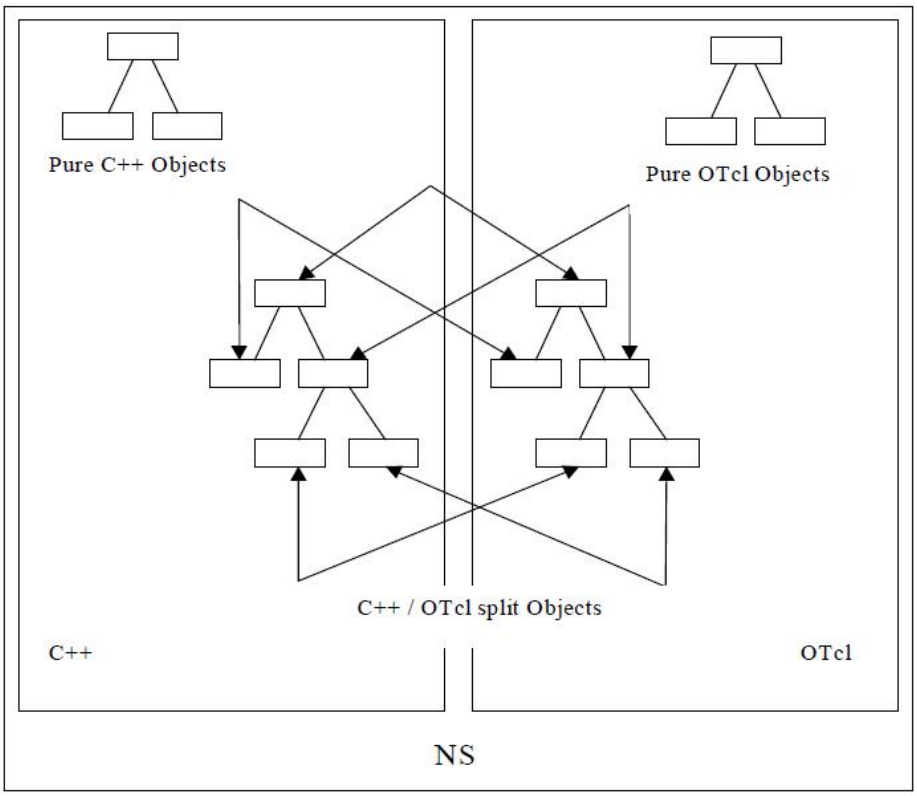

Рисунок 2.4 – Базова архітектура Ns2

2.6 Мова «Object Tcl»

У Tcl, значення можуть бути збережені в змінних і в подальшому використані в командах, що наведено на прикладі 2.1.

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

Приклад 2.1 – Збереження значень

У першому рядку змінній *a* було присвоєно значення «5», в другій за

допомогою операнда «$» було повернуто значення змінної *a* і присвоєно *b*. В останньому рядку результат команди «expr $a/$b» використовується як аргумент команди присвоєння змінної *x*.

Тип змінних в Tcl не вказується явно, за замовчуванням в залежності від значення присвоюються цілочисельний (integer) або строковий (string) типи. У разі використання іншого типу, його необхідно явно вказати. Команда поверне результат «0» (приклад 2.2), для правильного результату необхідно явно вказати тип (приклад 2.3).

```
expr 1/60
```

Приклад 2.2 – Команда без явного типу

```
expr 1.0/60.0
```

Приклад 2.3 – Команда з указанням типу

У Tcl включена підтримка структури вибору «якщо / то» («if / else»), а також циклів («for» і «foreach»). На прикладі 2.4 команда, яка є в тілі циклу, повториться 5 разів.

```
for {set i 0} {$i<5} {incr i}
{ <execute command> }
```

Приклад 2.4 – Підтримка структур вибору

Мова Tcl підтримує роботу з процедурами. Визначення процедури містить ключове слово «proc», ім’я, список аргументів і тіло процедури. В якості аргументів процедури можуть бути об’єкти, файли і змінні; процедура може повертати деякі значення, в такому випадку тіло закінчується командою «return».

Загальна структура оголошення процедури наведена на прикладі 2.5.

```
proc example {parametr1 parametr2 … }
{
global variable1 variable2 …
<commands>
return $something
}
```

Приклад 2.5 – Загальна структура оголошення процедури

Слід зазначити, що змінні, використовувані в тілі процедури і не визначені в списку прийнятих параметрів, повинні бути оголошені як глобальні.

2.7 Етапи моделювання

Моделювання мережі в Ns2 включає в себе етапи створення топології (вузлів, зв’язків, черг), об’єктів транспортного і прикладного рівнів (агентів і застосунків), а також збір даних (трасування і моніторинг). Нижче детально розглянуто кожен етап.

В системі Ns2 базовим класом моделювання є клас «Simulator». В даному класі визначено практично всі необхідні методи для моделювання, в тому числі методи створення та управління топологією. Під топологією в Ns2 мається на увазі набір вузлів і зв’язків між ними. Основним примітивом створення вузла є виклик методу «node» об’єкта класу «Simulator» (приклад 2.6).

```
set ns [new Simulator]
$ns node
```

Приклад 2.6 – Виклик методу «node» об’єкта класу «Simulator»

Команди для створення двох вузлів «n0» і «n1» наведено на прикладі 2.7.

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

Приклад 2.7 – Створення двох вузлів

Вузол є складовим об’єктом, процедура node класу «Simulator» створює вузол з об’єктів наступних класів: «Address Classifiers», «Multicast Classifiers», «MultiPath Classifier», «Hash Classifier», «Replicator». Кожен вузол має унікальну адресу («id_») і список сусідніх вузлів («neighbor_»).

За замовчуванням вузли в Ns2 створюються для групової передачі (рисунок 2.3). Для включення режиму одноадресного моделювання необхідно при створенні об’єкта «Simulator» вказати опцію «-multicast on» (приклад 2.8).

```
set ns [new Simulator -multicast on]
```

Приклад 2.8 – Включення режиму одноадресного моделювання

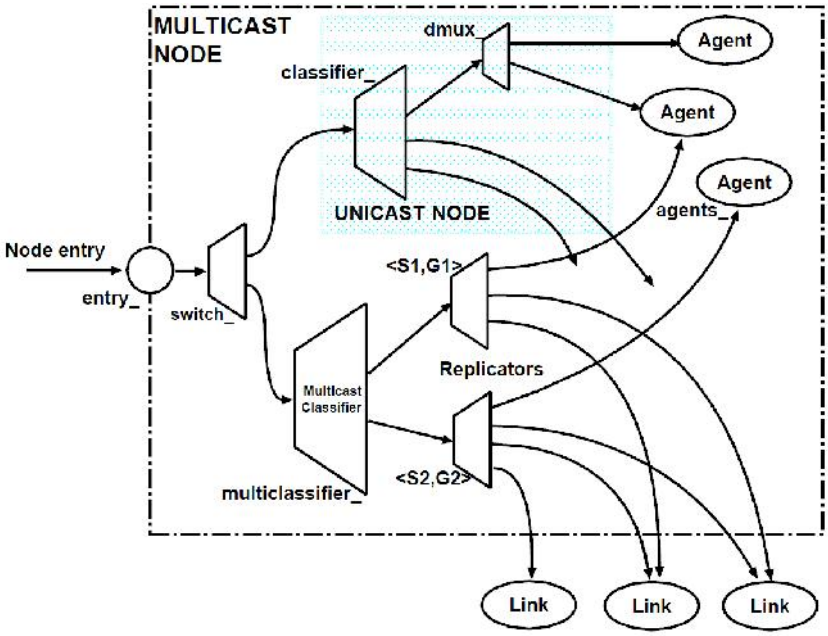

Рисунок 2.3 – Структура багатоадресного вузла

Для завершення створення топології необхідно визначити зв’язки. У Ns2 вихідна черга («queue») вузла реалізована як частина зв’язку, тому при створенні зв’язку користувач також зобов’язаний визначити тип черги.

В даний час підтримуються наступні дисципліни обслуговування черг:

- DropTail – видалення пакетів, що надходять, коли черга заповнена;
- Random Early Detect (RED) – видалення випадкових пакетів при значному збільшенні середньої довжини черги;
- Stochastic Fair Queuing (SFQ) – поділ трафіку на окремі черги типу «FIFO» за допомогою хеш-функції.

На рисунку 2.4 приведена структура симплексного зв’язку в Ns2. При створенні дуплексної лінії зв’язку, спільно організовуються дві симплексних, по одній на кожен напрямок. Усередині зв’язку пакет ставиться в чергу. Після цього він або скидається (передається агенту «Agent/Null» і вивільняється), або вибирається з черги і передається об’єкту затримки (агент «Delay»), який моделює затримку в з’єднанні. Далі обчислюється і оновлюється час життя пакета (параметр «TTL»).

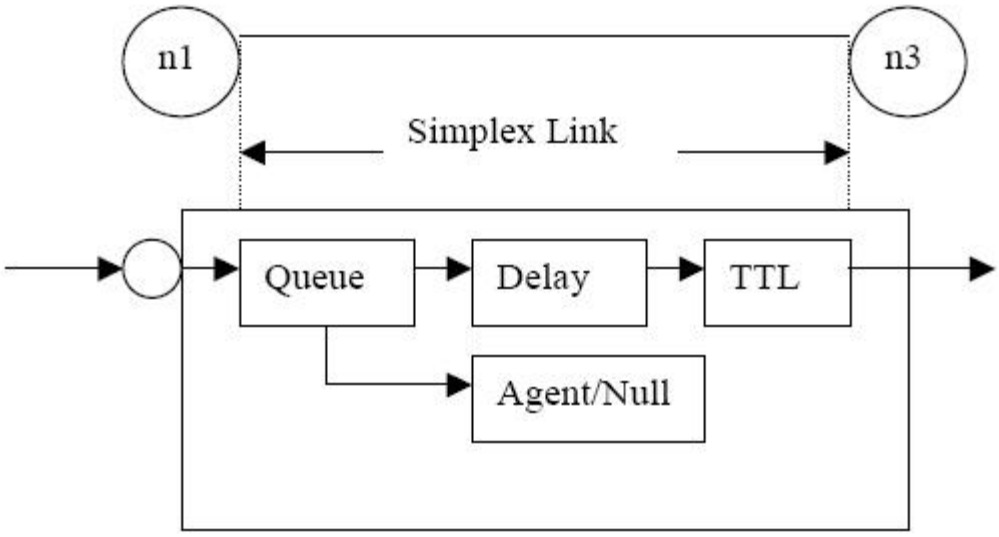

Рисунок 2.4 – Реалізація симплексного зв’язку в системі Ns2

Зв’язки створюються за допомогою наступної команди, що наведена на прикладі 2.9.

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

Приклад 2.9 – Створення зв’язків

Приклад створення дуплексної лінії зв’язку з механізмом управління чергою «DropTail» між вузлами «n0» і «n2» (перший рядок) і симплексної лінії зв’язку з механізмом управлінням чергою «RED» між вузлами «n1» і «n3» (другий рядок) наведено на прикладі 2.10.

```
$ns duplex-link $n0 $n2 15Mb 10ms DropTail
$ns simplex-link $n1 $n3 10Mb 5ms RED
```

Приклад 2.10 – Створення ліній зв’язку

Значення для смуги пропускання вказуються безрозмірним числом або з можливістю використання приставок k (кіло), M (мега), b (біт) і B (байт). При описі затримки також можуть використовуватися приставки m (мілі) і u (мікро).

