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)

Модель та метод для управління передачею даних бездротового з’єднання в ОС Linux

(тема)

Виконав:
студент II курсу, групи СПм-18-3
Помінчук Ю.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 Юлії Вадимівні
(прізвище, ім’я, по батькові)

1. Тема роботи Модель та метод для управління передачею даних бездротового з’єднання в ОС Linux

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

Реалізації протоколів TCP: Reno, NewReno, Hybla

Стек протоколів TCP/IP

Топологія комп’ютерної мережі з бездротовою ділянкою

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної області і постановка завдань

2. Розробка моделей та метода управління передачею даних бездротового з’єднання, вибір інструментальних засобів

3. Моделювання запропонованих результатів для бездротового з’єднання в ОС Linux

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 алгоритмів управління передачею даних	30.03.20-06.04.20	
2	Розробка моделей та метода управління	07.04.20-04.05.20	
	передачею даних, вибір засобів моделювання		
3	Моделювання 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: 86 с., 20 рис., 1 дод., 14 джерел.

МЕРЕЖНИЙ ПРОТОКОЛ, АЛГОРИТМ, ПЕРЕВАНТАЖЕННЯ, БЕЗДРОТОВА МЕРЕЖА, ЕФЕКТИВНІСТЬ, З'ЄДНАННЯ, ЗАГОЛОВОК, ПРОДУКТИВНІСТЬ, МОДЕЛЮВАННЯ, ТОПОЛОГІЯ.

Метою атестаційної роботи є розробка моделі та методу для управління передачею даних бездротового з'єднання.

У ході виконання атестаційної роботи проведено огляд теоретичних аспектів функціонування транспортного протоколу TCP та його різноманітних реалізацій в бездротових мережах, виконано аналіз поведінки і характеристик сучасних мережних протоколів транспортного рівня в бездротових мережах при різних граничних умовах, розроблено модель та метод для управління передачею даних бездротового з'єднання в ОС Linux, виконано вибір інструментальних засобів оцінки ефективності мережних протоколів та проведено моделювання запропонованих результатів в залежності від різних параметрів бездротового каналу зв'язку. Розглянуто результати моделювання, дана оцінка пристосованості реалізацій протоколу TCP до поставлених задач.

ABSTRACT

Master’s thesis: 86 pages, 20 figures, 1 appendix, 14 sources.

NETWORK PROTOCOL, ALGORITHM, CONGESTION, WIRELESS NETWORK, EFFICIENCY, CONNECTION, HEADER, THROUGHPUT, SIMULATION, TOPOLOGY.

The major goal of this thesis is to develop a model and method for wireless data transmission control.

During preparation of the thesis, theoretical aspects of TCP protocol operation, including its various implementations, in wireless networks were analyzed, the behavior and characteristics of modern transport protocols of transport layer in wireless networks under different boundary conditions were analyzed, a model and method for wireless data communication control in Linux OS were proposed, the choice of tools for assessing the effectiveness of network protocols was performed and simulations of proposed results depending on different wireless settings were conducted. The simulation results are discussed and estimation of TCP protocol implementations applicatibility is given.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACK – підтвердження отримання даних (англ., acknowledgment)

CA – запобігання перевантаження (англ., congestion avoidance)

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

CWND – ковзаюче вікно (англ., congestion window)

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)

SS – повільний старт (англ., slow start)

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

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

ВСТУП

У сучасному суспільстві все більш і більш значну роль відіграють комп’ютерні та інформаційні технології, по праву займаючи чільне місце серед інших. Еволюціонувавши від перших примітивних електронно-обчислювальних машин, комп’ютери продовжують удосконалюватися, надаючи все більш різноманітні методи роботи з інформацією. Але є очевидним фактом те, що комп’ютер, як і людина, є обмеженим у своїх можливостях. Розв’язанням такої проблеми стала концепція обчислювальних мереж, які дозволяють організувати взаємодію між певною множиною комп’ютерів, що відкриває нові перспективи, як в сфері організації обчислювального процесу, так і в багатьох інших сферах життя.

Проектування і побудова великих комп’ютерних мереж, що можуть успішно функціонувати і протидіяти різного роду атакам, є надзвичайно складним завданням. Для роботи глобальних і локальних комп’ютерних мереж необхідний певний набір правил, що регламентує процес взаємодії на різних рівнях. Такі правила називаються протоколами і на даний момент їх існує велика кількість, і всі вони виконують принципово різні функції. Протоколи об’єднуються в стеки, що містять в своєму складі набори протоколів, які реалізують більшість з усіх необхідних функцій з організації взаємодії між комп’ютерами.

В даний час домінуючим стеком мережевих протоколів є стек TCP/IP, що відповідає моделі організації взаємодії відкритих систем (OSI). Всі протоколи, що входять в стек TCP/IP, є стандартизованими міжнародною організацією ISO, і їх опис визначено в численних документах, що мають назву «request for comments» (RFC).

Транспортні протоколи грають найважливішу роль з точки зору управління трафіком в мережах на основі різних технологій. Згідно [1, 2], у сучасних комп’ютерних мережах близько 95% всіх байтів або 85-95% всіх

пакетів передаються протоколом управління передачею (TCP). Цей протокол є основним протоколом транспортного рівня у стеці TCP/IP. Головним завданням цього протоколу є забезпечення надійної передачі даних через ненадійне середовище передачі, одним з найбільш яскравих прикладів якого є бездротове середовище передачі.

У цій атестаційній роботі розглядаються питання, пов’язані з функціонуванням протоколу TCP, відповідного транспортному рівню моделі OSI. Як і для багатьох інших суб’єктів інформаційної діяльності, протокол TCP вже має багато реалізацій, і, в той же час, він піддається безперервним змінам, спрямованим на поліпшення його характеристик.

Важливим критерієм будь-якого протоколу є його ефективність, в яку входить множина складових, таких як продуктивність, надійність, простота застосування та інші. На сьогоднішній день існує ряд інструментальних засобів, що дозволяють зробити оцінку ефективності різних протоколів при різних умовах. Такі інструментальні засоби використовують різноманітні методи збору, обробки і виведення інформації про різні параметри та умови роботи протоколу і дозволяють в зручній формі провести їх аналіз.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПРЕДМЕТНОЇ ОБЛАСТІ І ПОСТАНОВКА ЗАВДАНЬ

1.1 Аналіз стека протоколів TCP/IP

Стек протоколів TCP/IP було розроблено з ініціативи Міністерства оборони США тому для взаємодії експериментальної мережі ARPAnet з іншими мережами у вигляді набору загальних протоколів для різнорідного обчислювального середовища [1, 2].

Великий внесок у розвиток стека TCP/IP, який отримав свою назву завдяки найбільш популярним протоколам TCP і IP, вніс Каліфорнійський університет в Берклі, реалізувавши протоколи стека у версії своєї операційної системи (ОС) UNIX. Популярність такої ОС привела до широкого поширення протоколів TCP, IP та інших протоколів стека. Сьогодні цей стек використовується для зв’язку комп’ютерів глобальної комп’ютерної мережі Internet, а також у величезному числі корпоративних мереж.

На нижньому рівні стек TCP/IP підтримує всі популярні стандарти фізичного і канального рівнів: для локальних мереж це Ethernet, Token Ring, FDDI; для глобальних – протоколи роботи на аналогових комутованих і виділених лініях SLIP, PPP, протоколи територіальних мереж X.25 та ISDN.

За роки використання в мережах різних країн і організацій, стек TCP/IP увібрав у себе велику кількість протоколів прикладного рівня. До них відносяться такі популярні протоколи, як протокол пересилки файлів FTP, протокол емуляції терміналу Telnet, поштовий протокол SMTP, використовуваний в електронній пошті мережі Internet, гіпертекстові сервіси служби WWW і багато інших. Сьогодні стек TCP/IP являє собою один з найпоширеніших стеків транспортних протоколів комп’ютерних мереж.

Стрімке зростання популярності комп’ютерних мереж, поряд з прогресом бездротових технологій, призвело до змін в розстановці сил у світі

комунікаційних протоколів – протоколи стеку TCP/IP стали швидко витісняти безперечного попереднього лідера – протоколи стеку IPX/SPX компанії Novell. Сьогодні в світі загальна кількість комп'ютерів, на яких встановлено стек TCP/IP, давно перевищила загальну кількість комп'ютерів, на яких працює стек IPX/SPX. Це говорить про різкий перелом відносно адміністраторів локальних комп'ютерних мереж до протоколів, що використовуються на настільних комп'ютерах, оскільки саме вони складають переважну кількість світового комп'ютерного парку і саме на них раніше майже скрізь працювали протоколи компанії Novell, необхідні для доступу до файлових серверів NetWare. Процес становлення стека TCP/IP як основного стека у всіх типах мереж продовжується, і зараз будь-яка мережна операційна система обов'язково включає програмну реалізацію цього стека.

Оскільки стек TCP/IP спочатку створювався для глобальної мережі Internet, він має багато особливостей, що дають йому перевагу над іншими протоколами, коли мова заходить про побудову мереж, що включають різні типи зв'язків: як провідні, так і бездротові. Зокрема, дуже корисною властивістю, що робить можливим його застосування в різнорідних мережах, є 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 (або кадру). У такому випадку при переході з однієї мережі, що має велику максимальну довжину кадру, в мережу з меншою максимальною довжиною, може виникнути необхідність розподілу кадру на частини. Протокол IP ефективно вирішує таку задачу.

Іншою особливістю стеку TCP/IP є гнучка система адресації, що дозволяє більш просто, в порівнянні з іншими протоколами аналогічного призначення, включати в об'єднану мережу інші мережі на основі принципово різних технологій. Така властивість також сприяє застосуванню стеку TCP/IP для побудови великих гетерогенних мереж, що обов'язково включають бездротові та супутникові канали зв'язку.

Платою за переваги виявляються високі вимоги до ресурсів і складність адміністрування IP-мереж. Гнучка система адресації і відмова від широкомовних розсилок призводять до наявності в IP-мережах різних централізованих служб (наприклад, DNS, DHCP та інших). Кожна з цих служб призначена для полегшення адміністрування мережі, включаючи конфігурування обладнання, але, в той же час, сама вимагає уваги з боку адміністраторів.

1.2 Протокол управління передачею даних TCP

Протокол IP є дейтаграмним протоколом і, внаслідок своєї природи, не може забезпечувати надійність передачі даних. Завдання забезпечення надійності обміну даними між прикладними процесами в складовій мережі вирішує протокол TCP, що працює безпосередньо над протоколом IP. Надійність передачі даних протоколом TCP досягається за рахунок того, що він заснований на встановленні логічних з’єднань між парами взаємодіючих процесів. До тих пір, поки програми протоколу TCP продовжують функціонувати коректно, а складена мережа не розпалася на незв’язні фрагменти, помилки передачі даних на мережевому рівні (IP) не впливатимуть на отримання коректних даних.

Протокол IP використовується протоколом TCP виключно як засіб транспорту. Перед відправкою своїх блоків даних, протокол TCP поміщає їх в оболонку IP-пакета. За необхідності, протокол IP здійснює будь-яку фрагментацію та зборку блоків даних TCP, що вимагаються для здійснення передачі та доставки через велику кількість проміжних мереж та шлюзів.

Протокол TCP забезпечує надійну доставку потоків і сервіс підтримки віртуальних з’єднань за рахунок використання підтверджень і повторної передачі пакетів при виникненні необхідності. Структура сегменту TCP показана на рисунку 1.1.

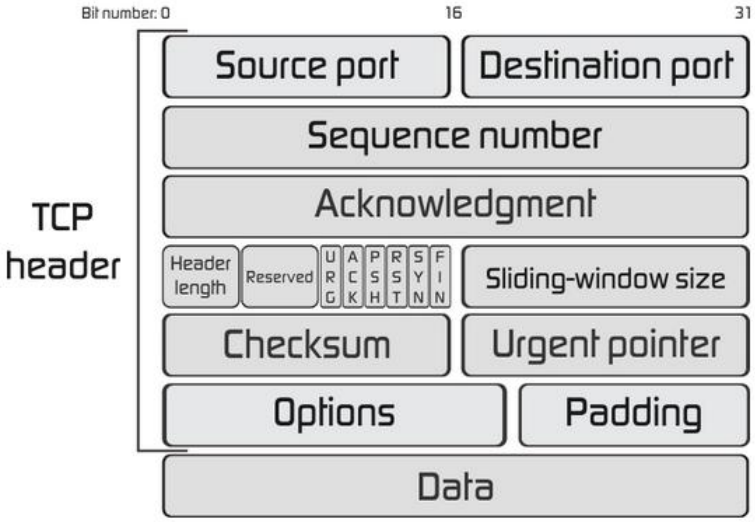

Рисунок 1.1 – Структура сегменту TCP

Опис полів сегменту, їх порядок, розміри і можливі значення наведені нижче. Необхідно зауважити, що в заголовку міститься основна інформація, яка забезпечує функціонування протоколу TCP, тому дана частина пакета є критичною.