Агенти представляють кінцеві точки відправника і одержувача пакетів на мережевому рівні, а також використовуються для реалізації протоколів на інших рівнях. Найбільш часто використовуваними агентами в Ns2 є агенти TCP і UDP.

У Ns2 агенти TCP класифікуються на дві основні групи: односпрямовані (one-way) і двонаправлені (two-way). Однонаправлені агенти діляться на агенти-відправники (senders) і агенти-приймачі (sinks). Класифікація наведена на рисунку 2.5.

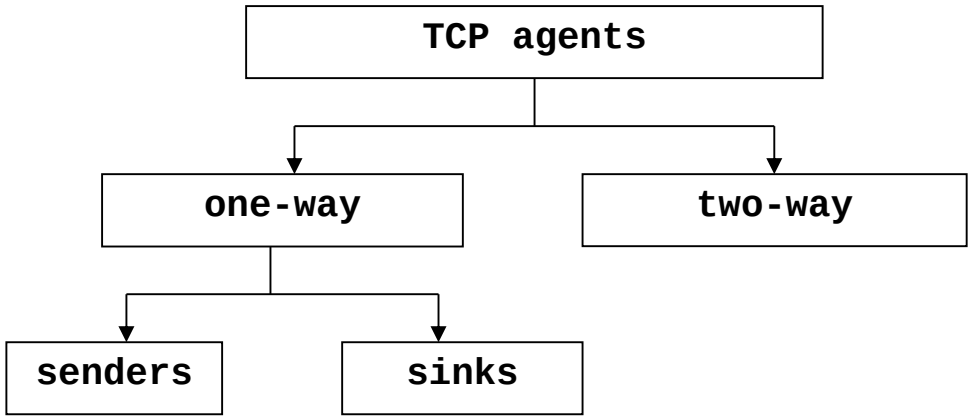

Рисунок 2.5 – Класифікація агентів TCP

В даний час підтримуються наступні реалізації агентів-відправників:

- Agent/TCP – TCP Tahoe;
- Agent/TCP/Reno –TCP Reno;
- Agent/TCP/Newreno –TCP NewReno;
- Agent/TCP/Sack1 – TCP з вибірковим підтвердженням;
- Agent/TCP/Linux – TCP- відправник з підтримкою модулів управління перевантаженнями ядра Linux.

Також підтримуються наступні агенти-приймачі:

- Agent/TCPSink;
- Agent/TCPSink/DelAck;
- Agent/TCPSink/Sack1;
- Agent/TCPSink/Sack1/DelAck.

Створення агента означає створення екземпляра відповідного агенту класу. Надалі стан агента може бути змінено за допомогою методів даного класу і Tcl функції «set».

За замовчуванням в системі Ns2 визначені наступні параметри:

- Agent/TCP set window_ 20 – максимальний розмір вікна;
- Agent/TCP set windowInit_ 1 – ініціалізація (скидання) значення розміру вікна;
- Agent/TCP set windowThresh_ 0.002 – використовується при

обчисленні середнього значення вікна;

- Agent/TCP set overhead_ 0 – значення, відмінне від 0, додає випадкове час між посилками;
- Agent/TCP set packetSize_ 1000 – розмір пакетів, який використовується відправником;
- Agent/TCP set maxrto_ 64 – максимальне значення таймера повторної передачі (сек);
- Agent/TCP set dupacks_ 0 – лічильник повторних ACK;
- Agent/TCP set ack_ 0 – найвищий отриманий ACK;
- Agent/TCP set cwnd_ 0 – розмір перевантажувального вікна;
- Agent/TCP set awnd_ 0 – середнє значення перевантажувального вікна;
- Agent/TCP set ssthresh_ 0 – поріг повільного старту;
- Agent/TCP set rtt_ 0 – значення часу передачі пакету;
- Agent/TCP set srtt_ 0 – середнє значення часу передачі пакету.

Приклад створення і модифікації агента TCP наведено на прикладі 2.11.

```
set newtcp [new Agent/TCP]
$newtcp set window_ 20
$newtcp target $dest
$newtcp set portID_ 1
```

Приклад 2.11 – Створення і модифікація агента TCP

Агенти приєднуються до вузлів і з’єднуються між собою за допомогою методів «attach-agent» і «connect» класу «Simulator» відповідно.

Наступний приклад (приклад 2.12) ілюструє створення і підключення двох TCP агентів.

Двонаправлені агенти симетричні, тобто кожен агент має властивості як відправника, так і адресата. Двонаправлені агенти представлені класом «Agent/TCP/FullTcp». На даний момент підтримується лише реалізація TCP Reno (приклад 2.13).

```
#створення TCP агентів
set tcp1 [new Agent/TCP]
set sink1 [new Agent/TCPSink]
#приєднання агентів до вузлів
$ns attach-agent $n0 $tcp1
$ns attach-agent $nl $sink1
#підключення агентів
$ns connect $tcp1 $sink1
```

Приклад 2.12 – Створення і підключення двох TCP агентів

```
#створення агентів
set src [new Agent/TCP/FullTcp]
set sink [new Agent/TCP/FullTcp]
$ns attach-agent $node0 $src
$ns attach-agent $node1 $sink
$ns connect $src $sink
```

Приклад 2.13 – Створення агентів

Вище наведено приклад створення двонапрямлених агентів TCP. Агенти-відправники і адресати є екземплярами класу «Agent/TCP/FullTcp». Прив’язка агентів до вузлів і підключення ідентичні односпрямованим агентам.

Аналогічно TCP створюються і підключаться UDP агенти, як агента-приймача використовується «Agent/Null» (приклад 2.14).

```
set udp1 [new Agent/UDP]
set sink1 [new Agent/Null]
$ns attach-agent $n0 $udp1
$ns attach-agent $nl $sink1
$ns connect $udp1 $sink1
```

Приклад 2.14 – Підключення агентів

Агенти TCP і UDP представляють в Ns2 транспортний рівень. Представниками прикладного рівня є об’єкти класу «додаток» (application). Всі додатки діляться на два типи (див. рисунок 2.6): «джерела трафіку» (traffic generators) і «моделі додатків» (simulated applications).

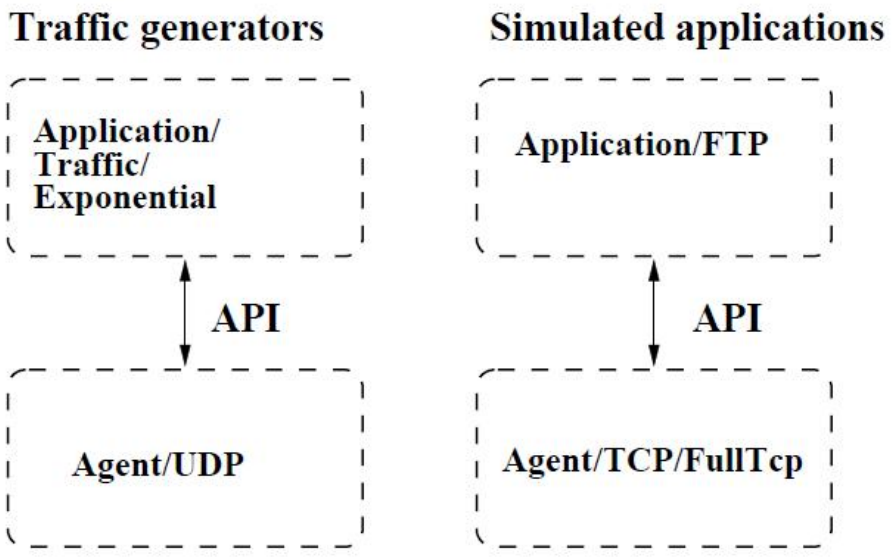

Рисунок 2.6 – Приклад структури застосунка

Клас «TrafficGenerator» представлений як на рівні C++ так і на рівні OTcl. На рівні C++ клас «TrafficGenerator» є спадкоємцем класу «Application» і визначається чином, що наведено на прикладі 2.15.

```
class TrafficGenerator : public Application {
public:
TrafficGenerator();
virtual double next_interval(int &) = 0;
virtual void init() {}
virtual double interval() { return 0; }
virtual int on() { return 0; }
virtual void timeout();
virtual void recv() {}
virtual void resume() {}
protected:
virtual void start();
virtual void stop();
double nextPkttime_;
int size_;
int running_;
TrafficTimer timer_;
};
```

Приклад 2.15 – Визначення класу

Функція «next_interval(int&)» повертає час до моменту створення наступного пакета, а також встановлює його розмір в байтах. Метод «start()»

викликає «init()» і запускає таймер. Функція «timeout()» відправляє пакет і планує наступний таймаут. Функція «stop()» зупиняє передачу пакетів.

В даний час в Ns2 визначені чотири спадкоємця класу «TrafficGenerator»:

- EXPOO_Traffic – генерує ON/OFF трафік з експоненційним законом розподілу: пакети передаються з певною інтенсивністю протягом ON-періоду і відсутня передача протягом періоду OFF; тривалість періодів ON/OFF підпорядковується експоненційним законом розподілу; пакети мають сталу суму;
- POO_Traffic – генерує ON/OFF трафік відповідно до закону Парето; тривалість періодів ON/OFF підкоряється закону Парето і має співвідношення 20:80; розмір пакетів не змінюється;
- CBR_Traffic – генерує трафік з постійною інтенсивністю і постійним розміром пакетів;
- TrafficTrace – генерує трафік згідно з даними трейс-файлу; кожен запис в трейс-файлі містить два 32-бітових поля: перше вказує час в мікросекундах до генерації наступного пакета, друге – його довжину в байтах.

Зазначені вище класи можуть бути створені на рівні OTcl.

Об’єкти експоненційного ON/OFF трафіку реалізовані в класі «Application/Traffic/Exponential». Об’єкти даного класу мають наступні параметри:

- packetSize_ – розмір пакетів, що генеруються;
- burst_time_ – середній час періоду ON;
- idle_time_ – середній час періоду OFF;
- rate_ – інтенсивність трафіку протягом періоду ON.

Об’єкти ON/OFF трафіку, що підкорюються принципу Парето, реалізовані в класі «Application/Traffic/Pareto». Параметри аналогічні експоненційному класу, за винятком параметра «shape_» – коефіцієнта розподілу Парето.

Приклад створення джерела трафіку наведено на прикладі 2.16 нижче.

```
set udp1 [new Agent/UDP]
set p [new Application/Traffic/Pareto]
$p set packetSize_ 210
$p set burst_time_ 500ms
$p set idle_time_ 500ms
$p set rate_ 200k
$p set shape_ 1.5
$p attach-agent $udp1
```

Приклад 2.16 – Створення джерела трафіку

Модель трафіку з постійною інтенсивністю представлена в класі «Application/Traffic/CBR». В наявності є такі параметри:

- rate_ – інтенсивність відправки пакетів;
- interval_ – інтервал між відправленням пакетів (опціонально);
- packetSize_ – розмір пакетів, що генеруються;
- random_ – включення випадкових перешкод в планувальник (за замовчуванням має значення «off»);
- maxpkts_ – максимальна кількість пакетів, що відправляються.