Поля сегменту TCP [3]:

- порт відправника: номер порту, який є відправником;
- порт отримувача: номер порту, який є отримувачем;
- порядковий номер: порядковий номер першого октету в даному сегменті (за винятком присутності SYN); за наявності SYN поле порядкового номера містить початковий порядковий номер (Initial Sequence Number, ISN) і перший октет має номер ISN+1;
- номер підтвердження: якщо біт управління ACK є встановленим (значення «1»), то це поле містить значення наступного порядкового номера, який відправник сегмента очікує отримати; після організації з’єднання значення цього поля встановлюється в усіх випадках;
- зсув даних: 4-бітове поле, яке вказує число 32-бітових слів в заголовку; після заголовку розміщуються дані, тому поле вказує на початок даних в пакеті; заголовок пакетів завжди має розмір, кратний 32 бітам;

- резерв: зарезервоване поле розміром 6 біт;
- біти управління: 6-бітове поле, що містить флаги управління:
- U (URG) – значуще поле покажчика важливості;
- A (ACK) – значуще поле підтвердження;
- P (PSH) – функція push;
- R (RST) – скидання з’єднання;
- S (SYN) – синхронізація порядкових номерів;
- F (FIN) – немає даних від відправника;
- вікно: 16-бітове поле, що містить число октетів даних, які відправник даного сегменту буде сприймати, починаючи з октету, зазначеного в полі підтвердження;
- контрольна сума: 16-бітове значення контрольної суми, що обчислюється для 16-бітових слів заголовка і тексту; якщо сегмент містить непарне число октетів в заголовку або тексті, останні октети доповнюються праворуч 8 нулями для вирівнювання по 16-бітовому кордону; біти заповнення (0) не передаються в сегменті і служать тільки для розрахунку контрольної суми; при розрахунку контрольної суми значення самого поля контрольної суми приймається рівним 0;
- покажчик важливості: 16-бітове значення позитивного зсуву від порядкового номера в даному сегменті; це поле вказує порядковий номер октету, з якого починаються важливі (urgent) дані; поле береться до уваги тільки для пакетів зі встановленим флагом U;
- опції: опції передаються в кінці заголовку TCP і завжди мають розмір, кратний 8 бітам; при розрахунку контрольної суми пакета значення опцій беруться до уваги; опції можуть починатися на будь-якому кордоні октету; існують два формати опцій: одноктетні опції та багатоктетні опції, які містять поля типу опції (1 октет), її розміру (1 октет) і власне опцій;
- дані: поле даних TCP або протоколів верхніх рівнів.

1.3 Сучасні алгоритми управління передачею даних

Сучасні версії протоколу TCP (Tahoe, Reno, NewReno) мають проблеми продуктивності в бездротових з’єднаннях, які характеризуються відносно високою частотою помилок канального рівня і великим часом передачі пакету даних. Оскільки протокол TCP був розроблений переважно з метою управління перевантаженнями, втрати сегментів TCP через помилки в з’єднаннях помилково трактуються перевантаженнями, часто викликаючи непотрібну активацію механізмів запобігання перевантаження.

Ця проблема лише частково вирішується відповідними реалізаціями протоколу TCP. Необхідно підкреслити, що навіть в разі наявності каналу без втрат, у бездротових з’єднаннях все ж буде досить великий час передачі пакету даних (RTT). Це обумовлено алгоритмом управління передачею даних протоколу TCP, який заснований на механізмі плаваючого вікна і залежить від затримок в мережі.

При великому значенні RTT зменшується швидкість росту розміру вікна, що призводить до суттєвого зменшення продуктивності і нерівномірного використання доступного ресурсу смуги пропускання каналу усіма одночасно існуючими з’єднаннями (зазвичай бездротові ділянки маршруту характеризуються великими значеннями параметра RTT). У цьому випадку зменшення продуктивності є прямим наслідком поведінки протоколу TCP і не може бути виправлено без відповідних модифікацій алгоритму управління перевантаженнями. Для цієї мети може бути використана модифікація TCP Hybla.

В останні роки було запропоновано кілька рішень для збільшення продуктивності протоколу TCP для дротових і бездротових мереж. Деякі модифікації полягають у зміні стандартних процедур або внутрішніх параметрів протоколу TCP, а інші передбачають додавання альтернативних алгоритмів в TCP. Протоколи TCP Reno, TCP NewReno, TCP Hybla є спробами вирішення вищезазначених проблем.

1.4 Особливості бездротових з’єднань

Розглянемо найбільш істотні фактори, що впливають на пропускну здатність бездротових з’єднань та на ефективність стеку протоколів TCP/IP, і, відповідно, на пропускну здатність об’єднаної IP мережі на основі бездротового середовища передачі. Ставлячи перед собою мету аналізу протоколу TCP з позицій системного підходу, при якому робиться спроба врахування інших рівнів стеку, розташуємо ці фактори відповідно до гібридної моделі мережевих протоколів [2-3]:

- прикладний рівень: тип застосунків;
- транспортний рівень: протокол управління передачею даних (відновлення втрачених байтів, управління потоком), фрактальність трафіку;
- мережний рівень: помилки, викликані перевантаженням, протоколи маршрутизації, методи шифрування, фрактальність трафіку;
- рівень передачі даних: метод доступу до середовища передачі даних, ймовірність виникнення помилок при передачі, асиметрія радіоканалу;
- фізичний рівень: перешкоди, загасання, дифракція, віддзеркалення сигналу, мобільність вузла, помилки переадресації (хендовер).

Прикладний рівень стеку протоколу TCP/IP, виходячи з вимог до якості обслуговування відповідного застосунку, обумовлює частку пропускної здатності каналу, необхідну для цього додатка.

Транспортний рівень визначає протокол управління передачею, який має свої особливості, переважні області застосування і настроювані параметри, що дає можливість, за рахунок вибору реалізації протоколу та його налаштування, впливати на пропускну здатність [2-4]. Протокол транспортного рівня TCP гарантує достовірну доставку даних між вузлами і включає в себе механізми встановлення з’єднання, відновлення помилок і управління потоком в залежності від завантаження мережі [2-5]. Однак в бездротових і мобільних мережах пропускна здатність протоколу TCP істотно знижується внаслідок характерних особливостей бездротових мереж

(переважання канальних помилок, наявність помилок при переміщеннях та інше).

На рівні передачі даних впливають нижченаведені чинники [6-8].

Метод доступу до середовища. Одним з основних завдань, що вирішується при побудові бездротових мереж, є завдання розмежування доступу множини користувачів до обмеженого ресурсу середовища передачі. Існує кілька базових методів доступу, заснованих на просторовому, часовому, частотному, і кодовому поділі між вузлами.

Помилки при передачі даних обумовлюють зниження пропускної здатності і виникають при зниженні потужності і загасанні сигналу, як правило, при видаленні абонента від точки доступу. Наприклад, в технології IEEE 802.11n швидкість передачі в 300 Мбіт/с підтримується тільки в безпосередній близькості від точки доступу (в радіусі до 10 метрів), а при видаленні на значну відстань (100 метрів) знижується до 1-2 Мбіт/с .

Кількість вузлів і їх розміщення. Кількість точок доступу і характер їх фізичного розташування в зоні обслуговування впливають на пропускну здатність мережі. Якщо точка доступу є загальним засобом зв’язку і використовується в якості бездротового концентратора, то пропускна здатність каналу, що виділяється кожному з користувачів, знижується в міру збільшення кількості користувачів кожної окремої точки доступу [2-3].

Методи шифрування підвищують надмірність інформації, що призводить до збільшення часу, необхідного для обробки пакетів даних, що, в свою чергу, призводить до зниження пропускної здатності мережі.

Асиметрія радіоканалу. У бездротових мережах дані передаються в двох напрямках – від базової станції до мобільного вузла і навпаки. Ці два напрямки не є симетричними, оскільки при передачі даних від базової станції кількість бітових помилок менше і, отже, пропускна здатність вище, ніж у зворотному напрямку.

На фізичному рівні істотний вплив на пропускну здатність каналу мають наступні фактори.

Загасання сигналу і перешкоди. Загасання сигналу при подоланні природних перешкод призводить до зниження потужності і інтерференції сигналу, що, поряд з перешкодами, які можуть виникати при роботі сусідніх точок доступу на однакових або пересічних за частотами каналах, можуть стати причиною втрат пакетів даних і, відповідно, зменшення пропускної здатності.

Мобільність вузла: в процесі переміщення вузла параметри сигналу можуть змінюватися в значних межах, що негативно позначається на пропускної здатності усього з'єднання.

Характер трафіку. Фрактальний характер трафіку впливає на роботу мережевого рівня і рівня передачі даних, призводить до періодичного перевантаження комутаторів і маршрутизаторів. Наслідки фрактальності можуть бути частково компенсовані на транспортному рівні [2-5].

1.5 Постановка завдань атестаційної роботи

Бездротовий зв'язок є одним з основних перспективних напрямків розвитку сучасних високошвидкісних комп'ютерних мереж і забезпечує широкосмуговий інтегрований доступ до ресурсів різноманітних мереж. В якості транспортного протоколу широке застосування отримав протокол TCP, що створює надійний канал обміну даними між прикладними процесами в складовій мережі.

Хоча немає ніяких практичних обмежень продуктивності TCP, існує ряд важливих факторів, що впливають на ефективність застосування даного протоколу в бездротових каналах. Серед таких факторів можна виділити наступні: високий рівень помилок в бездротовому з'єднанні, ширина виділеної смуги частот і асиметрія з'єднань. Ряд проблем також пов'язаний зі значними змінами часу передачі пакету на встановленому маршруті, які є особливо великими в бездротових мережах. Флуктуації RTT ведуть до неправильних визначень величин тайм-аутів і помилкових повторних

передач пакетів даних, що в результаті знижує ефективність роботи протоколу. Також до проблем протоколу TCP відноситься специфіка поділу ресурсів каналу між з'єднаннями з різними значеннями RTT: з'єднанню з великим значенням RTT завжди дістається менша частина пропускної здатності фізичного каналу.

Вище було проаналізовано кілька сучасних реалізацій транспортного протоколу TCP. Можна помітити, що з кожною новою реалізацією, розробники прагнули позбутися від певного недоліку TCP, пов'язаного з функціонуванням протоколу при виникненні різного роду перевантажень. Всі розглянуті реалізації TCP справляються з ними по-різному, причому ступінь ефективності залежить від характеру наявних перевантажень пакетів.

У атестаційній роботі поставлено задачу більш докладного вивчення поведінки розглянутих реалізацій протоколу TCP операційної системи Linux при граничних умовах, а також розробку відповідної моделі та методу для управління передачею даних бездротового з'єднання в ОС Linux. Засобом реалізації отриманих результатів може бути модифікація протоколу TCP, що дозволяє підвищити його продуктивність в бездротових каналах.

Використання інструментальних засобів моделювання та моніторингу мереж дозволяє проводити оцінку різних характеристик протоколу при конкретних умовах його функціонування і різних параметрах технологій бездротового з'єднання, таких як пропускна здатність, затримка поширення сигналу, розмір черги пакетів в пристроях маршрутизації.

На підставі отриманих експериментальних результатів необхідно сформулювати рекомендації щодо подальшого удосконалення методів і вибору тієї чи іншої їх реалізації для конкретного роду задач.

2 РОЗРОБКА МОДЕЛЕЙ ТА МЕТОДА УПРАВЛІННЯ ПЕРЕДАЧЕЮ ДАНИХ БЕЗДРОТОВОГО З'ЄДНАННЯ, ВИБІР ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ

2.1 Розробка моделей протоколу TCP

2.1.1 Реалізація TCP Reno

Протокол TCP Reno містить дві фази зміни розміру вікна: фаза повільного старту і фаза запобігання перевантажень. При виявленні втрати пакету, відправка даних в канал сповільнюється; передача кожного пакету вважається успішною лише при отриманні пакету підтвердження. Отримання дублюючого підтвердження свідчить про те, що або сегменти досягли адресата не в первісному порядку, або сегмент було втрачено. Якщо надходить деяка кількість дубльованих підтверджень, то це означає, що вже пройшло чимало часу, і навіть якщо сегмент був відправлений по більш довгому маршруту, то він вже повинен досягти адресата. Існує велика ймовірність того, що його було втрачено.

В TCP Reno розмір вікна змінюється циклічно: він збільшується до тих пір, поки не відбудеться втрата сегмента. При отриманні відправником пакету підтвердження в момент часу $t+t_A$, поточне значення розміру вікна перевантаження $w(t)$ в наступний момент часу $w(t+t_A)$ обчислюється відповідно до наступного виразу:

$$w(t+t_A)=\begin{cases} \textit{фаза повільного старту:} \\ w(t)+1,\ \textit{якщо}\ w(t)<s(t); \\ \textit{фаза запобігання перевантажень} \\ w(t)+\dfrac{1}{w(t)},\ \textit{якщо}\ w(t)\geq s(t), \end{cases}$$

де $s(t)$ – порогове значення, при досягненні якого розміром вікна перевантаження TCP проводиться перехід із фази повільного старту в фазу запобігання перевантажень.

Якщо втрата пакета виявляється в результаті тайм-ауту, значення $w(t)$ і $s(t)$ обчислюються наступним чином:

$$w(t) = 1;$$

$$s(t) = \frac{w(t)}{2}.$$

Якщо втрата пакета виявляється в результаті інших подій, значення $w(t)$ і $s(t)$ змінюються інакше:

$$s(t) = \frac{w(t)}{2};$$

$$w(t) = s(t).$$

Після виявлення події втрати пакета і установки нових значень $w(t)$ і $s(t)$, протокол TCP Reno переходить в фазу швидкого відновлення. У фазі швидкого відновлення розмір вікна збільшується на один пакет при отриманні кожного пакету підтвердження, що являє собою лінійне зростання розміру вікна.

Швидка повторна передача. Протокол TCP генерує негайне підтвердження (пакет ACK, що дублюється) при отриманні пошкодженого сегмента, призначенням якого є оповіщення відправника про факт отримання пошкодженого сегмента, а також повідомлення номера наступної послідовності, що очікується.

Оскільки протокол TCP не знає, чи є дубльований пакет ACK ознакою

втраченого сегмента або наслідком повторного упорядкування сегментів, він очікує прийому невеликої кількості дубльованих пакетів ACK. Передбачається, що якщо відбулося лише повторне упорядкування сегментів, з'явиться один або два пакети ACK, що дублюються, до обробки заново упорядкованого сегмента, що згенерує черговий ACK. Якщо поспіль отримані три або більше однакові пакети ACK, то це є точним зазначенням втрати сегмента. Після цього протокол TCP виконує повторну передачу того, що виявилося відсутнім, без очікування закінчення таймера повторної передачі.

Алгоритми швидкої повторної передачі і швидкого відновлення зазвичай реалізовані спільно в такий спосіб:

- при отриманні третього поспіль пакету ACK, що дублюється, значення $s(t)$ встановлюється в половину поточного розміру вікна $w(t)$, але не менше ніж в два сегменти, і потім повторно передається відсутній сегмент; значення $w(t)$ встановлюється в значення $s(t)$ плюс три розміри сегмента, що збільшує перевантажувальний вікно на кількість сегментів, які залишили мережу і потрапили в чергу на іншому кінці з'єднання;

- при отриманні другого пакету ACK, що дублюється, значення розміру вікна $w(t)$ збільшується на розмір сегмента (для врахування сегмента, який покинув мережу) і потім передається пакет, якщо це можливо, вже при новому значенні $w(t)$;

- коли прибуває наступний пакет ACK, що підтверджує нові дані, значення $w(t)$ встановлюється в значення $s(t)$ (значення, встановлене спочатку); цей пакет ACK повинен бути підтвердженням повторної передачі з кроку 1 і, крім того, він повинен підтверджувати всі проміжні сегменти, відправлені після втраченого пакета і перед отриманням першого дублюючого ACK; цей крок необхідний для усунення можливої ситуації перевантаження, оскільки протокол TCP знижує швидкість передачі в два рази після втрати пакета.

Рисунок 2.1 представляє процес управління перевантаженнями в різних алгоритмах TCP.

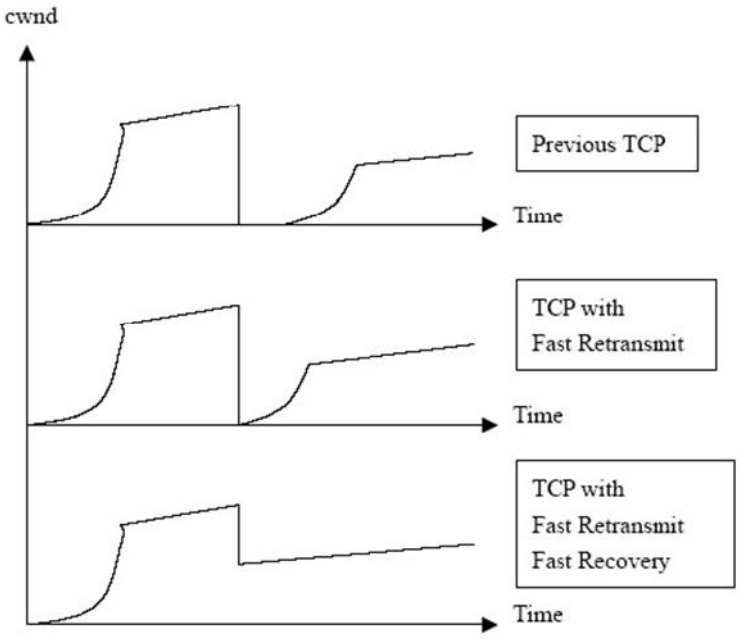

Рисунок 2.1 – Зміна розміру вікна перевантаження протоколу TCP від часу при різних схемах управління перевантаженнями

Протокол TCP Reno є вельми ефективним, поки втрати пакетів незначні. Однак коли відбуваються множинні втрати пакетів в межах одного вікна перевантаження, його продуктивність стає приблизно однаковою з Tahoe, внаслідок їх взаємної здатності виявлення виключно поодиноких втрат пакетів. Якщо присутні множинні втрати пакетів, то інформація про першу втрату пакета надходить за допомогою пакетів ACK, що дублюються. Інформація про другий втрачений пакет буде отримана тільки після досягнення пакетом ACK, для повторно переданого першого сегмента, відправника, після одного періоду часу RTT. Також можливо, що актуальний розмір вікна $w(t)$ двічі зменшиться при втраті пакетів в одному вікні.

Ще одна проблема полягає в тому, що якщо розмір вікна маленький в момент виникнення втрати, то достатню кількість дубльованих підтверджень для швидкої повторної передачі не буде отримано, і необхідно буде чекати тайм-аут. Отже, така схема не може бути ефективною для виявлення множинних втрат пакетів.

2.1.2 Реалізація TCP NewReno

В даний час найбільш популярною є модифікація TCP NewReno, що використовує алгоритм «Fast Retransmit & Fast Recovery» (швидка повторна передача і швидке відновлення) і реалізує опцію вибіркових підтверджень (SACK), при якій можливо підтверджувати одним пакетом значно більший обсяг відправлених даних. У разі, коли є доступною опція вибіркового підтвердження, відправник знає, які пакети слід переслати повторно, перебуваючи у фазі швидкого відновлення. При відсутності опції SACK немає достатньої інформації щодо пакетів, які потрібно надіслати повторно.

При отриманні трьох підтверджень, що дублюються, відправник вважає пакет втраченим і посилає його повторно. Після цього відправник може отримати додаткові підтвердження, що дублюються, оскільки одержувач здійснює підтвердження пакетів, які знаходяться в мережі, коли відправник перейшов в режим швидкої повторної передачі. У разі втрати декількох пакетів з одного вікна, відправник отримує нові дані, коли приходить підтвердження для повторно надісланих пакетів. Якщо втрачено один пакет і не відбулося змін їх порядку, то підтвердження цього пакета буде означати успішну доставку всіх попередніх пакетів до переходу в режим швидкої повторної передачі. Однак якщо втрачено кілька пакетів, то підтвердження повторно посланого пакету підтверджує доставку деяких пакетів (але не всіх), які були надіслані до переходу в режим швидкої повторної передачі.

Всі модифікації, внесені протоколом TCP NewReno в Reno, реалізуються під час пересилання і дозволяють усунути очікування протоколом TCP Reno закінчення таймера повторної передачі під час множинних втрат в рамках одного вікна. Дана зміна відноситься до поведінки відправника під час фази швидкого відновлення, коли надходять часткові ACK для пакетів, переданих на початку фази. Вони сприймаються як ознака того, що пакет слідував відразу за підтвердженим пакетом у

втраченій послідовності і повинен бути повторно переданий.

Таким чином, при множинних втратах пакетів з одного вікна, протокол TCP NewReno може відновлюватися без використання таймаута повторної передачі, повторно передаючи один втрачений пакет за один такт до тих пір, доки всі втрачені пакети з вікна не будуть повторно передані, залишаючись в фазі швидкого відновлення.

Розробники назвали алгоритм швидкого відновлення NewReno, оскільки він значно відрізняється від базового алгоритму Reno, описаного в RFC-2581. Запропонований алгоритм дає певні переваги в порівнянні з канонічним Reno при будь-яких сценаріях крім сценарію зі зміною порядку проходження пакетів.

2.1.3 Реалізація TCP Hybla

Метою протоколу TCP Hybla є поліпшення динаміки в гетерогенних бездротових мережах, головним чином за рахунок запобігання підключень, які мають високий час очікування. З рисунка 2.2 видно, що коли значення RTT з’єднань TCP зростає, коефіцієнт використання каналу ними знижується.

Головним чином це має відношення до часу, який потрібен вузлу для збільшення розміру вікна *w*. Коли з’єднання з великим часом передачі пакета даних конкурує зі з’єднанням з невеликим часом передачі в загальному каналі, то у останнього розмір вікна після колізій зростає набагато швидше, що призводить до отримання більшої частки пропускної здатності каналу. Загальна кількість пакетів, успішно доставлених вузлу-одержувачу, впливає на продуктивність. Навіть в разі, коли розміри вікон однакові, у з’єднань з більшим і меншим часом передачі пакета, ефективність першого є нижчою. Це відбувається через те, що пакети знаходяться в мережі тільки час RTT, отже, чим менше значення RTT, тим більше пакетів передається в одиницю часу, що майже завжди веде до кращої продуктивності протоколу.