Об’єкт «Traffic Trace» реалізований в OTcl класі «Application/Traffic/Trace». Прив’язка трейс-файлу до примірника даного класу проводиться за допомогою методу «attach-tracefile» (приклад 2.17).

```
set tfile [new Tracefile]
$tfile filename example-trace
set t1 [new Application/Traffic/Trace]
$t1 attach-tracefile $tfile
```

Приклад 2.17 – Прив’язка трейс-файлу до класу

Тип моделі застосунків представлений в Ns2 класами «Application/FTP» і «Application/Telnet». Передача пакетів контролюється потоками TCP і алгоритмом управління перевантаженнями.

Клас «Application/FTP» моделює передачу великих обсягів даних. Клас

містить наступні методи:

- attach-agent – приєднання застосунка FTP до агента;
- start – запуск застосунка FTP за допомогою виклику функції TCP агента «send (-1)»;
- stop – зупинення передачі;
- produce n – установка значення лічильника відправлених пакетів у n;
- producemore n – збільшення лічильника на n;
- send n – збільшення лічильника відправлених пакетів на n байт.

Єдиним доступним параметром класу є «maxpkts» – максимальна кількість пакетів, згенерованих джерелом.

Об’єкти класу «Application/Telnet» генерують пакети двома способами. Якщо параметр «interval_» не дорівнює нулю, то межпакетний інтервал вибирається за експоненційним законом розподілу із середнім значенням, що дорівнює «interval_». В іншому випадку інтервал між пакетами вибирається згідно розподілу «tcplib». Методи «attach-agent», «start», «stop» приєднують додаток до агента, починають і зупиняють генерацію пакетів відповідно (приклад 2.18).

```
set telnet [new Application/Telnet]
$telnet attach-agent $tcp1
$telnet start
$telnet stop
```

Приклад 2.18 – Використання методів «attach-agent», «start», «stop»

Крім джерел трафіку і додатків, організація передачі даних на прикладному рівні можлива за допомогою віртуальних функцій класу «Agent»:

- send (int nbytes) – відправка «nbytes» агенту того ж рівня; для TCP агентів при «nbytes==-1» виконується нескінченна передача;
- sendmsg(int nbytes, const char* flags = 0) – відправка «nbytes» в режимі, що задається полем «flags»;

- close () – закриття з’єднання;
- listen () – переклад агента в стан очікування з’єднання.

Середовище Ns2 підтримує дві основні можливості спостереження: трейси і монітори. Трейси дозволяють записувати пакети, при виникненні певних подій, таких як, наприклад, скидання пакета або постановка його в чергу до каналу зв’язку. Монітори надають методи збору чисельних значень, таких як кількість скидів пакетів або число прибулих пакетів в чергу. Монітор може бути використаний при отриманні цих значень для всіх пакетів, або тільки для певних потоків (потоковий монітор).

Всі події при моделюванні, при використанні трейсів, можуть бути записані в файл наступними командами, що наведені на прикладі 2.19.

```
set trace_all [open all.dat w]
$ns trace-all $trace_all
$ns flush-trace
close $trace_all
```

Приклад 2.19 – Запис подій у файл

У першому рядку вихідний файл відкривається для запису і йому призначається дескриптор. Потім зазначені події записуються в файл, що визначений дескриптором. В кінці моделювання необхідно скинути буфер трасування і закрити файл. Дані команди, як правило, виконуються в окремій заключній процедурі. Якщо після їх використання створюються канали зв’язку, то в них додаються додаткові об’єкти для трасування («EnqT», «DeqT», «DrpT» і «RecvT»), як показано на рисунку 2.7.

Ці об’єкти потім записуються в трейс-файл кожного разу, коли вони отримують пакет. Формат трейс-файлу наступний, у відповідності до прикладу 2.20.

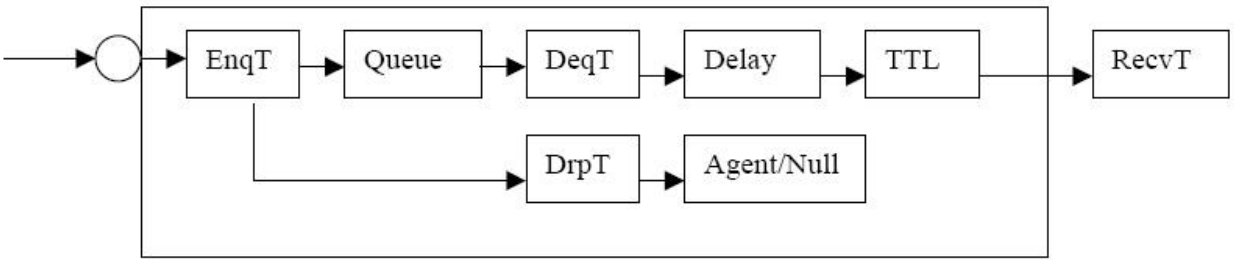

Рисунок 2.7 – Реалізація зв’язку в Ns2 при включеному трасуванні

```
+ 1.84375 0 2 cbr 210  ------- 0 0.0 3.1 225 610
- 1.84375 0 2 cbr 210  ------- 0 0.0 3.1 225 610
r 1.84471 2 1 cbr 210  ------- 1 3.0 1.0 195 600
r 1.84566 2 0 ack 40   ------- 2 3.2 0.1 82  602
+ 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
- 1.84566 0 2 tcp 1000 ------- 2 0.1 3.2 102 611
+ : постановка в чергу
- : выбірка із черги
d : скидання пакета
r : отримання пакета
```

Приклад 2.20 – Формат трейс-файлу наступний

Полями трейс-файлу є: тип події, час моделювання при виникненні події, вузли відправника і одержувача, тип пакета (протокол, дія або джерело трафіку), розмір пакету, флаги, ідентифікатор потоку, адреси відправника та одержувача, номер послідовності і ідентифікатор пакета.

На додаток до трасування всіх подій моделювання, також можливо створити об’єкт трейса між конкретним відправником і отримувачем командою, що наведена на прикладі 2.21.

```
$ns create-trace <type> <file> <src> <dest>
```

Приклад 2.21 – Створення об’єкту трейса між відправником і отримувачем

Тип трейса може бути наступним:

- Enque – прибуття пакета (наприклад в чергу);
- Deque – відправлення пакета (наприклад з черги);

- Drop – скидання пакета;
- Recv – отримання пакету адресатом.

За допомогою моніторів черги можна відстежувати статистику прибуття, відправлення і скидів, як пакетів, так і байтів. Додатково монітор черги здатний зберігати інтегральну довжину черги з плином часу.

Нижче показаний приклад (приклад 2.22) створення монітора черги між вузами «n0» та «n1».

```
set qmon0 [$ns monitor-queue $n0 $n1]
```

Приклад 2.22 – Створення монітора черги

Прибуття пакетів і скидання байтів відстежується наступними командами, що наведено на прикладі 2.23.

```
set parr [$qmon0 set parrivals_]
set bdrop [$qmon0 set bdrops_]
```

Приклад 2.23 – Відстеження прибуття пакетів і скидання байтів

Слід зазначити, що поряд з привласненням значення змінної, команду «set» можна використовувати для отримання значення змінної. У наведеному вище прикладі команда «set» використовується для отримання значення змінної «parrivals», певної в класі монітора черзі.

Монітор потоку відстежує статистику для конкретного потоку, а не для всього трафіку, як монітор черги. Визначення приналежності потоку здійснюється за допомогою класифікатора, після пакет потрапляє в монітор потоку.

Монітор потоку створюється і прив’язується до певної зв’язку наступним чином, що наведено на прикладі 2.24.

Дані команди визначені в класі «Simulator», змінні і команди класу «Flowmonitor» можуть бути використані тільки після створення і прив’язки

монітора до каналу зв’язку. Наприклад, щоб отримати дамп вмісту монітора потоку (всіх потоків), використовується команда, що наведена на прикладі 2.25.

```
set fmon [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $n1 $n3] $fmon
```

Приклад 2.24 – Створення і прив’язка монітора потоку

```
$fmon dump
```

Приклад 2.25 – Отримання дампу вмісту монітора потоку

Для збору статистики певного потоку необхідно визначити класифікатор з ідентифікатором потоку (приклад 2.26).

```
set fclassifier [$fmon classifier]
set flow [$fclassifier lookup auto 0 0 1]
```

Приклад 2.26 – Визначення класифікатору з ідентифікатором потоку

2.8 Управління моделюванням

Після створення топології, конфігурації агентів та інших дій, має бути задано розклад початку, зупинки та інших подій моделювання.

Моделювання може бути розпочато і зупинено наступними командами, що наведені на прикладі 2.27.

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

Приклад 2.27 – Запуск і зупинка процесу моделювання

Перша команда запускає в кінці моделювання процедуру «finish», а друга безпосередньо починає моделювання. Кінцева 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 постпроцесінг даних і будувати по ним графік.

Кінцева процедура повинна містити, принаймні, такі елементи, що наведено на прикладі 2.28.

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

Приклад 2.28 – Кінцева процедура

Інші події, такі як час початку і зупинки роботи клієнтів можна задати наступним чином, що наведено на прикладі 2.29.

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

Приклад 2.29 – Настроювання подій

Для зчитування поточного часу в Ns2 визначена змінна «now», що дозволяє організувати періодичний запуск процедур. Нижче наведено приклад процедури з періодом запуску 5 секунд (приклад 2.30).

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

Приклад 2.30 –Приклад процедури з періодом запуску 5 секунд

3 МОДЕЛЮВАННЯ ЗАПРОПОНОВАНИХ РЕЗУЛЬТАТІВ

3.1 Опис поставлених завдань та сценарію

При моделюванні буде використана одна з найбільш поширених реалізацій TCP Reno, а також більш сучасні реалізації TCP NewReno, TCP Westwood, TCP Hybla і модифікація протоколу TCP NewReno на базі запропонованого методу.

В якості системи моделювання була обрана система Ns2, що поєднує в собі високу продуктивність, візуалізацію отриманих результатів і гнучкість.

Для моделювання необхідного режиму роботи буде використана певна топологія, що має критичний ділянку для процесу передачі даних – канал з обмеженою пропускною спроможністю, за яким дані передаються до одержувача. Сумарна пропускна здатність відправників даних буде значно перевищувати пропускну спроможність цього каналу.

Для заданої топології мережі будуть створені сценарії моделювання, які враховують основні характеристики наявних з’єднань і дозволяють спостерігати за змінами в роботі різних реалізацій протоколу TCP з плином часу. При цьому існує можливість зміни режимів і параметрів реалізацій протоколу TCP.

Існує ряд параметрів протоколу TCP, що визначають ступінь його ефективності. До них можна віднести кількість втрачених пакетів, довжину черг, розмір вікна та інші. Проте всі вони є окремими випадками загальної ефективності транспортного протоколу. Сумарним параметром, що визначає продуктивність тієї чи іншої реалізації протоколу, є коефіцієнт завантаження каналу. У даній роботі у всіх сценаріях буде використовуватися така характеристика.

Отримані експериментальні дані є основою для розробки рекомендацій по використанню конкретних реалізацій протоколу TCP і відповідних

програмно-апаратних засобів.

3.2 Розробка топології

У контексті поставленого завдання, топологія експериментальної мережі повинна відповідати таким умовам:

- наявність відправників і одержувачів TCP трафіку;
- наявність в каналі фонового трафіку;
- єдність маршруту від відправників до адресатів;
- наявність проміжного маршрутизатора.

Топологія комп'ютерної мережі, що відповідає зазначеним вимогам, представлена на рисунке 3.1.

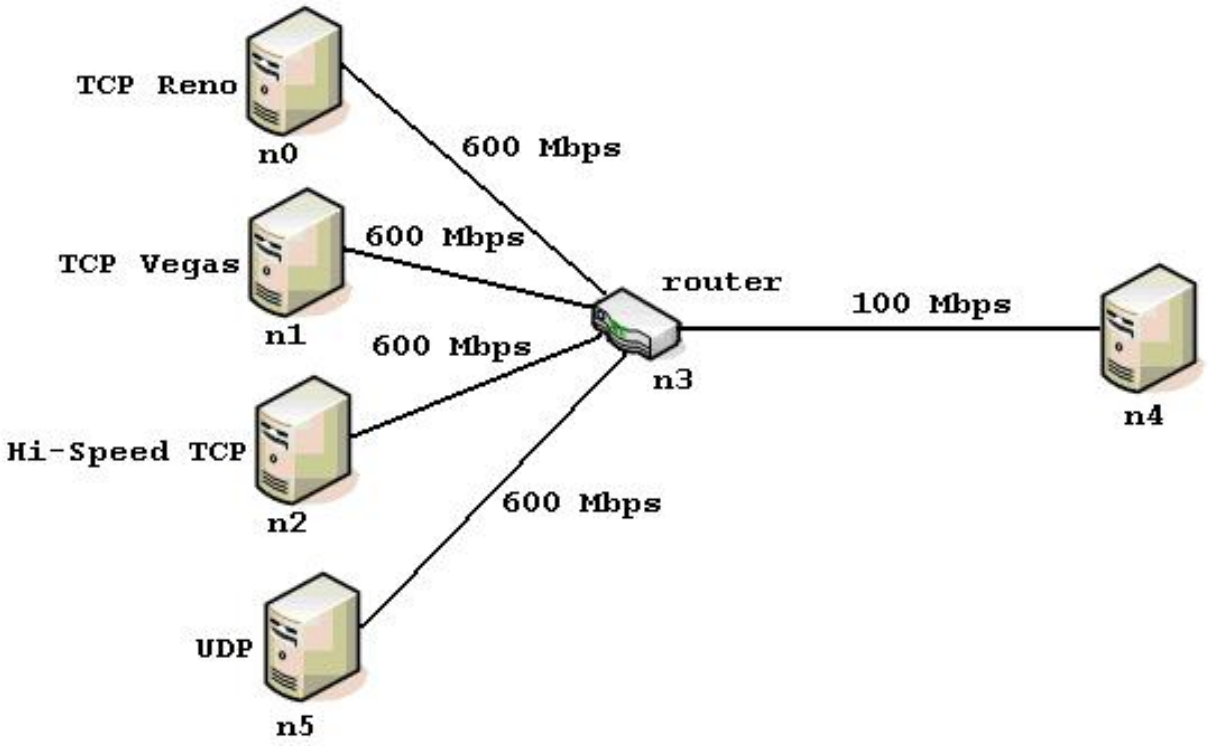

Рисунок 3.1 – Топологія мережі

У розробленій топології «n0», «n1», «n2», «n5» – вузли-відправники

даних, «n4» – адресат, «n3» – маршрутизатор. Всі відправники з’єднані з маршрутизатором каналами пропускною спроможністю 600 Мбіт/с. Транспортним протоколом для доставки даних від відправників «n0», «n1», «n2» адресату «n4» є TCP, відправник «n5» використовує протокол ненадійної доставки UDP. Об’єктами прикладного рівня для TCP з’єднань виступають додатки FTP, для UDP – джерело трафіку з постійною бітовою швидкістю (CBR). Додатки FTP дозволяють організувати безперервну передачу даних без обмеження інтенсивності, інтенсивність джерела трафіку CBR вказується при ініціалізації.

Маршрутизатор здійснює пересилку отриманих пакетів одержувачу через канал з пропускною спроможністю 100 Мбіт/с. Механізмом обслуговування черг, використовуваним в маршрутизаторі, є алгоритм DropTail, який моделює стандартний буфер FIFO, який реалізує принцип обслуговування за правилом «перший надійшов – перший пішов», і при досягненні розміром черги встановленого граничного значення, відкидає знову пакети, що надходять.

3.3 Створення сценаріїв моделювання

Все сценарії, що моделюються, розбиті на наступні типи:

- сценарії для роботи в каналі одного з’єднання – даний тип призначений для отримання характеристик кожної реалізації протоколу при вільному каналі;
- сценарії для роботи в каналі кількох з’єднань – метою даного типу є отримання відношення завантаження каналу декількома реалізаціями протоколу TCP, в якості фонового трафіку можна розглядати з’єднання, відмінні від з’єднання, що аналізуєтсья;
- сценарії декількох з’єднань TCP і потоків UDP – дані сценарії моделюють найбільш характерні для сучасних мереж ситуації, фоновим є UDP трафік.

У контексті обраної топології слід виділити наступні параметри, реалізація яких необхідна у всіх в сценаріях:

- пропускна здатність каналів між відправниками даних і маршрутизатором – 600 Мбіт/с;
- пропускна здатність каналу між маршрутизатором і адресатом – 100 Мбіт/с;
- затримка в каналі зв’язку маршрутизатора і відправників – 0,5 мс, маршрутизатора і адресата – 1 мс;
- максимальна довжина черги маршрутизатора – 20 сегментів;
- інтервал отримання параметрів, що моделюються – 0,8 с.

Сценарії моделювання знаходяться в файлах з розширенням «.tcl». Детальний процес створення сценарію моделювання для даного завдання відповідно до описаних вище методів роботи системи Ns2 і мови Tcl наведено нижче.

Створення нового об’єкта моделювання виконується відповідно до прикладу 3.1.

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

Приклад 3.1 – Створення нового об’єкта моделювання

Відкриття файлів для запису одержуваних параметрів і призначення їм дескрипторів наведено на прикладі 3.2.

```
set f0 [open out0.tr w]
set f1 [open out1.tr w]
set f2 [open out2.tr w]
set f3 [open out3.tr w]
```

Приклад 3.2 – Відкриття файлів

Наступна ділянка коду програми (приклад 3.3) описує визначену топологію мережі. В даному випадку це шість вузлів і лінії зв’язку, що їх

з’єднують, з параметрами, зазначеними вище.

```
set n0 [$ns node]
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
$ns duplex-link $n0 $n3 0.6Gb 0.5ms DropTail
$ns duplex-link $n1 $n3 0.6Gb 0.5ms DropTail
$ns duplex-link $n2 $n3 0.6Gb 0.5ms DropTail
$ns duplex-link $n5 $n3 0.6Gb 0.5ms DropTail
$ns duplex-link $n3 $n4 0.1Gb 1ms DropTail
```

Приклад 3.3 – Опис топології мережі

Установка довжини черги маршрутизатора наведена на прикладі 3.4.

```
$ns queue-limit $n3 $n4 20
```

Приклад 3.4 – Установка довжини черги маршрутизатора

Створення з’єднань TCP і UDP включає в себе створення агентів відправників і адресатів TCP і UDP необхідної реалізації протоколу, визначення вузлів, в яких знаходяться агенти, установка з’єднання між ними, щоб встановити розмір вікна. Для кожного з’єднання це реалізується наступними командами у відповідності до приклада 3.5.

```
set tcp1 [new Agent/TCP/Newreno]
$ns attach-agent $n0 $tcp1
set sink0 [new Agent/TCPSink]
$ns attach-agent $n4 $sink0
$ns connect $tcp1 $sink0
$tcp1 set window_ 1500
```

Приклад 3.5 – Створення з’єднань TCP і UDP

Залежно від реалізації протоколу TCP вибирається відповідний агент. У разі реалізацій TCP Westwood і TCP Hybla задається агент «Agent/TCP/Linux», а реалізація вибирається методом «select_ca» (приклад 3.6).

```
$tcp4 select_ca westwood
```

Приклад 3.6 – Вибір агента

Генерації даних в з’єднанні TCP здійснюється за допомогою застосунків FTP. Опис складається зі створення екземпляра класу «Application/FTP» і прив’язки до певного TCP агенту (приклад 3.7).

```
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
```

Приклад 3.7 – Встановлення застосунку FTP

Програма також містить опис двох процедур. Процедура «finish» виконує дії, необхідні для коректного завершення моделювання (закриття файлів з даними, скидання буфера трасування) і здійснює виклик програми побудови графіків отриманих результатів (приклад 3.8).

```
proc finish {} {
global f0 f1 f2 f3
#Close the output files
close $f0
close $f1
close $f2
close $f3
#Call xgraph to display the results
exec xgraph -bg white -lw 1 -zg black out3.tr &
#-geometry 800x400 &
        exit 0
      }
```

Приклад 3.8 – Процедура «finish»

Наступна процедура «record» (приклад 3.9) здійснює підрахунок завантаження каналу кожним з’єднанням і запис у файли з певним інтервалом відповідних результатів. Для цього змінній «time» присвоюється значення необхідного інтервалу часу, поточний час зчитується в змінну «now». В файли записуються значення змінної «now» і кількість даних, отриманих

приймачем в форматі Мбіт/с. Після здійснюється скидання приймачів і повторний виклик процедури через заданий у змінній «time» інтервал.

```
proc record {} {
global sink0 sink1 sink2 sink3 f0 f1 f2 f3
#Get an instance of the simulator
set ns [Simulator instance]
set time 0.1
set bw0 [$sink0 set bytes_]
set bw1 [$sink1 set bytes_]
set bw2 [$sink2 set bytes_]
set bw3 [$sink3 set bytes_]
#Get the current time
set now [$ns now]
#Calculate the bandwidth and write it to the files
puts $f0 "$now [expr $bw0/$time*8/1000000]"
puts $f1 "$now [expr $bw1/$time*8/1000000]"
puts $f2 "$now [expr $bw2/$time*8/1000000]"
puts $f3 "$now [expr $bw3/$time*8/1000000]"
#Reset the bytes_ values on the traffic sinks
$sink0 set bytes_ 0
$sink1 set bytes_ 0
$sink2 set bytes_ 0
$sink3 set bytes_ 0
#Re-schedule the procedure
$ns at [expr $now+$time] "record"
}
```

Приклад 3.9 – Процедура «record»

Розклад моделювання буде відрізнятися для кожного типу сценаріїв. Спільними будуть час запуску процедур «record» і «finish».

3.4 Результати моделювання

3.4.1 Моделі з одиничними з’єднаннями

Нижче наведені результати ряду експериментів, в яких проводилося моделювання одиночних з’єднань протоколом TCP. У каналі встановлювалося одне з’єднання TCP з додатком FTP. Мета даних експериментів – отримання «чистих» графіків ефективності моделей різних реалізацій протоколу TCP для подальшого аналізу і порівняння з роботою при наявності в каналі фонового трафіку.