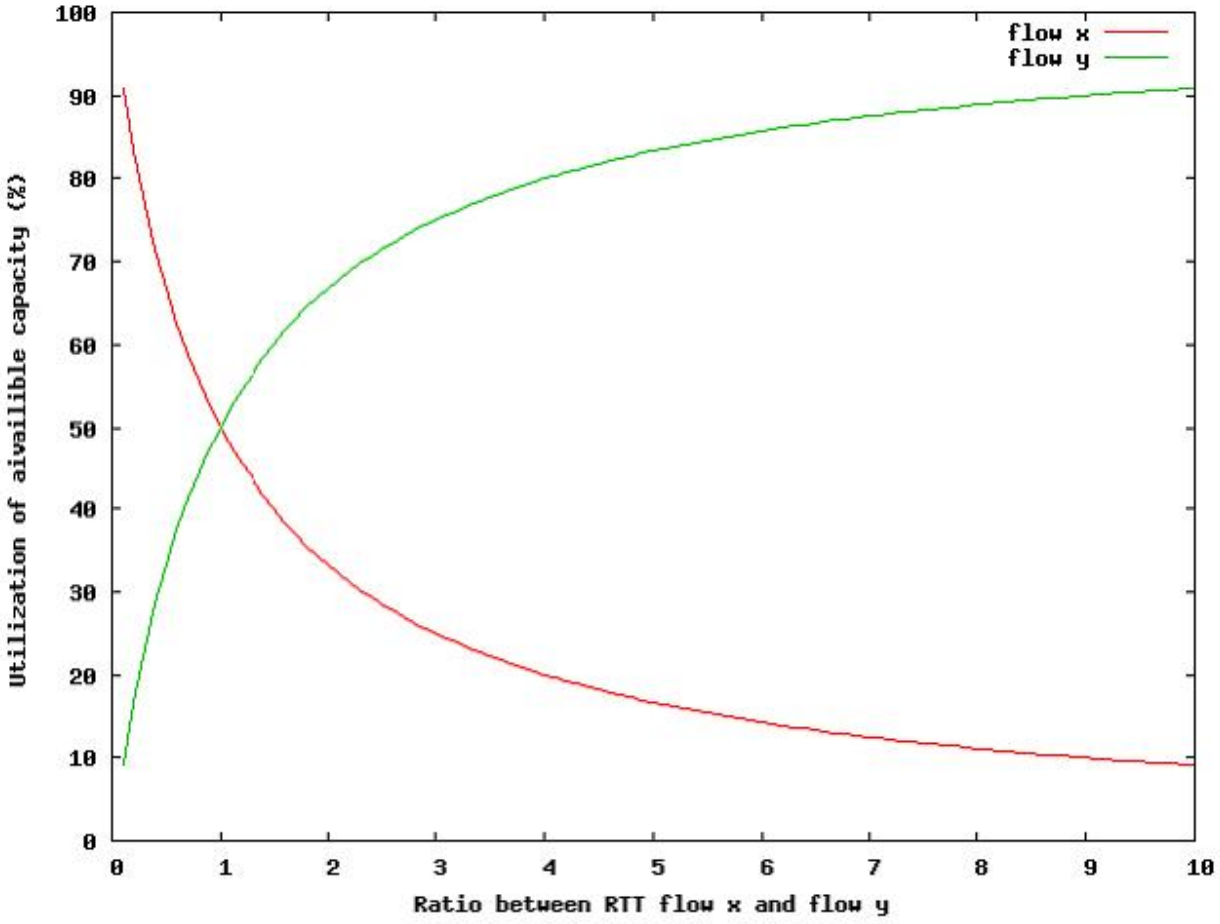

Рисунок 2.2 – Залежність коефіцієнта використання каналу від значення RTT

Протокол TCP Hybla повинен забезпечувати спосіб, при якому два канали з однаковою пропускною спроможністю, але різними значеннями RTT, мали б однакову продуктивність. Відповідно TCP Hybla, розмір перевантажувального вікна не повинен залежати від значення RTT. Це досягається використанням нормалізованого значення RTT (ρ), яке обчислюється діленням значення RTT на RTT_0:

$$\rho = \frac{RTT}{RTT_0},$$

де RTT_0 – це найменший виміряний час передачі пакета у встановленому з'єднанні.

Для узгодження зростання розміру вікна з табличним (базовим) значенням, потрібно множення на ρ правій частині виразу (2.1).

$$w(t) = \begin{cases} 2^{\frac{t}{RTT}}, 0 \le t < t_\gamma; \\ \dfrac{t - t_\gamma}{RTT} + \gamma, t \ge t_\gamma, \end{cases} \quad (2.1)$$

де $t_\gamma = RTT \cdot \log_2 \gamma$.

В отриманому виразі зростання розміру вікна залежить від значення RTT_0, надаючи можливість значенню *w* зростати так само швидко, як і в базовому з'єднанні. Вираз множиться на ρ, щоб компенсувати той факт, що розмір вікна змінюється в залежності від RTT, а не від RTT_0:

$$w(t) = \begin{cases} \rho 2^{\rho\frac{t}{RTT}}, 0 \le t < t_\gamma; \\ \rho\left[\dfrac{t - t_\gamma}{RTT} + \gamma\right] = \rho\left[\dfrac{t - t_\gamma}{RTT_0} + \gamma\right], t \ge t_\gamma, \end{cases} \quad (2.2)$$

Використовуючи вираз (2.2) можна обчислити значення *w* незалежно від RTT. У виразі (2.3) враховуються правила зміни розміру вікна перевантаження, використовуючи нормалізований розмір вікна перевантаження:

$$w_{i+1} = \begin{cases} w_i + 2^\rho - 1; \\ w_i + \dfrac{\rho^2}{w_i}. \end{cases} \quad (2.3)$$

Як і у випадку з нормалізованим TCP, розмір вікна може бути обчислений діленням величини *w*(*t*) на RTT: використовуючи вираз (1.2) отримаємо вираз (2.4), яке показує, що норма передачі не залежить від RTT. Таким чином, RTT_0 визначає норму передачі, яка є теоретичною нормою

передачі для протоколу TCP. [9]

Теоретично, такі вирази можуть застосовуватися тільки в ідеальних каналах без втрат. В інших ситуаціях вираження вірні до тих пір, поки в каналі не відбудеться перша втрата.

$$B(t) = \frac{w(t)}{RTT} \begin{cases} \dfrac{2^{\frac{t}{RTT_0}}}{RTT_0}, 0 \le t < t_\gamma; \\ \dfrac{1}{RTT_0}\left[\dfrac{t - t_\gamma}{RTT_0} + \gamma\right], t \ge t_\gamma, \end{cases} \tag{2.4}$$

де B(t) – швидкість відправки сегментів в мережу. Тоді

$$T_d(t) = \int_{x=0}^{x=t} B(\tau)d(\tau) \begin{cases} \dfrac{2^{\frac{t}{RTT_0}} - 1}{\ln(2)}, 0 \le t < t_\gamma; \\ \dfrac{\gamma - 1}{\ln(2)} + \dfrac{(t - t_\gamma)^2}{2 * RTT_0^{\,2}} + \dfrac{\gamma * (t - t_\gamma)}{RTT_0}, t \ge t_\gamma, \end{cases} \tag{2.5}$$

де $T_d(t)$ – кількість сегментів, відправлених з моменту встановлення з'єднання.

Таким чином, з виразу (2.5) виключено значення RTT.

Протокол TCP Hybla на сьогоднішній день є в більшій мірі теоретичним протоколом і не отримав широкого розповсюдження. До відомих недоліків можна віднести наступні:

- невирішена задача вибору оптимального значення RTT_0, що тягне за собою непропорційність розподілу доступної смуги пропускання каналу: з'єднання протоколом TCP Hybla може бути більш агресивним, що призводить до захоплення більшої частини доступного розміру смуги пропускання каналу, що особливо яскраво виражено в разі існування з'єднань з різними значеннями RTT [10-12];

- неоднозначність обчислення поточного значення параметру RTT:

якщо мережа відчуває перевантаження на певній ділянці, значення RTT збільшується, що веде до збільшення значення ρ, приводячи до більш агресивної поведінки з'єднання протоколом TCP Hybla, ще більше перевантажуючи мережу; завдяки великим розмірам вікон з'єднань протоколом TCP Hybla, з порівняно великими значеннями RTT, існує ймовірність того, що в мережі з'явиться перевантаження, коли з'єднання TCP Hybla посилає відразу все вікно після отримання пакету ACK, після цього настає тайм-аут до отримання наступного ACK; можливим вирішенням цієї проблеми може бути передача всього вікна даних за один час [13-14].

Версія протоколу TCP Hybla також доступна як модуль в системі NS-2, який дозволяє досліджувати її властивості і продуктивність.

2.2 Розробка методу управління передачею даних бездротового з'єднання

Суть запропонованого методу управління передачею даних бездротового з'єднання полягає в більш ефективному відновленні втрат в каналах з великим значенням часу передачі пакету (параметр RTT). Для цього в оригінальний метод протоколу TCP Hybla додана функція селективних підтверджень (SACK), яка полягає в тому, що адресат за допомогою пакетів ACK інформує відправника про всі коректно доставлені сегменти даних, дозволяючи, таким чином передавати повторно тільки втрачені пакети, причому в кількості більше одного пакету за один інтервал RTT. Така функція широко поширена в даний час, включена в більшість сучасних протоколів, і доступна в більшості реалізацій протоколу TCP.

Послідовність етапів пропонованого методу управління передачею даних бездротового з'єднання наведено на рисунку 2.3.

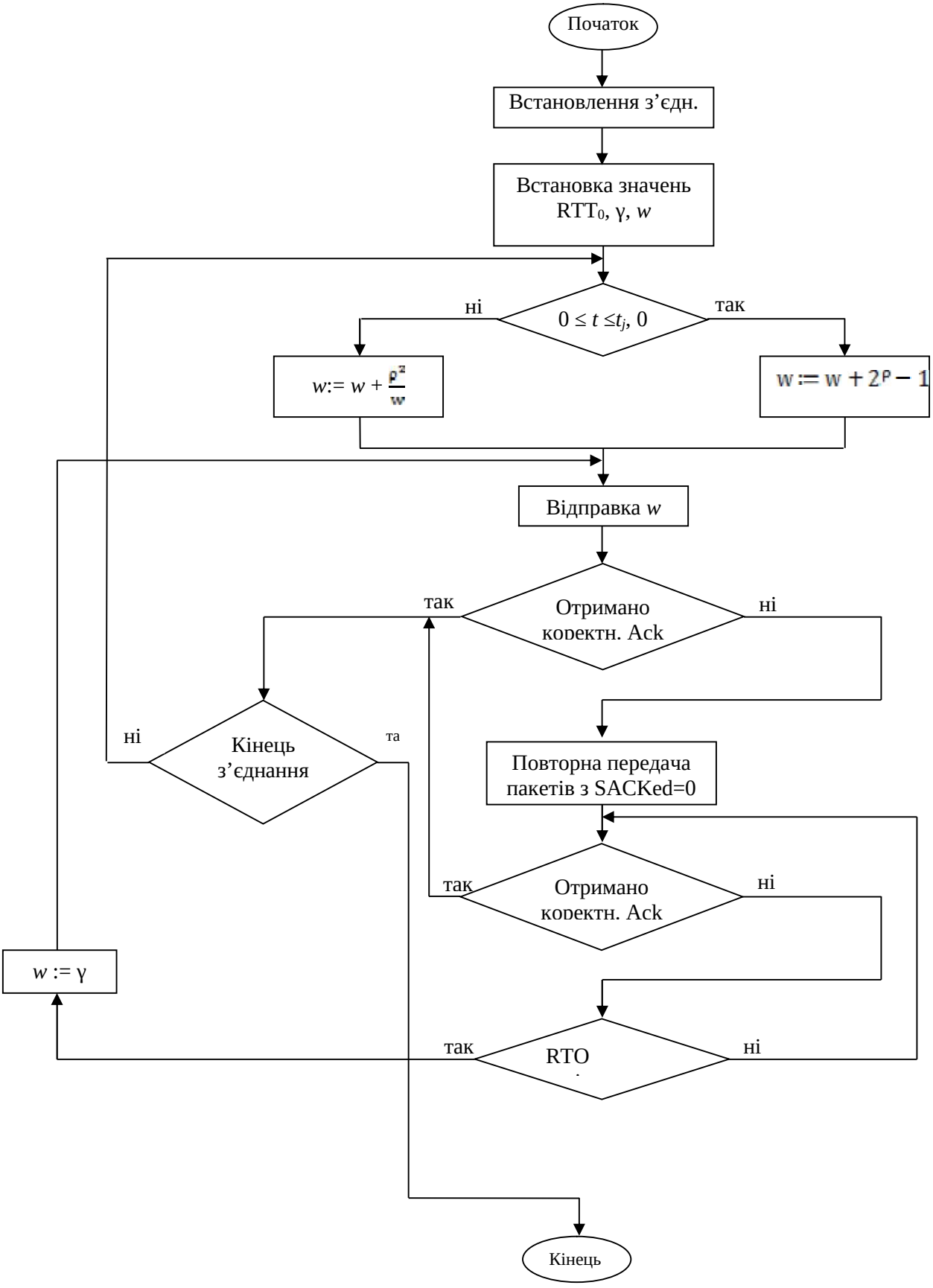

Рисунок 2.3 – Послідовність етапів методу управління передачею даних бездротового з’єднання

Одним з достоїнств запропонованого методу є простота його реалізації: зміни повинні вноситися тільки на стороні відправника, не зачіпаючи алгоритми одержувача.

Одержувачем відправляється опція SACK з метою оповіщення відправника про те, що було прийнято і буферизовано фрагментований блок даних. З цього моменту одержувач очікує отримання даних (наприклад, за допомогою механізму повторних передач, які відсутні в прийнятому фрагментированном блоці).

Після успішної повторної передачі всіх втрачених пакетів, одержувач підтверджує дані звичайним способом – зрушенням лівої межі вікна перегрузки в полі «Acknowledgement Number Field» заголовка сегмента TCP. Таким чином, опція SACK дозволяє не вносити ніяких змін в механізм підтверджень протоколу TCP Hybla.