На рисунку 3.2 представлений графік залежності завантаженості каналу від часу при наявності в ньому одного TCP з’єднання з додатком FTP. В даному прикладі використовується реалізація TCP Reno. Передача даних джерелом трафіку починається і закінчується в моменти часу 5 і 25 с відповідно. На графіку можна спостерігати «пилкоподібну» криву, що відображає завантаженість каналу в часі. З’єднання з реалізацією TCP Reno практично весь період активності джерела трафіку займає від 95 до 100% пропускної спроможності каналу.

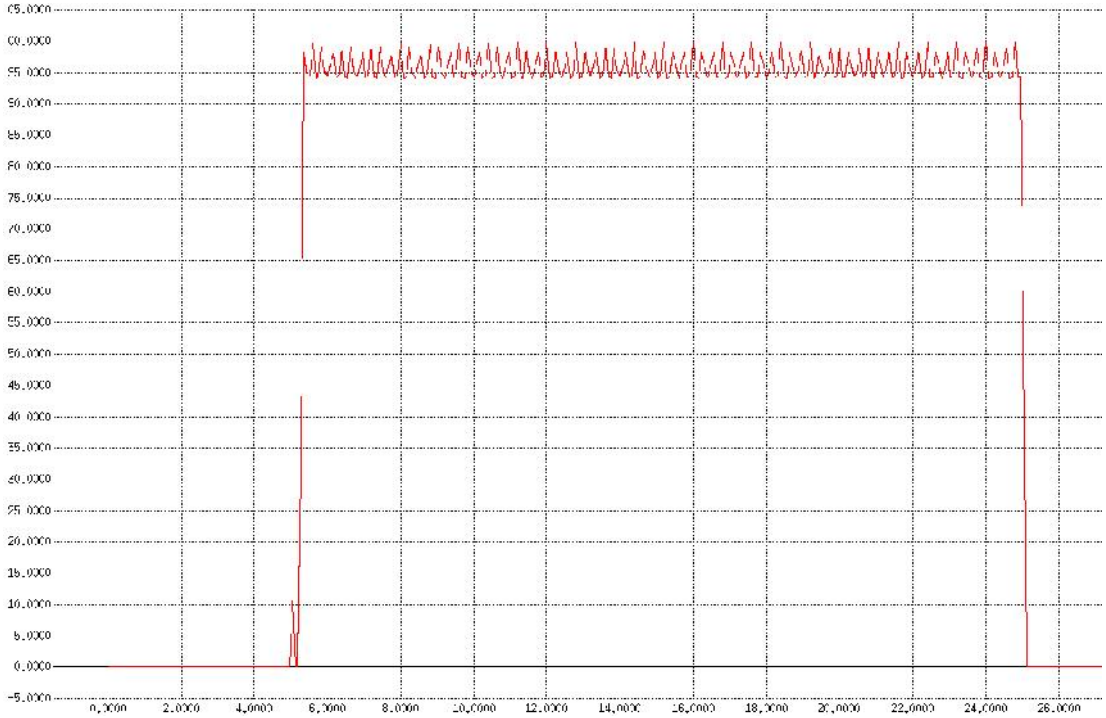

Рисунок 3.2 – Графік залежності завантаженості каналу від часу для одного з’єднання протоколом TCP Reno

Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP NewReno представлений на рисунку 3.3. У каналі з одним встановленим з’єднанням передача пакетів додатком FTP проходить в період часу з 5 до 25 с. На даній ділянці часової осі з’єднання TCP NewReno повністю займає канал, що обумовлюється незмінністю з моменту 5,2 с параметра часу передачі пакету. Даний факт вказує TCP NewReno, що в

мережі встановився стаціонарний режим, і збільшення розміру перевантажувального вікна призведе до перевантаження, а зменшення – знизить рівень завантаженості каналу.

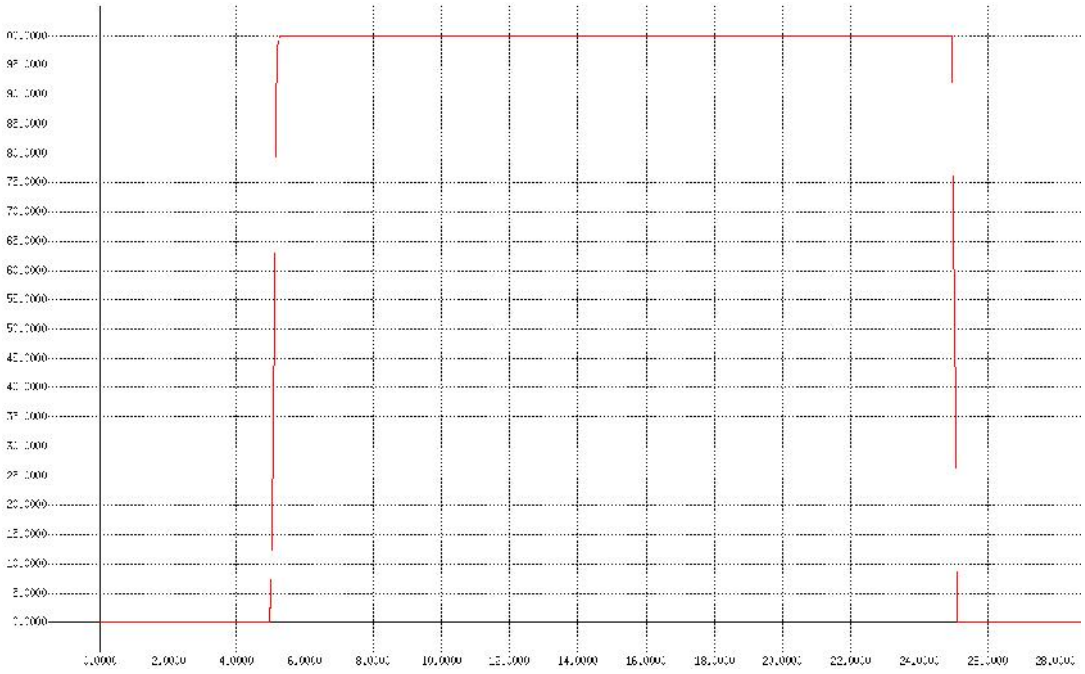

Рисунок 3.3 – Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP NewReno

На рисунку 3.4 представлений графік залежності завантаженості каналу від часу для з’єднання протоколом TCP Westwood. Алгоритми управління перевантаженнями даної реалізації протоколу TCP засновані на оцінці потоку даних і значенні поточного розміру смуги пропускання. Як випливає з графіка, значення кривої завантаженості каналу в період активності джерела трафіку коливаються від 92 до 93 Мбіт, що в середньому становить 92,5% пропускної здатності каналу.

Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP Hybla представлений на рисунку 3.5. У період активності джерела трафіку завантаженість каналу приймає значення від 90 до 100 Мбіт, тобто завантаження каналу становить 90-100%.

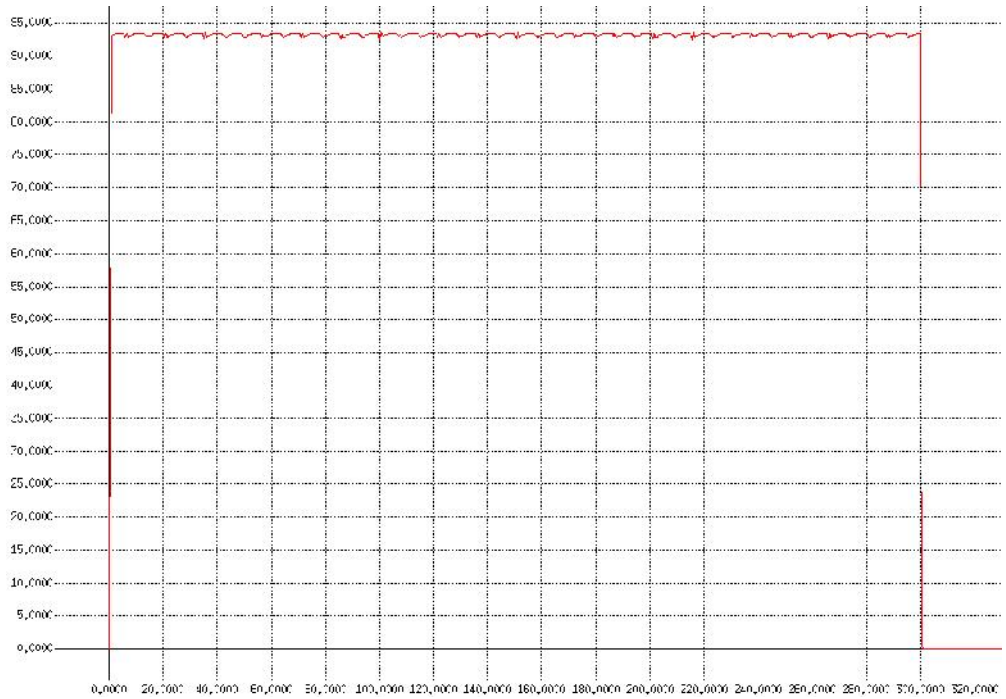

Рисунок 3.4 – Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP Westwood

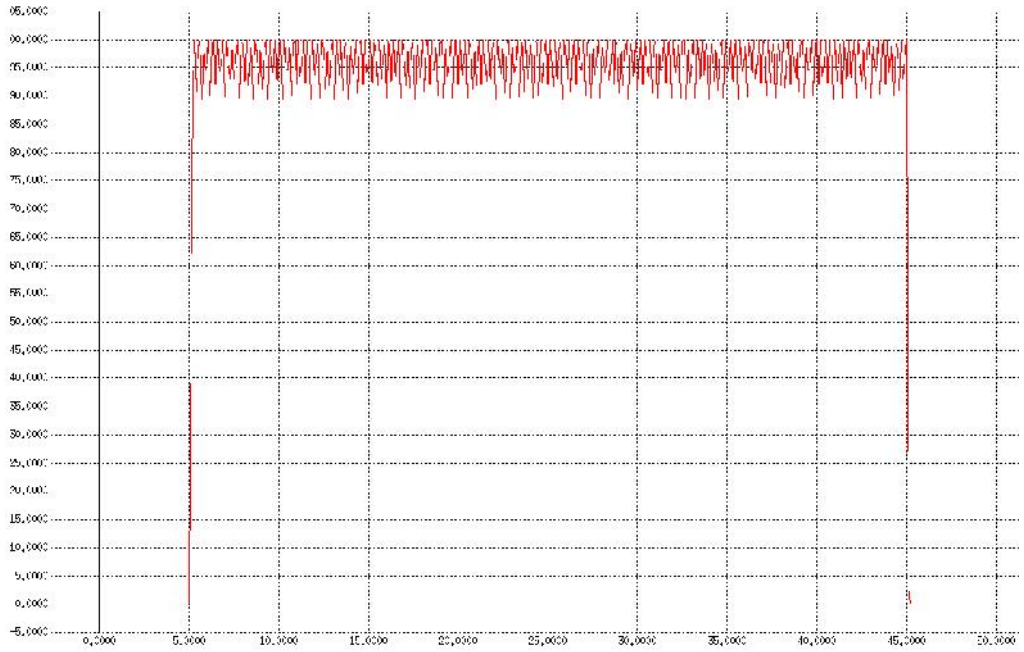

Рисунок 3.5 – Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP Hybla

Отримані результати дозволяють зробити висновок про те, що найбільш ефективно смугу пропускання у вільному каналі використовує реалізація протоколу TCP NewReno. Меншу пропускну здатність показують реалізації Reno, Westwood і Hybla – 90-100%.