Використання опції SACK вимагає дотримання наступних правил одержувачем:

- перший блок SACK повинен визначати суміжний блок даних, що містить сегмент, який ініціював такий пакет ACK (що дозволяє отримати актуальну інформацію про вільний розмір буфера одержувача);

- одержувач повинен включати в опцію SACK максимально можливе число окремих блоків SACK (хоча цього теж може бути недостатньо для оповіщення про всіх блоках, прийнятих у вхідний буфер одержувача);

- опція SACK повинна розширюватися за допомогою повторення останніх блоків SACK, які не є підмножинами блоків SACK, вже включених в опцію SACK (забезпечує нормальне функціонування, оскільки втрата будь-якого сегменту з нефрагментованого блоку даних в буфері одержувача може повідомлятися щонайменше трьома послідовними опціями SACK).

У свою чергу, джерело повинне зберігати в вихідному буфері відправлене, але ще непідтверджене, вікно даних. Для кожного сегмента в такій черзі існує новий флаг «SACKed», що підтверджує наявність опції SACK для нього. При отриманні сегменту підтвердження, що містить опцію

SACK, відправник встановлює флаги «SACKed» для всіх підтверджених бітів даних, і виключає такі сегменти при наступних повторних передачах. Будь-який з сегментів даних, для якого не встановлено флаг «SACKed», і номер якого менше найбільшого номера сегмента з встановленим флагом «SACKed», буде переданий відправником повторно.

Після тайм-ауту повторної передачі відправник обнуляє всі флаги «SACKed», оскільки такий тайм-аут може бути наслідком збою одержувача. У такому випадку відправник повинен передати повторно всі сегменти після закінчення тайм-ауту, починаючи з лівої межі вікна, незалежно від стану флагів «SACKed» сегментів даних. [1]

2.3 Вибір засобів моделювання

Математична модель апроксимує властивості і поведінку досліджуваної мережі, як наслідок, дозволяє вирішувати задачі з оптимізації і управління. Також на математичній моделі можлива апробація тих чи інших рішень, що незрівнянно дешевше, ніж на реальній системі, і виключає можливі помилки в ній.

Імітаційною є математична модель, реалізована як програмне забезпечення і використовує спеціальні або стандартні мови програмування. При побудові подібної моделі мережі можуть використовуватися як статичні, так і динамічні моделі. Під статичними розуміються моделі, що використовуються для дослідження стану мережі в задані моменти часу, а під динамічними – дискретні стохастичні моделі.

В даний час існує декілька програмних пакетів для моделювання процесів в комп’ютерних мережах на різних рівнях моделі OSI. У більшості випадків ці інструменти розроблені на платформах сімейства UNIX, але легко переносяться і на інші платформи, наприклад, Windows і Mac. Розглянемо найбільш відомий програмний пакет – NS-2.

2.3.1 Загальний опис

Система моделювання NS (Network Simulator) розроблена і розвивається в рамках проекту під назвою VINT спільно UC Berkeley, LBL, USC / ISI і Xerox PARC [5, 6]. Проект підтримується організацією DARPA (Defense Advanced Research Project Agency). Ця система написана на мові C ++ і поширюється з відкритим кодом. В якості керуючого інтерфейсу використовується мова «Object Tcl» (OTcl). Визначати і управляти моделлю можна як через інтерфейс OTcl, так і з використанням мови C++. У другій версії NS була проведена ревізія ієрархії класів, використовується об'єктна версія мови Tcl, а також внесено низку змін до структури модулів вихідних кодів.

Спочатку NS-2 розроблявся на платформі сімейства UNIX/Linux, однак може бути запущений і за допомогою ОС Windows. Крім того, робота NS-2 може бути організована під управлінням будь-якої підтримуваної платформи з використанням емуляторів POSIX-систем, наприклад RedHat CYGWIN або Microsoft Interix.

NS-2 – це керований подіями, об'єктно-орієнтований програмний пакет моделювання як локальних так і глобальних мереж [6-10]. Він містить опис різних мережевих протоколів (TCP, UDP), джерел трафіку (FTP, web, CBR, Exponential on/off), механізмів управління чергою (RED, DropTail), протоколів маршрутизації (Dijkstra) та іншого. NS-2 написаний на C++ і OTcl, щоб розділити реалізацію управління і потоку даних. Симулятор підтримує ієрархію класів в C++ (ієрархію, що компілюється) і відповідну ієрархію всередині інтерпретатора OTcl (ієрархію, що інтерпритується).

Причина, з якої NS-2 використовує дві мови, полягає в тому, що різні завдання мають різні вимоги. Наприклад, моделювання протоколів вимагає ефективного маніпулювання байтами і заголовками пакетів, роблячи швидкість виконання програми дуже важливою. З іншого боку, при дослідженнях мереж, коли важливо варіювання різних параметрів і швидка

перевірка множини сценаріїв, час, що витрачається на зміну моделі і повторний запуск, є дуже важливим параметром.

В системі NS-2 мова C++ використовується для детальної реалізації протоколів і для тих випадків, де потрібна безпосередня обробка кожного пакета з потоку. Наприклад, мова C++ використовується для реалізації нової дисципліни обслуговування черги. OTcl, в свою чергу, більше підходить для установки і настройки. Команди в OTcl виконуються досить повільно, але може бути дуже швидко змінені, дозволяючи легко вносити зміни в моделі. В NS-2 скомпільовані об’єкти C++ можна зробити доступними для інтерпретатора OTcl. В такому випадку, з рівня OTcl можна управляти вже створеними об’єктами C++.

2.3.2 Основи мови Object Tcl

У цьому підрозділі наводиться синтаксис і основні команди мови OTcl, що використовуються середовищем NS-2. Необхідно провести детальний аналіз OTcl, перед тим як розглядати питання щодо створення конкретного сценарію моделювання.

Присвоєння значень змінним. У Tcl, значення можуть бути збережені в змінних і в подальшому використані в наступних командах (приклад 2.1).

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

Приклад 2.1 – Присвоєння значень змінним

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

Процедури. Нові процедури можна визначати за допомогою команди «proc». Перший аргумент визначає ім’я процедури, другий містить список аргументів, які їй передаються. Наприклад, процедура, що обчислює суму двох чисел, могла б бути описана як показано прикладом 2.2.

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

Приклад 2.2 – Обчислення суми двох чисел

Наступна процедура обчислює факторіал деякого числа (приклад 2.3).

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

Приклад 2.3 – Обчислення факторіалу числа

Існує можливість передачі порожнього рядка в якості списку аргументів. Однак, в такому випадку змінні, використовувані процедурою, повинні бути визначені як глобальні (приклад 2.4).

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

Приклад 2.4 – Визначення глобальних змінних

Файли і списки. У Tcl файл може бути відкритий для читання за допомогою наступної команди (приклад 2.5).

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

Приклад 2.5 – Відкриття файлу для читання

Перший рядок файлу може бути збережена як список наступною командою (приклад 2.6).

```
gets $testfile list
```

Приклад 2.6 – Збереження рядку файла як список

Тепер можливо отримувати елементи списку за допомогою команд, де нумерація елементів починається з 0 (приклад 2.7).

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

Приклад 2.7 – Отримання елементів списку

Аналогічно, файл може бути записаний командою *puts* (приклад 2.8).

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

Приклад 2.8 – Записування файлу

Виклик підпроцесів. Команда «exec» створює підпроцес і очікує його виконання для завершення. Її використання схоже з передачею виконуваної рядки командному інтерпретатору (shell). Отже, для видалення файлу можна додати такий рядок, як наведено на прикладі 2.9.

```
exec rm $testfile
```

Приклад 2.9 – Створення підпроцесу

Команда «exec» зазвичай використовується, коли необхідно викликати tcl-скрипт з іншого tcl-скрипта. Наприклад, для запуску tcl-скрипта «example.tcl» багато разів із значенням параметра «test», що змінюється від 1 до 10, можливо використовувати рядки в іншому tcl-скрипті, що наведено на прикладі 2.10.

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

Приклад 2.10 – Багаторазовий запуск скрипта

2.3.3 Створення топології

Для запуску сценарію моделювання необхідно створити мережеву топологію. В NS-2 топологія представляється набором вузлів (nodes) і зв’язків між ними (links).

Перед тим як може бути встановлена топологія, необхідно створити новий об’єкт моделювання на початку скрипта, використовуючи команду, що наведено на прикладі 2.11.

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

Приклад 2.11 – Створення нового об’єкту моделювання

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

Нижче наведено приклад створення нових об’єктів вузлів (приклад 2.12).

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

Приклад 2.12 – Створення нових об’єктів вузлів

Метод класу «Simulator», що називається «node», створює чотири вузли і призначає їм описи, відповідно, «n0», «n1», «n2», і «n3». Ці описи в подальшому можуть бути використані для посилань на вузли. Якщо вузол є кінцевим (а не проміжним маршрутизатором), то, з метою моделювання, можуть бути задані агенти трафіку (TCP, UDP та інші) і джерела трафіку (FTP, CBR та інші). Джерела трафіку повинні бути послідовно прикріплені до агентів, а агенти – до вузлів.

Агенти, додатки і джерела трафіку. Найбільш часто використовуваними агентами в NS-2 є агенти TCP і UDP. У разі TCP, є кілька реалізацій такого протоколу.

Деякі типи агентів в NS-2:

- *Agent/TCP/Hybla* – відправник TCP Hybla;
- *Agent/TCP/Reno* – відправник TCP Reno;
- *Agent/TCP/Sack1* – відправник TCP з вибірковим підтвердженням;
- *Agent/TCP/Vegas* – відправник TCP Vegas.

Поширеними додатками і джерелами трафіку, наданими системою NS-2, є наступні:

- *Application/FTP* – генерує дані протоколом FTP для передачі транспортом TCP;

- *Application/Traffic/CBR* – генерує трафік, який має постійну бітову швидкість;

- *Application/Traffic/Exponential* – генерує ON/OFF трафік, в якому довжина періодів розподіляється експоненціально;

- *Application/Traffic/Trace* – трафік генерується на основі трейс-файлу, в якому визначаються розміри пакетів і час їх генерації.

Також існує можливість генерувати трафік, використовуючи методи, що надаються класом «Agent». Наприклад, якщо необхідно відправити дані протоколом UDP, може використовуватися метод на рівні tcl, за умови установки і призначення udp-агента вузла, який наведено на прикладі 2.13.

```
send (int nbytes)
```

Приклад 2.13 – Метод відправки даних протоколом UDP

Нижче наведено приклад створення джерела CBR-трафіку на основі протоколу UDP з прив’язкою до вузла «n0»:

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

Приклад 2.14 – Створення джерела CBR-трафіку на основі протоколу UDP

Додаток FTP, що використовує TCP в якості транспорту, може бути створено і прив’язано до вузла «n1» схожим чином, що наведено на прикладі 2.15.

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

Приклад 2.15 – Створення джерела FTP трафіку на основі протоколу TCP

Класи UDP і TCP є нащадками класу «Agent». Використовуючи «[new Agent/TCP]» і «[new Agent/UDP]», властивості цих класів можуть бути передані новим об’єктам «tcp1» і «udp0», відповідно, які, в свою чергу,

можуть бути прикріплені до вузлів «n0» і «n1». Потім визначається додаток і прив’язується до транспортного протоколу. Нарешті, встановлюються вихідні параметри джерела трафіку. У разі CBR, трафік може бути визначений параметрами «rate_» (або рівнозначним «interval_», визначальним проміжки між часом генерації пакетів), «packetSize_» (встановлює розмір пакета) і «random_». За допомогою параметра «random_» можливо додати певну випадковість під час появи пакетів. Значення за замовчуванням – 0, що означає, що випадковість не вводиться.

Одержувачі трафіку (traffic sinks). Якщо інформаційні потоки повинні завершуватися в вузлах без додаткової обробки, джерела UDP і TCP повинні бути з’єднані з одержувачами трафіку. Одержувач TCP визначений в класі «Agent/TCPSink», а UDP – в класі «Agent/Null».

Одержувач UDP може бути прив’язаний до вузла «n2» і з’єднаний з агентом «udp0» наступним чином, що наведено на прикладі 2.16.

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

Приклад 2.16 – Прив’язка одержувача UDP до вузла

Стандартний одержувач TCP трафіку, який генерує одне підтвердження (ACK) на один отриманий пакет, може бути прикріплений до вузла «n3» і з’єднаний з агентом «tcp1» наступним чином, ща наведено на прикладі 2.17.

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

Приклад 2.17 – Прив’язка одержувача TCP до вузла

Також є більш короткий шлях визначення зв’язку між відправником і

отримувачем (приклад 2.18).

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

Приклад 2.18 – Виклик одного скрипта із іншого

Наприклад, для створення з’єднання TCP між вузлами «n1» і «n3» з ідентифікатором класу пакета «1», можна використовувати метод, наведений на прикладі 2.19.

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

Приклад 2.19 – Створення з’єднання TCP між вузлами

Таким чином, можна легко створювати багато з’єднань TCP, використовуючи наведену команду всередині циклу.

Зв’язки. Для завершення топології необхідно створення зв’язків (links). В NS-2 вихідна черга (queue) вузла реалізована як частина зв’язку, тому, коли створюються зв’язки, користувач також повинен визначити тип черги.

На рисунку 2.4 наведено структуру симплексного зв’язку в NS-2. При створенні дуплексного зв’язку, спільно організовуються дві симплексних, по одній на кожен напрямок. Усередині каналу зв’язку пакет спочатку ставиться в чергу. Після цього він або скидається (передається агенту «Agent/Null» і звільняється), або вибирається з черги і передається об’єкту затримки «Delay», який моделює затримку в з’єднанні. Нарешті, обчислюється і оновлюється параметр TTL (time to live).

Зв’язки створюються способом, що наведено на прикладі 2.20.

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

Приклад 2.20 – Створення зв’язку

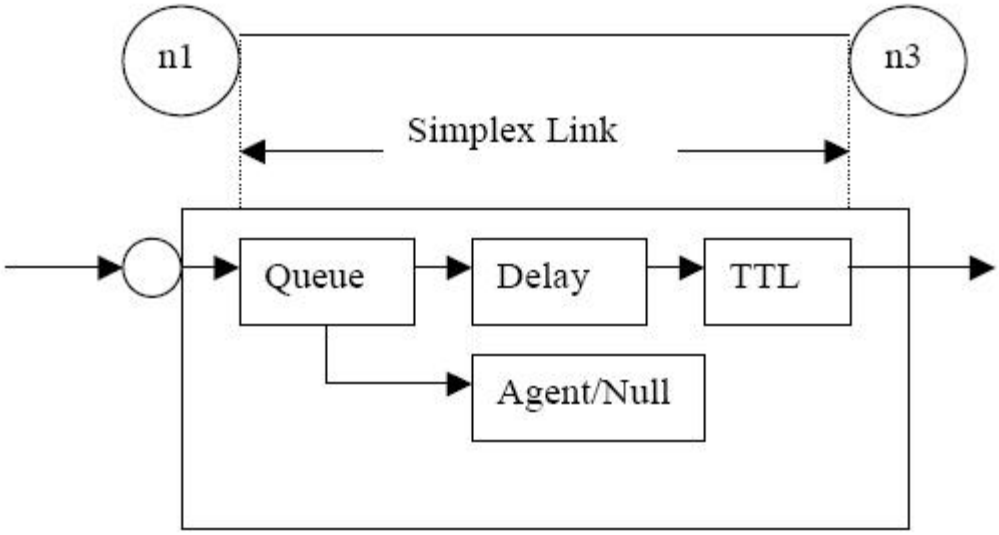

Рисунок 2.4 – Зв’язок в середовищі NS-2

Наприклад, щоб створити дуплексний зв’язок з механізмом управління чергою «DropTail» між вузлами «n0» і «n2», можна додати такий рядок, що наведено на прикладі 2.21.

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

Приклад 2.21 – Створення дуплексного зв’язку з механізмом управління чергою «DropTail»

Для створення симплексного зв’язку з механізмом управління чергою «RED» між вузлами «n1» і «n2», необхідно задати парметри у відповідності до прикладу 2.22.

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

Приклад 2.22 – Виклик одного скрипта із іншого

Значення смуги пропускання вказуються безрозмірним числом або, використовуючи приставки «k» (кіло), «M» (мега), «b» (біт) і «B» (байт).

Затримка теж може бути виражена схожим чином, використовуючи «m» (милі) і «u» (мікро) як приставки.

В системі NS-2 реалізовані різні алгоритми з управління чергою, але в даній роботі будуть використовуватися тільки «DropTail» і «RED».

2.3.4 Трасування і моніторинг

Для існування можливості аналізу результатів моделювання, дані повинні бути зібрані і представлені певним чином. NS-2 підтримує трейси і монітори.

Трейси дозволяють записувати пакети, при виникненні певних подій, таких як скидання пакета або постановка його в чергу каналу зв’язку.

Монітори надають методи збору чисельних значень, такі як кількість скидів пакетів або число прибулих пакетів в чергу. Монітор може бути використаний як при отриманні цих значень для всіх пакетів, так і тільки для певних потоків (потоковий монітор).

Всі події при моделюванні можуть бути записані в файл наступними командами у відповідності до прикладу 2.23.

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

Приклад 2.23 – Запис подій при моделюванні в файл

Спочатку вихідний файл відкривається для запису і йому призначається дескриптор. Потім зазначені події записуються в файл, визначений таким дескриптором, і, нарешті, при закінченні моделювання необхідно скинути буфер трасування і закрити файл. Всі такі операції зазвичай виконуються в окремій заключній процедурі.

Якщо в результаті використання цих команд створюються зв’язки, то в

них додаються додаткові об’єкти для трасування («EnqT», «DeqT», «DrpT» і «RecvT»), як показано на рисунку 2.5.

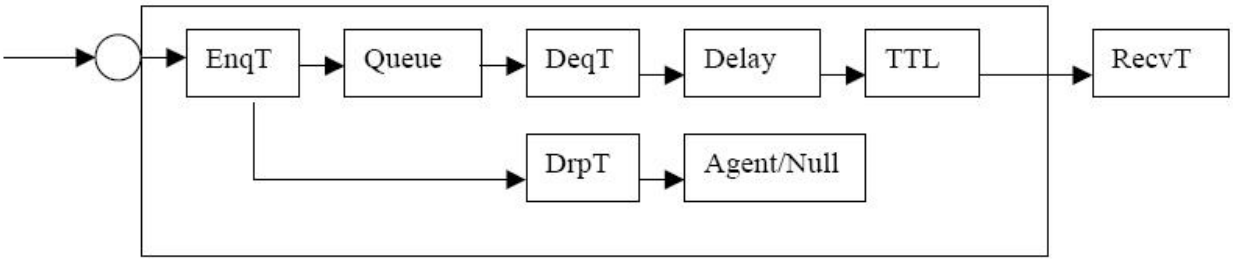

Рисунок 2.5 – Зв’язок в NS-2 при активованому трасуванні

Ці нові об’єкти потім записуються в трейс-файл кожного разу при отриманні пакету. Трейс-файлі мають формат, що наведено на прикладі 2.24.

```
+ 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
- : виборка із черги
d : скидання пакету
r : отримання пакету
```

Приклад 2.24 – Трейс-файл

Полями трейс-файлу є: тип події, модельний час виникнення події, ідентифікатори вузлів відправника та одержувача, тип пакету (протокол, дія або джерело трафіку), розмір пакету, флаги, ідентифікатор потоку, адреси відправника та одержувача, номер послідовності та ідентифікатор пакета.

На додаток до трасування всіх подій моделювання, також можливо створити об’єкт трейсу між конкретними відправником і отримувачем наступною командою, ща наведено на прикладі 2.25.

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

Приклад 2.25 – Виклик одного скрипта із іншого

Тип трейсу може бути наступним:

- enque: прибуття пакета (наприклад, в чергу);
- deque: відправка пакету (наприклад, з черги);
- drop: відкидання пакету;
- recv: отримання пакету адресатом.

Трасування всіх подій при моделюванні в певний файл з наступним розрахунком необхідних значень з цього файлу, використовуючи, наприклад, «perl» (або «awk») і пакет «Matlab», є досить простим способом, застосовним в разі, коли топологія мережі відносно проста і кількість джерел обмежена. При складній топології і великій кількості джерел цей спосіб збору інформації стає занадто повільним. Трейс-файли при цьому також займають значний обсяг місця на жорсткому диску.

Монітори (monitors). За допомогою моніторів черги (queue monitors) можна відстежувати статистику прибуття, відправлення і відкидання як пакетів, так і байтів. Додатково монітор черги здатний зберігати загальну довжину черги з плином модельного часу.

Для прикладу, якщо встановлено зв’язок між вузлами «n0» і «n1», монітор черги може бути створений таким чином, що наведено на прикладі 2.26.

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

Приклад 2.26 – Створення монітору черги

Прибуття пакетів і їх відкидання відстежується наступними командами, що наведено на прикладі 2.27.

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

Приклад 2.27 – Відстеження пакетів

Поряд з привласненням значення змінної, команда «set» може використовувати для отримання значення змінної. Тут, наприклад, команда «set» використовується, щоб отримати значення змінної «parrivals», що визначена в класі монітора черги.

Монітор потоку (flow monitor) схожий на монітор черги, але він відстежує статистику для конкретного потоку, а не всього трафіку. Спочатку класифікатор визначає, якому з потоків належить пакет, і тільки потім передає пакет монітору потоку.

Монітор потоку створюється і прив’язується до певної зв’язку наступним чином, що наведено на прикладі 2.28.

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

Приклад 2.28 – Створення і прив’язка монітору потоку

Оскільки такі команди відносяться до створення монітору потоку, то вони визначені в класі «Simulator», а не класі «Flowmonitor». Змінні і команди класу «Flowmonitor» можуть бути використані тільки після створення і прив’язки монітора до зв’язку. Наприклад, щоб отримати дамп вмісту монітору потоку (всіх потоків), використовується наступна команда, що наведена на прикладі 2.29.

```
$fmon dump
```

Приклад 2.29 – Отримання дампу вмісту монітору потоку

Якщо ж потрібно відстежити статистику певного потоку, необхідно

визначити класифікатор, щоб він вибирав потік, грунтуючись на його ідентифікаторі, який, наприклад, дорівнює «1» (приклад 2.30).

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

Приклад 2.30 – Відстеження статистики певного потоку

В даній роботі все дані, що відносяться до прибуття, відправлення і скидання пакетів, будуть отримані з використанням моніторів. Якщо знадобиться використовувати трейси, рекомендується не трасувати абсолютно всі події, оскільки це є надмірним. Монітори дозволяють безпосередньо отримати загальне число подій протягом зазначеного періоду часу, в той час як у випадку з трейсами, доведеться ще робити логічний аналіз вмісту файлу виводу для отримання цих значень.

2.3.5 Управління моделюванням

Після створення і конфігурації топології, має бути задано розклад запуску, зупинки та інших подій процесу моделювання.

Процес моделювання може бути запущений і зупинений в такий спосіб, що наведено на прикладі 2.31.

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

Приклад 2.31 – Запуск і зупинка процесу моделювання

Перша команда запускає в кінці моделювання процедуру «finish», а друга безпосередньо починає моделювання. Процедура «finish» повинна бути визначена для скидання буфера трасування, закриття трейс-файлів і завершення програми за допомогою підпрограми виходу. Також вона може

додатково запускати NAM (графічний аніматор мережі), проводити аналіз отриманих даних і будувати по ним графік.

Кінцева процедура мінімально повинна містити наступні елементи, що наведено на прикладі 2.32.

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

Приклад 2.32 – Кінцева процедура моделювання

Решта подій, такі як час початку і зупинки роботи клієнтів, можна задати наступним чином, що наведено на прикладі 2.33.

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

Приклад 2.33 – Задання подій моделювання

Якщо визначені власні процедури, то можна зробити так, щоб процедура запускалася, наприклад, кожні 5 секунд (приклад 2.34).

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

Приклад 2.34 – Запуск процедури кожні 5 секунд

2.3.6 Побудова графіків залежностей

NS-2 може генерувати шість різних типів графіків, що відображають основні параметри з’єднання TCP. Графіки можна переглядати, використовуючи програму «xplot», або її Java-версію «jPlot». Тип графіка задається за допомогою параметрів, переданих програмі «tcptrace», при цьому програма створює «*.xpl» файли даних в робочому каталозі, за якими будуються необхідні графіки (приклад 2.35).

```
xplot a2b_tsg.xpl
```

Приклад 2.35 – Задання типу графіка

NS-2 використовує ту ж систему найменування для з’єднань, яка була розглянута вище, з метою найменування файлів даних для графіків. Файли «a2b_*.xpl» і «b2a_*.xpl», містять графічну інформацію по обох напрямках трафіку для першого з’єднання, «c2d_*.xpl» і «d2c_*.xpl», відповідно для другого і так далі.