3.4.2 Моделі з декількома з’єднаннями

Ряд наступних результатів представляє моделювання спільного функціонування декількох з’єднань протоколом TCP для різних реалізацій протоколу. У даних експериментах в якості фонового можна розглядати трафік інших існуючих в каналі з’єднань.

На рисунку 3.6 зображений графік залежності завантаженості каналу від часу при наявності з’єднань TCP Reno і TCP NewReno. У момент часу 10 с починається передача пакетів застосунком FTP1 в з’єднанні TCP Reno. До початку періоду активності додатки FTP2 в з’єднанні TCP NewReno (20 с) можна спостерігати криву, характерну для реалізації TCP Reno при одиночному з’єднанні. У період функціонування двох застосунків (20-90 с) з’єднання TCP Reno займає близько 74 Мбіт, TCP NewReno – близько 26 Мбіт пропускної здатності, що становить 74% і 26% завантаженості каналу, відповідно. Після закінчення передачі даних застосунком FTP1, TCP NewReno повністю займає канал.

Співвідношення завантаження каналу при одночасній роботі залишається незмінним і при зміні порядку проходження запуску джерел трафіку в часі (рисунок 3.7).

Графік залежності завантаженості каналу від часу для каналу при наявності трьох з’єднань приведений на рисунку 3.8. У момент часу 5 с в з’єднанні TCP Hybla запускається застосунок FTP1, в 10 с – FTP2 в з’єднанні TCP Reno і в 15 с – FTP3 в з’єднанні TCP NewReno. Крива, зображена в періоді $[5, 10]$, відображає використання каналу з’єднанням TCP Hybla, характерне для даної реалізації у вільному каналі.

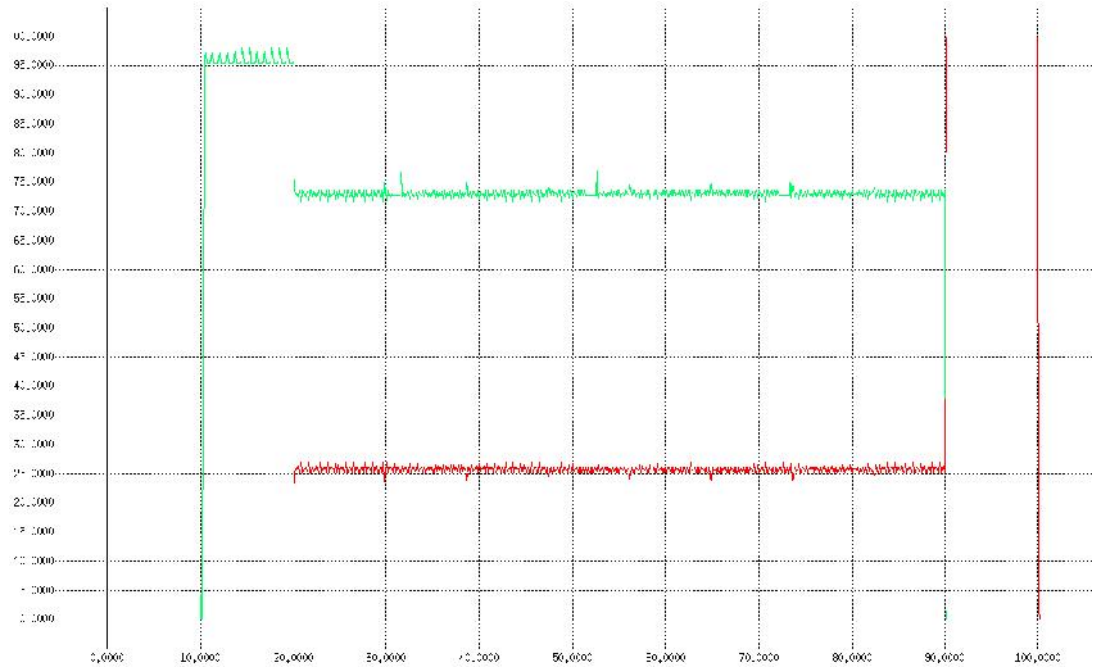

Рисунок 3.6 – Графік залежності завантаженості каналу від часу для з’єднань протоколами TCP Reno і TCP NewReno

Графік залежності завантаженості каналу від часу для каналу при наявності трьох з’єднань приведений на рисунку 3.8. У момент часу 5 с в з’єднанні TCP Hybla запускається застосунок FTP1, в 10 с – FTP2 в з’єднанні TCP Reno і в 15 с – FTP3 в з’єднанні TCP NewReno. Крива, зображена в періоді [5, 10], відображає використання каналу з’єднанням TCP Hybla, характерне для даної реалізації у вільному каналі.

Відрізок часової осі 10-15 с представляє завантаження каналу при активних з’єднаннях TCP Hybla і TCP Reno. Можна побачити, що значення завантаження смуги пропускання з боку TCP Reno знаходяться в діапазоні від 57 до 85 Мбіт, а TCP Hybla – від 10 до 40 Мбіт. Режим роботи трьох з’єднань зображений в періоді 15-40 с. У зазначений момент починає передачу застосунок FTP3 реалізації TCP NewReno. При цьому зменшується завантаження каналу з боку TCP Reno до 39-59 Мбіт, залишається незмінною середнє завантаження каналу з’єднанням TCP Hybla, трохи менше займає TCP NewReno (близько 24 Мбіт каналу).

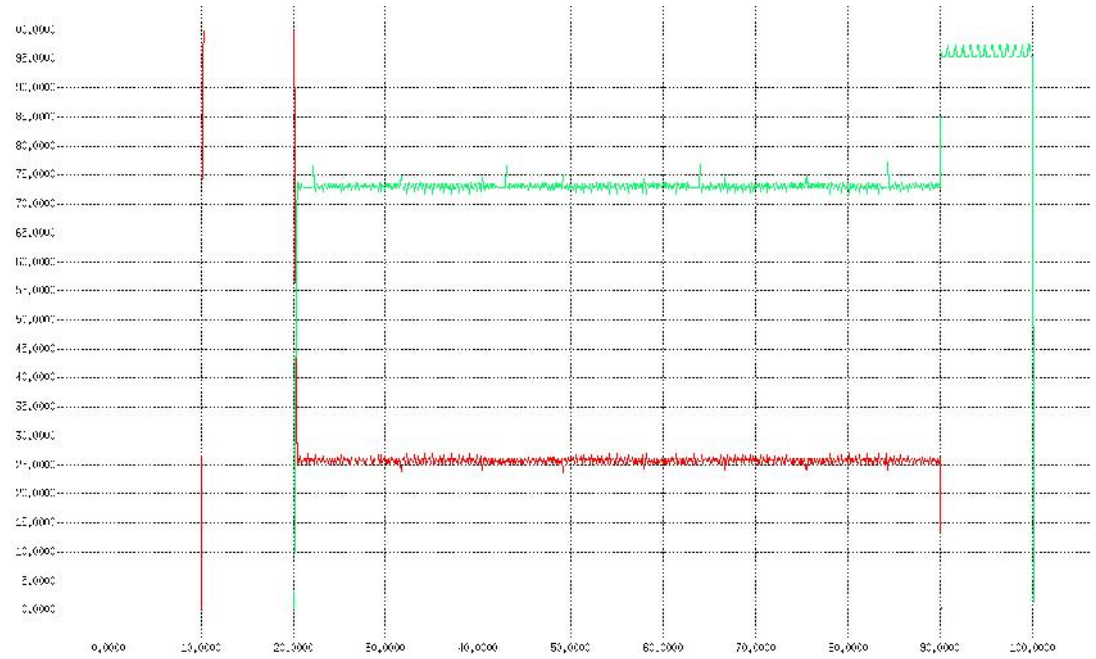

Рисунок 3.7 – Графік залежності завантаженості каналу від часу для з’єднань протоколами TCP NewReno і TCP Reno

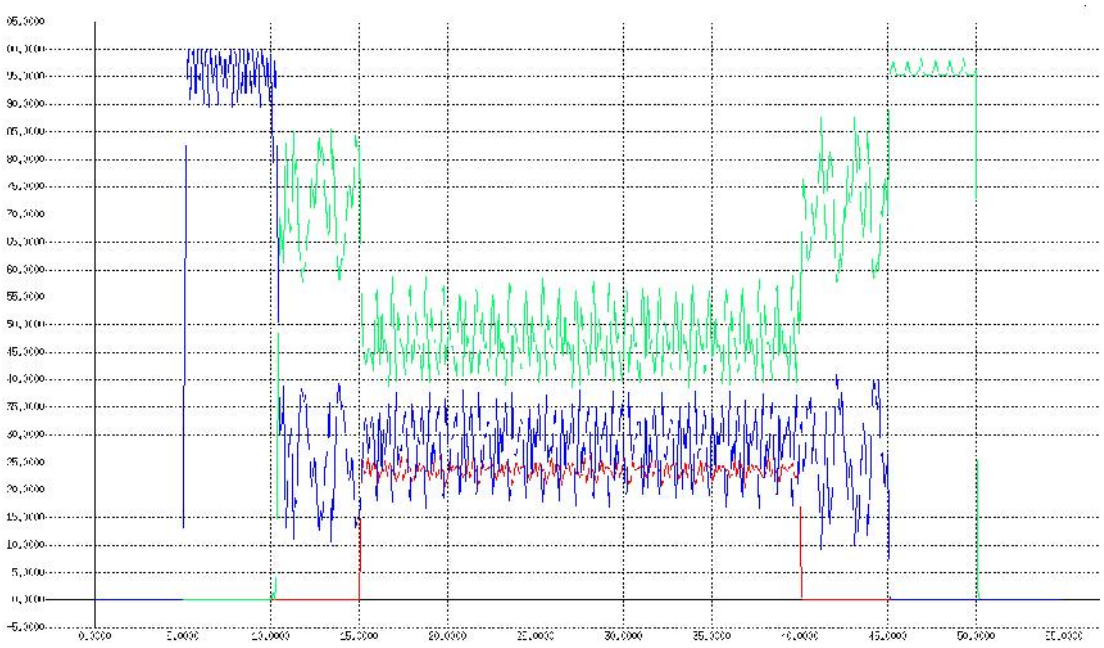

Рисунок 3.8 – Графік залежності завантаженості каналу від часу для з’єднань протоколами TCP Hybla, TCP Reno і TCP NewReno

Наступний сценарій (рисунок 3.9) моделює роботу з’єднань TCP

NewReno, TCP Westwood, TCP Reno. Криві в діапазоні 10-15 з відображають завантаження каналу з’єднаннями TCP NewReno і TCP Westwood із середніми значеннями 43 і 52 Мбіт відповідно.

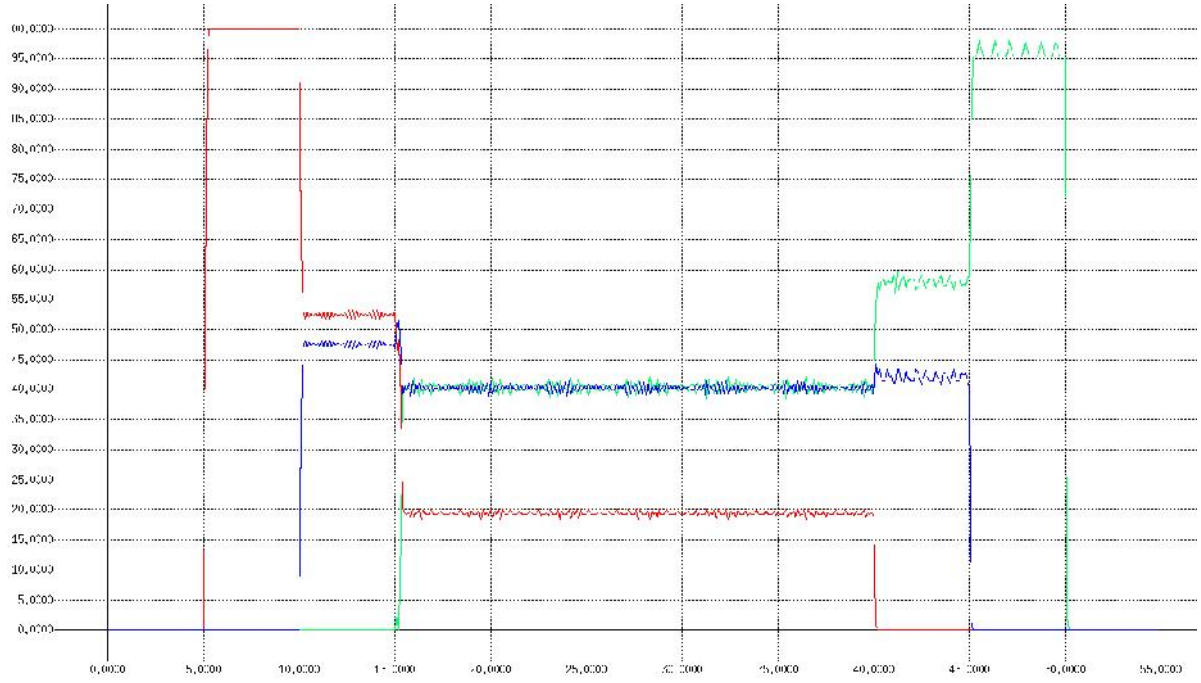

Рисунок 3.9 – Графік залежності завантаженості каналу від часу для з’єднань протоколами TCP NewReno, TCP Westwood і TCP Reno

Період активності всіх трьох з’єднань в даному сценарії знаходиться в діапазоні часу 15-40 с. При цьому завантаження смуг пропускання з боку TCP NewReno становить 19-20 Мбіт, а TCP Reno і TCP Westwood показують практично однакові криві завантаження з середнім значенням 40 Мбіт.

У період 40-45 з в каналі залишаються активними з’єднання TCP Reno і TCP Westwood. У зазначений діапазон часу можна спостерігати різке збільшення ступеня завантаження каналу з’єднанням TCP Reno (55-60 Мбіт) і середнє значення для з’єднання TCP Westwood, рівне 42 Мбіт.

Вище були наведені результати моделювання сценаріїв, які реалізують одночасну роботу декількох реалізацій протоколу TCP. Найбільшу ефективність показала реалізація TCP Reno, в сценаріях з двома з’єднаннями

займала 60% (TCP Westwood) і 75% (TCP NewReno, TCP Hybla) пропускної здатності каналу, з трьома з'єднаннями – 50% (TCP Hybla, TCP NewReno) і 40% (TCP NewReno, TCP Westwood).

TCP Westwood при наявності в каналі двох з'єднань TCP Reno і TCP NewReno, а також при одному з'єднанні TCP Reno демонстрував завантаження каналу на 40% пропускної здатності. З одним потоком TCP NewReno результат склав 48%.

Реалізації TCP NewReno і TCP Hybla у всіх сценаріях використовували канал на 20-25%. Виняток становить взаємодія з'єднань TCP NewReno і TCP Westwood.

3.4.3 Моделі зі з'єднанням TCP і одним UDP потоком

У даних експериментах проводилося моделювання одного або декількох TCP з'єднань в каналі при наявності UDP потоку з постійною бітовою швидкістю (CBR), виконуючого роль фонового трафіку.

Взаємодія з'єднання протоколом TCP Reno с UDP потоком представлено на рисунку 3.10. У момент часу 10 с починає передачу застосунок FTP в з'єднанні TCP Reno, в 20 с – джерело трафіку CBR з бітовою швидкістю 10 Мбіт/с. У період 10-20 с можна спостерігати «пилкоподібну» криву, характерну для пропускної здатності одного з'єднання TCP Reno. У період активності джерела трафіку завантаження каналу з'єднанням TCP Reno стрімко падає і місцями наближається до нуля. Середні значення знаходяться в діапазоні від 45 до 20 Мбіт, що становить 45-20% загальної смуги пропускання і 56-25% вільної.

На рисунку 3.11 представлений графік завантаження каналу з'єднанням TCP NewReno і потоком UDP. Додаток FTP в з'єднанні TCP NewReno починає передачу даних в момент часу 10 с. До моменту початку передачі даних джерелом трафіку CBR (20 с) графік завантаження каналу з'єднанням TCP NewReno є прямою, характерною для одного з'єднання реалізації TCP

NewReno. Весь період активності джерела трафіку UDP потоку (20-55 с) значення завантаження смуги пропускання TCP NewReno становить 900 Мб, що є 100% результатом завантаження вільного каналу.

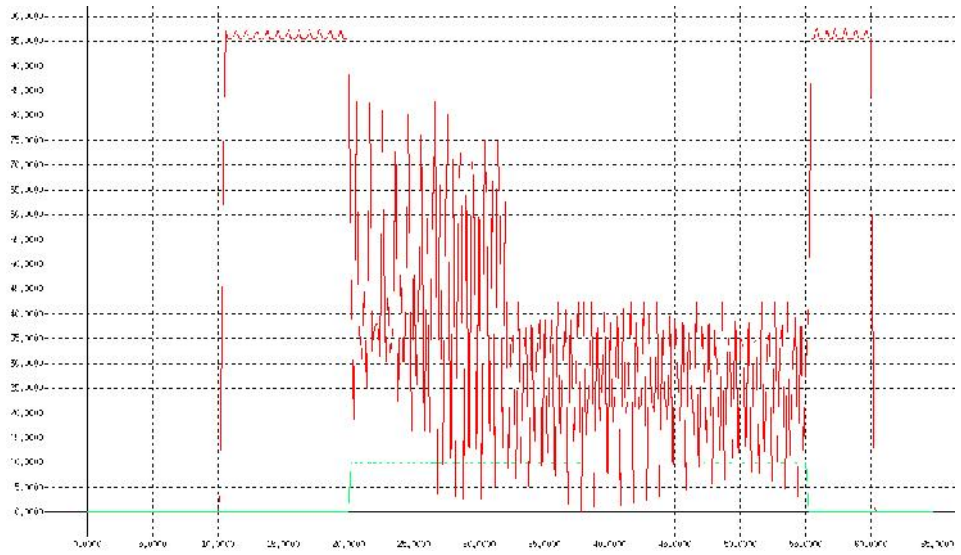

Рисунок 3.10 – Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP Reno і UDP

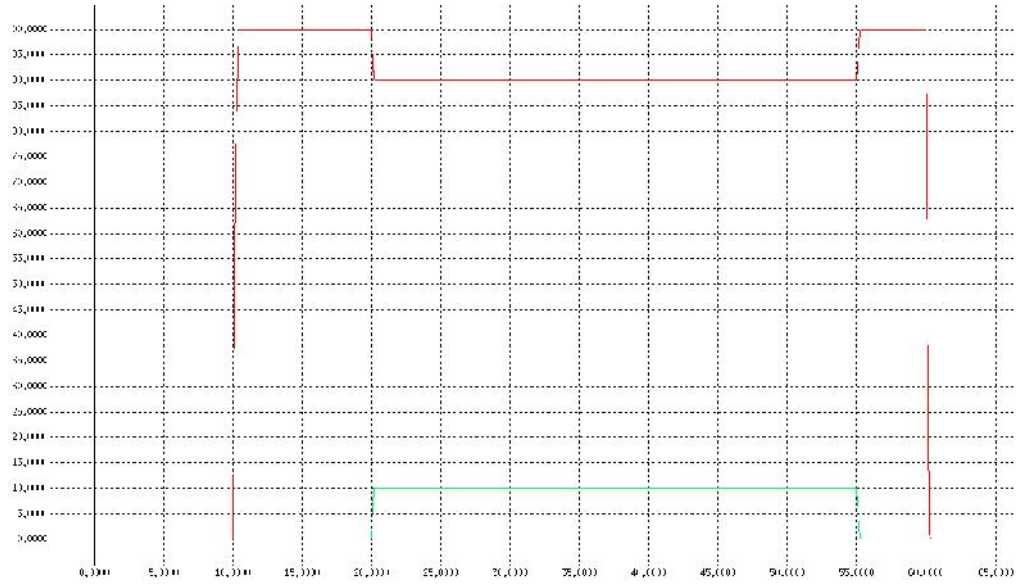

Рисунок 3.11 – Графік залежності завантаженості каналу від часу для з’єднання протоколом TCP NewReno і UDP

На рисунку 3.12 зображений графік використання смуги пропускання каналу з одним з'єднанням TCP Hybla і одним потоком UDP інтенсивністю 10 Мб/с. Середнє значення завантаження каналу з'єднанням TCP Hybla в період передачі даних джерелом трафіку – 800 Мб, що становить 88% вільної смуги пропускання каналу.

Рисунок 3.12 – График зависимости загруженности канала от времени для соединений протоколом TCP Hybla і UDP

Використання смуги пропускання каналу двома з'єднаннями TCP NewReno і TCP Reno, а також потоком UDP представлено на рисунку 3.13. У момент часу 5 с запускається застосунок FTP1 з'єднання TCP NewReno, в 10 с – застосунок FTP2 з'єднання TCP Reno, в 20 с – джерело трафіку потоку UDP. Період часу 10-20 с відображає характерний для з'єднань TCP NewReno і TCP Reno розподіл смуги пропускання. Однак в періоді передачі даних обома застосунками та джерелом трафіку, як і в випадку, описаному на рисунку 3.9, можна спостерігати різке зменшення пропускної здатності TCP Reno (середнє значення завантаження каналу становить 27 Мбіт) і збільшення TCP NewReno (середнє значення – 63 Мбіт).