Основні типи графіків, що генеруються NS-2, а також параметри командного рядка для їх отримання наведено нижче:

- графік часової послідовності (Time Sequence Graph): генерується параметром «-s» і показує основну активність і події, що відбуваються під час існування з’єднання;

- графік пропускної здатності (Throughput Graph): доступний при використанні параметра «-T»;

- графік значень часу передачі пакету в установленому з’єднанні (RTT Graph): генерується параметром «-R» і показує зміну параметра RTT з плином модельного часу;

- графік непідтверджених даних (Outstanding Data Graph): створюється параметром «-N» і відображає кількість вже відправлених, але ще не підтверджених даних (використовується при визначенні приблизного розміру

вікна відправника);

- графік розміру сегмента (Segment Size Graph): генерується за допомогою параметра «-F» і ілюструє зміну розміру сегмента з плином модельного часу з'єднання;

- графік часу надходження пакетів (Time-Line Graph): створюється параметром «-L» і відображає процес передачі сегментів в обох напрямках протягом усього модельного часу з'єднання.

2.4 Моделювання бездротових ліній зв'язку

Процес вивчення бездротових мереж вимагає детального моделювання радіочастотних характеристик (взаємний вплив сигналів, ослаблення сигналу), взаємодії протоколів, і, в разі супутникових каналів, орбітальних ефектів (гравітаційні аномалії та інше).

Наприклад, геостаціонарні супутники обертаються навколо Землі на висоті 32000 км. Позиція супутників визначається в термінах довготи пункту нижчої точки (підсупутникових пункт на поверхні Землі). Практично, геостаціонарні супутники можуть дрейфувати від свого визначеного місцеположення завдяки гравітаційним хвилюванням – такі ефекти не моделюються за допомогою NS-2.

NS-2 дозволяє моделювати два види геостаціонарних супутників. Традиційний геостаціонарні супутники з «зігнутим каналом» (прості повторювачі на орбіті) – всі пакети, отримані такими супутниками по каналу з'єднання, передаються по каналах на частотах, характерних для відповідної інформації, і, таким чином, супутниковий вузол є невидимим для протоколів маршрутизації. Перспективні моделі супутників будуть все більше використовувати обробку смуги немодульованих частот, з метою відновлення цифрового сигналу і виконання швидкої комутації файлів на борту.

При моделюванні супутники можуть бути представлені як традиційні

вузли NS-2 з агентами маршрутизації і класифікаторами. NS-2 також дозволяє моделювати низькоорбітальні супутники: полярні орбітальні супутникові системи, такі як «Іридій» і система «Teledesic». Зокрема існує підтримка специфікації супутників, які обертаються в простих кругових площинах, для яких сусідні площини рухаються в тому ж напрямку. Існують інші негеостаціонарні можливі конфігурації (наприклад, сукупності Ходока) – користувач може розробити нові класи сукупності для моделювання інших типів сукупності, що вимагає визначення нового порядку зв’язку.

В NS-2 можуть моделюватися такі параметри ліній зв’язку: базове визначення сукупності (basic constellation definition); орбіти (orbits); межспутнікові лінії (intersatellite links); канали зв’язку «земля-супутник» (ground to satellite links).

Існує два основних види супутникових вузлів: геостаціонарні і негеостаціонарні. Крім того, вузли можуть бути розміщені на поверхні Землі. Фактично, такі вузли відносяться до одного класу, але з різним розташуванням, типом управління передачею і підключеними об’єктами зв’язку. Об’єкт положення відстежує місце розташування супутникового вузла в системі координат як функцію кінцевого часу моделювання. Ця інформація використовується для визначення затримки поширення і відповідного часу передачі даних.

Лінії супутникового зв’язку вельми схожі з бездротовими лініями зв’язку. У кожного супутникового вузла є один або більше стеків інтерфейсу супутникової мережі, з якими канали пов’язані об’єктом фізичного рівня. Лінії супутникового зв’язку відрізняються від бездротових ліній тим, що інтерфейси передачі і прийому повинні бути пов’язані з різними каналами без використання протоколу ARP.

У простому випадку бездротову лінію зв’язку можна представити як звичайний канал зв’язку з великою затримкою (до 0,5 секунди), використовуючи стандартні бібліотечні компоненти NS-2.

3 МОДЕЛЮВАННЯ ЗАПРОПОНОВАНИХ РЕЗУЛЬТАТІВ ДЛЯ БЕЗДРОТОВОГО З’ЄДНАННЯ В ОС LINUX

3.1 Опис поставленого завдання

Метою даного розділу є моделювання функціонування запропонованого методу на основі його реалізації у протоколі TCP при різних режимах його функціонування [11-14]. Особливий інтерес представляє процес управління перевантаженнями протоколу, що вносить основний вклад в його продуктивність. Для моделювання буде використана певна конфігурація мережевих з’єднань (топологія), що включає критичний ділянку, пропускна здатність якої є визначальною для з’єднань, що проходять крізь неї. Це означає, що сумарна пропускна здатність з’єднань буде значно перевищувати пропускну здатність такої ділянки.

При заданій топології комп’ютерної мережі необхідно створити сценарій моделювання, який врахує основні характеристики наявних з’єднань і дозволить спостерігати за змінами в роботі протоколу TCP з плином модельного часу. При цьому повинна бути можливість зміни режимів і параметрів роботи протоколу, таких як конкретна реалізація протоколу, пропускна здатність протоколу і каналів зв’язку, затримки в каналах, дисципліна обслуговування черги та інших. Відповідно будуть змінюватися значення параметрів, що відображають поточний стан протоколу, такі як розмір вікна перевантаження відправника (cwnd), розмір сегмента даних, частота відкидання пакетів, кількість переданих даних на поточний момент.

На підставі результатів необхідно зробити висновки про ефективність запропонованої модифікації протоколу TCP і про можливість застосування різних реалізацій протоколу TCP при заданих умовах.

3.2 Розробка топології комп’ютерної мережі

Для вирішення поставленого завдання буде використовуватися система NS-2, опис роботи з якою було приведено в розділі 2. Перед безпосереднім написанням tcl-скрипта, в якому і буде визначено сценарій, необхідно задати топологію бездротової мережі, що моделюється.

У контексті загальної задачі моделювання, дана топологія повинна відповідати таким вимогам:

- забезпечувати існування відправників і одержувачів трафіку;
- пропускна здатність каналів зв’язку повинна використовуватися максимальним чином;
- необхідна наявність критичної ділянки з метою штучного створення перевантажень;
- наявність проміжних маршрутизаторів з можливістю зміни механізму обслуговування черги;
- єдиність маршруту від відправників до одержувачів.

Розроблена топологія мережі, що відповідає необхідним вимогам, наведена на рисунку 3.1.

Тут «n1», «n3» – відправники даних, «n2», «n4» – одержувачі даних, «r1», «r2» – проміжні маршрутизатори. Відправники з’єднані з маршрутизатором лініями зв’язку з пропускною здатністю 500 Мбіт/с, в якості фізичного і канального рівнів стека протоколів в даному випадку може бути застосований стандарт Ethernet 1000Base-T. Транспортним протоколом для надійної доставки повідомлень одержувачу є TCP, поверх якого працює протокол прикладного рівня FTP, та максимальним чином завантажує наявну смугу пропускання.

Маршрутизатор «r1» здійснює маршрутизацію одержуваних пакетів і пересилку їх за допомогою бездротової технології з одержувачем, має асиметричну швидкість передачі 30/50 Мбіт/с, що набагато менше сумарної пропускної спроможності каналів, що з’єднують маршрутизатор з

відправниками. Значення смуги пропускання, що змінюється, в даному випадку застосовується для моделювання різної пропускної здатності маршрутизатора, і, відповідно, різної частоти відкидання пакетів. На нижніх рівнях стека протоколів даного з'єднання можна використовувати, наприклад, технологію Ethernet 1000Base-T. Необхідно відзначити, що при моделюванні в системі NS-2 питання, пов'язані з фізичною реалізацією передачі даних по лінії зв'язку, враховуються вельми обмеженим чином (задається тільки пропускна здатність і затримка), забезпечуючи, таким чином, незалежність функціонування протоколів більш високого рівня. Тому детально розглядати питання організації даного рівня не представляється доцільним.

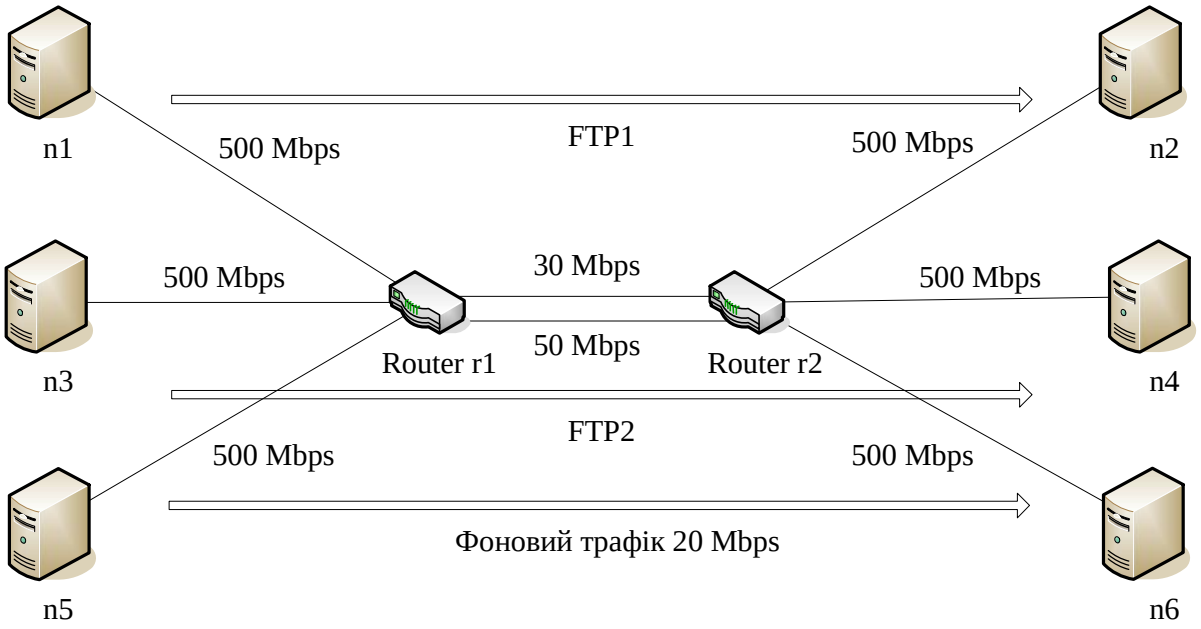

Рисунок 3.1 – Топологія мережі, що моделюється

У якості механізмів обслуговування черги, маршрутизатор застосовує реалізований в NS-2 алгоритм «DropTail» [15], що дозволяє моделювати стандартний буфер типу «FIFO» і при досягненні розміром черги певного значення, відкидає пакети, які надходять.

3.3 Створення сценарію моделювання

Після завдання топології комп’ютерної мережі, що моделюється, можна приступити до написання сценарію моделювання. З огляду на особливості розробленої топології і поставлене завдання з моделювання, виділимо наступні параметри сценарію:

- пропускна здатність ліній зв’язку маршрутизатора з відправниками даних – 500 Мбіт/с;
- пропускна здатність лінії зв’язку маршрутизатора з одержувачем трафіку – 500 Мбіт/с;
- затримка в лініях зв’язку маршрутизатора з відправниками – 2 мс;
- затримка в лінії зв’язку маршрутизатора з одержувачем – 2 мс;
- максимальна довжина черги маршрутизатора – 20 сегментів;
- реалізації протоколу TCP – NewReno і Hybla;
- розмір сегмента даних – 1400 байт;
- алгоритм обслуговування черги маршрутизатора – «DropTail» (з параметрами за замовчуванням, визначеними в ns-default.tcl);
- загальний час моделювання – 1000 с;
- час початку передачі даних відправниками – 70 і 130 с, відповідно, для першого і другого відправників;
- інтервал між отриманням значень характеристик – 5 с.
- в якості основного завантаження в мережі використовуються потоки даних протоколом FTP, який виконує передачу файлів нескінченної довжини.

Створюємо файл з назвою «model.tcl» який буде містити сценарій моделювання, і, відповідно до описаних вище методів роботи з системою NS-2, формуємо його вміст на мові Tcl [16].