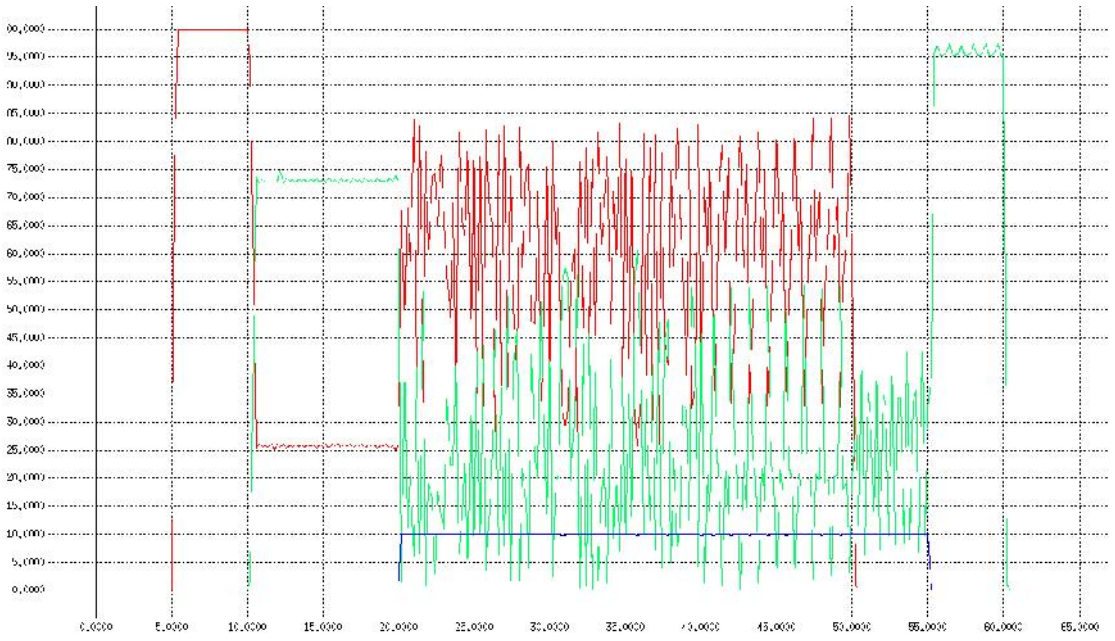

Рисунок 3.13 – Графік залежності завантаженості каналу від часу для з’єднань протоколами TCP NewReno, TCP Reno і UDP

Сценарії, результати моделювання яких наведені вище, є найбільш характерними для сучасних комп’ютерних мереж, особливістю яких є складна топологія і, як наслідок, різнорідність трафіку. В даний час широке поширення має програмне забезпечення, що використовує в якості протоколу транспортного рівня протокол UDP, що забезпечує більшу, ніж TCP, швидкість передачі даних. Складно уявити комп’ютерну мережу, в якій існують потоки одного транспортного протоколу або однієї його реалізації.

Найвищий ступінь завантаженості каналу при наявності в ньому фонового трафіку у вигляді потоку UDP і інших з’єднань TCP показала реалізація TCP NewReno, у випадку з одним UDP потоком завантаживши вільний канал на 100%.

Завантаження каналу з’єднанням TCP Reno, що значно перевищує цей параметр інших реалізацій протоколу при наявності в каналі тільки TCP з’єднань, різко зменшується з появою UDP потоків.

Вплив трафіку UDP на реалізації TCP Hybla і TCP Westwood є менш значним: пропускна здатність протоколів зменшується в середньому на 10%.

Далі проведено моделювання сценарію, аналогічного попереднім, але за винятком того, що замість протоколу TCP NewReno застосована запропонована модифікація методу.

На рисунку 3.14 наведені результати моделювання сценарію з двома з'єднаннями запропонованою модифікацією протоколу TCP NewReno. Спостерігається збільшення розміру перевантажувального вікна і пропорційність розподілу ресурсів каналу для запропонованої модифікації протоколу TCP NewReno.

Рисунок 3.15 відображає результати моделювання сценарію, що включає з'єднання запропонованої модифікацією протоколу TCP NewReno і наявність фонового трафіка.

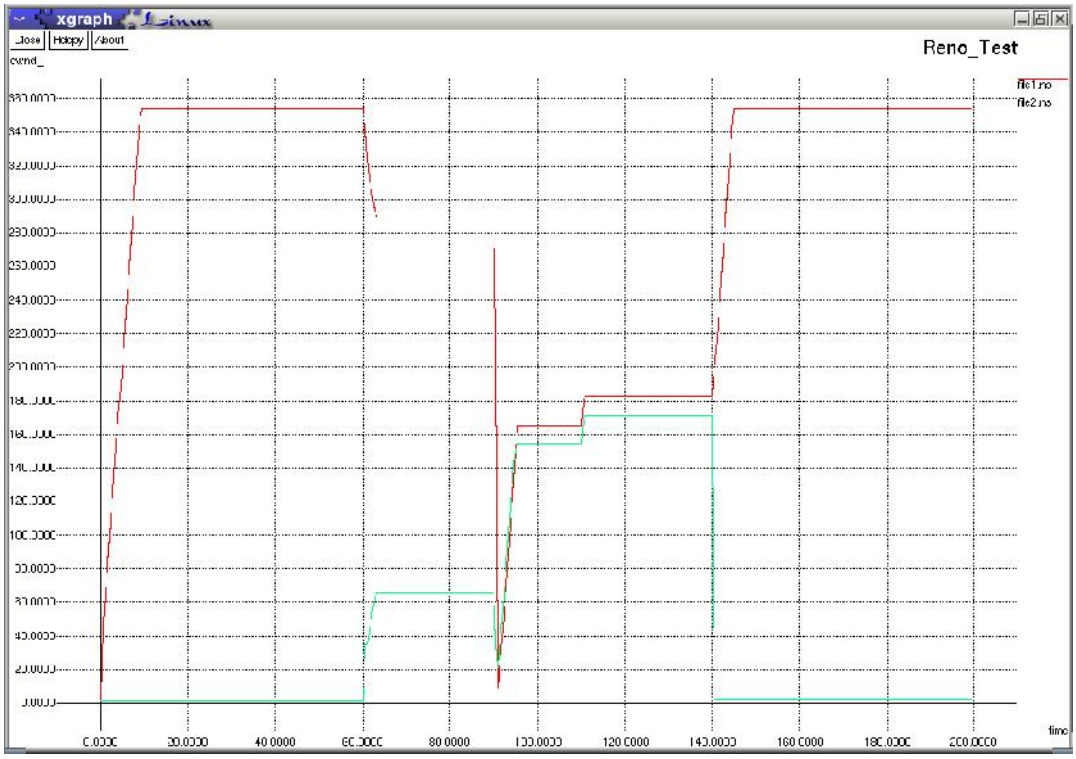

Рисунок 3.14 – Залежність пропускної здатності з'єднань від часу для сценарію, що включає одночасне існування сполук модифікованим методом, TCP Reno і наявність фонового трафіку

Рисунок 3.15 – Залежність розміру перевантажувального вікна з’єднань від часу для сценарію, що включає одночасне існування сполук модифікованим методом, TCP Reno і наявність фонового трафіку

3.5 Застосування реалізацій протоколу TCP при наявності фонового трафіку

Аналіз отриманих результатів моделювання дозволяє зробити наступні висновки. Все реалізації протоколу TCP прекрасно справляються з виникаючими в каналі перевантаженнями, завантажуючи смугу пропускання вільного каналу не менше 90%.

Реалізація TCP Reno демонструє досить «агресивну» поведінку при сценаріях роботи з більш пізніми реалізаціями протоколу. Результати моделювання показують нерівномірний розподіл смуги пропускання каналу, з’єднання TCP Reno займають її більшу частину. Даний факт пояснюється недосконалістю алгоритмів управління перевантаженнями, покладених в основу реалізації TCP Reno, що підтверджує моделювання сценаріїв з наявністю в каналі UDP трафіку з постійною бітовою швидкістю, де

пропускна здатність TCP Reno зменшується в три рази.

Алгоритми управління перевантаженнями більш пізніх протоколів адаптовані під сучасні комп'ютерні мережі, які характеризуються різнорідною природою протоколів транспортного і канального рівнів.

Серед них слід виділити реалізацію TCP NewReno, що демонструє майже ідеальну характеристику завантаження смуги пропускання, як в монопольному режимі, так і при постійній наявності в каналі UDP трафіку. Дисципліна управління розміром перевантажувального вікна, заснована на моніторингу часу передачі пакету, забезпечує рівномірність розподілу ресурсів мережі з невеликими затримками часу передачі пакета.

Рівномірність розподілу смуги пропускання демонструють і реалізації TCP Westwood і TCP Hybla, алгоритми управління перевантаженнями яких засновані на аналізі потоку даних і поточному значенні смуги пропускання. Дані реалізації відносяться до класу високошвидкісних протоколів TCP і занижені показники в даній роботі пояснюються певною топологією мережі. Високошвидкісні протоколи поступово починають впроваджувати в операційні системи. Так, в використовуваних сценаріях моделювання алгоритми управління перевантаженнями TCP Westwood і TCP Hybla були імпортовані з ядра Linux.

Результати моделювання запропонованої модифікації методу та відповідної реалізації протоколу TCP NewReno показали її ефективність як в разі однорідного, так і різнорідного (яка передбачає наявність фонового трафіку) мережевих середовищ. Її застосування дозволяє збільшити пропускну здатність протоколу і поліпшити рівномірність розподілу доступних з'єднанню ресурсів каналу.

ВИСНОВКИ

Таким чином, в ході виконання атестаційної роботи було проведено огляд функціонування сучасних програмних засобів протоколу TCP. Проаналізовано їх методи управління перевантаженнями, взаємодія між собою, іншими протоколами стека TCP/IP. Запропоновано метод збільшення пропускної здатності, що дозволяє підвищити пропускну здатність протоколу при наявності фонового трафіку. Перш за все, запропонований метод може використовуватися для реалізації у вигляді програмних засобів протоколу TCP NewReno.

Також розглянуто існуючі програмні засоби для вивчення ефективності мережевих протоколів і зроблений вибір імітаційної системи Ns-2, за допомогою якої була промодельована робота отриманих теоретичних результатів. Зроблено висновки про можливість застосування сучасних реалізацій протоколу TCP.

Згідно з результатами моделювання в гетерогенної мережевому середовищі було встановлено, що запропонований метод:

- дозволяє протоколу TCP досягати більш пропорційного розподілу доступної смуги пропускання каналу з іншими потоками TCP;
- має більшу пропускну здатність (приблизно на 15%) у порівнянні з існуючими методами при наявності в каналі фонового трафіку.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Коваленко А.А., Рубан І.В., Помінчук А.В. Метод збільшення ефективності протоколів транспортного рівня сучасних 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. 79.

2. Станкевич А.А. Имитационная дисциплина обслуживания для систем с очередями. / А.Е.Кучерявый, А.А.Станкевич // Электросвязь. 2013. №8. С.24-26.

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. Коваленко А.А. Метод перерозподілу пропускної здатності бездротової комп’ютерної мережі / А.А. Коваленко, А.А. Ляшенко // Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Матеріали восьмої міжнародної НТК. – Полтава: ПНТУ; Баку: ВА ЗС АР; Харків: ДП «ХНДІ ТМ», Жиліна: УмЖ, 2018. – 26-27 квітня 2018. – С. 35.

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 c.

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.