Визначаємо новий об’єкт моделювання, що показано на прикладі 3.1.

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

Приклад 3.1 – Визначення нового об’єкту моделювання

Описуємо змінні, що використовуються в сценарії: колір графіків, що відображають потоки даних (color); спостережуваний параметр моделювання (statevar); інтервал запису значень модельованих параметрів (record_interval); файли в які заносяться ці значення (file1, file2). Відповідну ділянку коду наведено на прикладі 3.2 нижче.

```
$ns color 1 Blue
$ns color 2 Black
$ns color 3 Green
set statevar cwnd_
set record_interval 5
set file1 [open TCP_Hybla w]
set file2 [open TCP_NewReno w]
#set file3 [open file3.ns w]
```

Приклад 3.2 – Опис змінних для моделювання

Потім визначаємо процедуру «finish», в якій виконуються дії по закінченню моделювання, такі як: закриття файлів з даними, скидання буфера трасування, виклик програми побудови графіків отриманих значень [17, 18]. Текст процедури наведено на прикладі 3.3 нижче.

```
#Define a 'finish' procedure
proc finish {} {
        global ns nf file1 file2 file3 statevar
        close $file1
        close $file2
#     close $file3
        $ns flush-trace
          eval "exec xgraph -nb -bg white -lw 1 -zg black -x t -y wnd -t
TCP_Model TCP_Hybla TCP_NewReno" &
#          eval "exec xgraph -nb -bg white -lw 1 -zg black -x t -y wnd -
t TCP_Model TCP_NewReno" &
        exit 0
}
```

Приклад 3.3 – Процедура «finish»

Створюємо процедуру «record», яка в необхідні моменти часу записує значення модельованих параметрів в файл. Для цього визначається поточний час моделювання отриманням значення змінної «$ns now» і значення

спостережуваних параметрів по кожному з’єднанню («$tcp1 set $statevar» і «$tcp2 set $statevar»), а потім пари значень «поточний час моделювання – спостережуваний параметр» записуються в файли. Після цього процедура повторно ставиться до розкладу запуску, а наступний час запуску обчислюється як сума поточного та інтервалу запису значень (record_interval). Відповідний фрагмент коду наведено на прикладі 3.4 нижче.

```
#Define a ’record’ procedure
proc record {} {
global ns tcp1 file1 tcp2 file2 udp1 file3 statevar record_interval
#Set the time after which the procedure should be called again
set time $record_interval
      #Get the current time
      set now [$ns now]
puts $file1 "$now [$tcp1 set $statevar]"
puts $file2 "$now [$tcp2 set $statevar]"
#     puts $file3 "$now [$udp1 set $statevar]"
#Re-schedule the procedure
$ns at [expr $now+$time] "record"
}
```

Приклад 3.4 – Процедура «record»

Наступний код (приклад 3.5) описує задану топологію: створюємо чотири вузли («n1», «n3», «n5» – відправники, «r1», «r2» – маршрутизатори і «n2», «n4», «n6» – адресати) і описуємо лінії зв’язку, що їх з’єднують.

```
set n1 [$ns node]
set n2 [$ns node]
set n3 [$ns node]
set n4 [$ns node]
set n5 [$ns node]
set n6 [$ns node]
set r1 [$ns node]
set r2 [$ns node]
$ns duplex-link $n1 $r1 500Mb 2ms DropTail
$ns duplex-link $n3 $r1 500Mb 2ms DropTail
$ns simplex-link $r1 $r2 30Mb 300ms DropTail
$ns simplex-link $r2 $r1 50Mb 300ms DropTail
$ns duplex-link $n2 $r2 500Mb 2ms DropTail
$ns duplex-link $n4 $r2 500Mb 2ms DropTail
$ns duplex-link $n5 $r1 500Mb 2ms DropTail
$ns duplex-link $n6 $r2 500Mb 2ms DropTail
```

Приклад 3.5 – Опис топології

Задаємо максимальну довжину черги маршрутизатора (приклад 3.6).

```
#Set Queue Size of link (r1-r2) to $QUEUE
$ns queue-limit $r1 $r2 $QUEUE1
$ns queue-limit $r2 $r1 $QUEUE2
```

Приклад 3.6 – Встановлення максимальної довжини черги маршрутизатора

Далі необхідно визначити і налаштувати два з’єднання TCP. Для цього визначаємо агентів і адресатів трафіку TCP, пов’язуємо їх з необхідними вузлами, встановлюємо з’єднання між агентами і адресатами, задаємо максимальний розмір вікна TCP. Дані операції здійснюються такими рядками коду (приклад 3.7).

```
#Setup two TCP connections
set tcp1 [new Agent/TCP/Linux]
$ns attach-agent $n1 $tcp1
set sink1 [new Agent/TCPSink/Sack1/DelAck]
$ns attach-agent $n2 $sink1
$ns connect $tcp1 $sink1
$tcp1 set window_ 1400
set tcp2 [new Agent/TCP/Linux]
$ns attach-agent $n3 $tcp2
set sink2 [new Agent/TCPSink/Sack1/DelAck]
$ns attach-agent $n4 $sink2
$ns connect $tcp2 $sink2
$tcp2 set window_ 1400
```

Приклад 3.7 – Визначення і налаштування з’єднань

Для генерації даних, що відправляються, поверх транспортного протоколу TCP працює протокол прикладного рівня FTP. Його опис включає в себе створення необхідних агентів, і прив’язування їх до наявних TCP з’єднань (приклад 3.8).

```
#Setup two FTP over TCP connections
set ftp1 [new Application/FTP]
$ftp1 attach-agent $tcp1
$ftp1 set type_ FTP
set ftp2 [new Application/FTP]
$ftp2 attach-agent $tcp2
$ftp2 set type_ FTP
```

Приклад 3.8 – Опис протоколу FTP

В процесі моделювання здійснюється запуск необхідних процедур в зазначений час за допомогою наступного коду (приклад 3.9).

```
#Schedule events for the FTP agents
$ns at $START "record"
$ns at $START "$tcp1 select_ca hybla"
$ns at $START "$tcp2 select_ca reno"
$ns at $HYBLA_START "$ftp1 start"
$ns at $HYBLA_STOP  "$ftp1 stop"
$ns at $NEWRENO_START "$ftp2 start"
$ns at $NEWRENO_STOP  "$ftp2 stop"
$ns at 170.0 "$cbr1 start"
$ns at 930.0 "$cbr1 stop"
#Call the finish procedure after $END seconds of simulation time
$ns at $END "finish"
```

Приклад 3.9 – Запуск процедур в зазначений час

Запуск процесу моделювання здійснюється за допомогою команди, ща наведено на прикладі 3.10.

```
$ns run
```

Приклад 3.10 – Запуск процесу моделювання

3.4 Результати моделювання

Розроблений в попередньому підрозділі файл зі сценарієм моделювання можна запустити на виконання за допомогою наступної команди, що наведено на прикладі 3.11.

```
ns2 model.tcl
```

Приклад 3.11 – Запуск файлу на виконання

В результаті такої команди, NS-2 запускає інтерпретатор мови «Tcl», який покроково виконує рядки зазначеного tcl-скрипта, перевіряючи їх синтаксис і коректність операцій. У разі помилок виводиться повідомлення в стандартний потік виведення і припиняється виконання скрипта. Результатом запуску сценарію є виклик програми візуалізації графіків «xgraph», яка, використовуючи дані з певних в сценарії файлів, будує графіки зміни значення модельованого параметра з плином часу моделювання.

У додатку Б наведено результати моделювання в графічному вигляді для різних параметрів реалізацій протоколу TCP, а також їх коротке теоретичне обгрунтування. Як було зазначено, в будь-який момент кількість переданих сегментів даних не може перевищувати меншого зі значень «cwnd» і заявленого вікна одержувача («awnd»). З огляду на те, що при виникненні перевантажень, розмір вікна «cwnd» в більшості випадків значно менше розміру вікна «awnd», можна зробити висновок, що загальна продуктивність протоколу при даному режимі роботи прямо пропорційна розміру вікна перевантаження «cwnd». Таким чином, проаналізувавши поведінку даної характеристики при модельованих умовах роботи протоколу TCP, стане можливим вирішити поставлене завдання.

На рисунку 3.2 наведено результати моделювання в разі встановлення одного з’єднання TCP реалізації TCP Hybla, смуга пропускання асиметричного бездротового канала зв’язку – 100 Мбіт/с і 50 Мбіт/с, відповідно, затримка поширення сигналу становить 250 мс, довжина черги маршрутизатора становить 200 пакетів, механізм обслуговування черги – «DropTail». Генерація трафіку відбувається в моменти часу від t_1 = 130 с до t_2 = 970 с.

За представленими результатами можна зробити висновок, що спостерігаються значні (більше 80% від максимального значення) флуктуації

вікна перевантаження, що призводить до недовикористання всього доступного з’єднанню розміру смуги пропускання каналу.

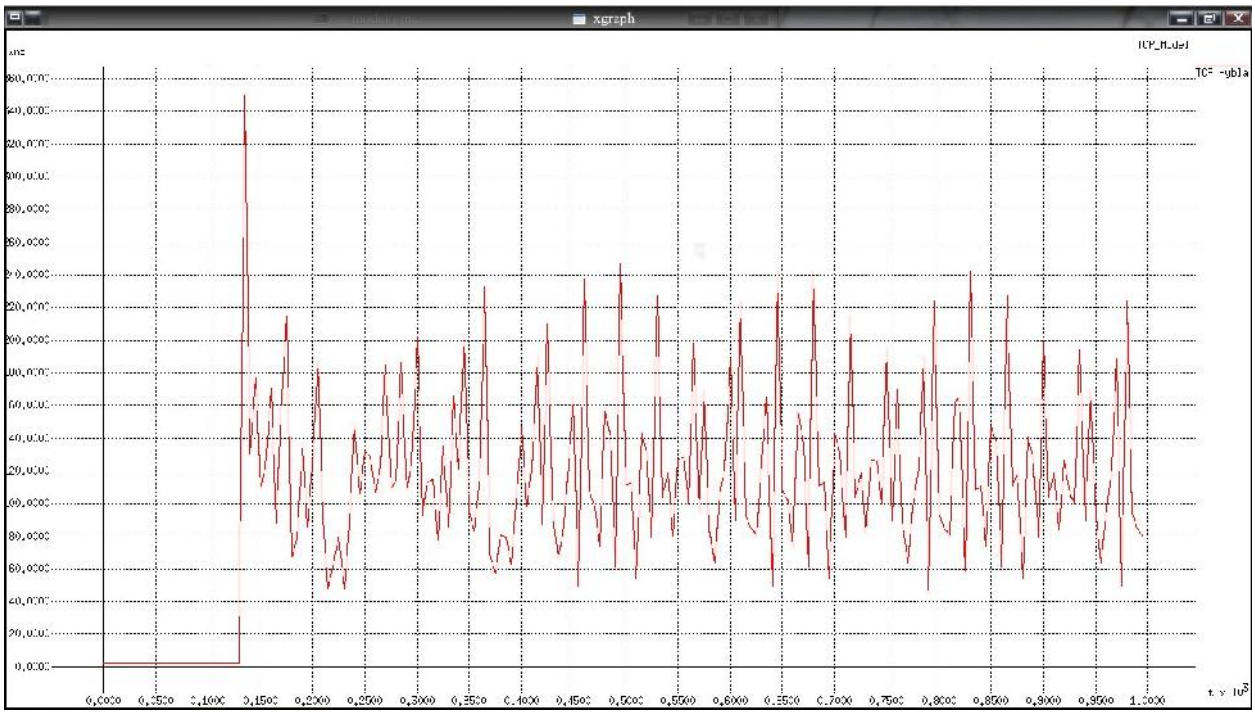

Рисунок 3.2 – Реализація TCP Hybla

На рисунку 3.3 показано результати моделювання в разі встановлення в каналі з’єднання TCP NewReno, параметри бездротового каналу зв’язку і черги маршрутизатора залишаються такими ж, як і в попередньому експерименті.

У реалізації TCP NewReno також спостерігаються значні (більше 30% від максимального значення) флуктуації вікна перевантаження, що призводить до недовикористання всього доступного з’єднанню розміру смуги пропускання каналу. Флуктуації вікна викликані значними затримками часу поширення сигналу в бездротовому каналі зв’язку.

Рисунок 3.4 ілюструє випадок однорідного мережного середовища, в якому відкриті два з’єднання протоколами TCP Hybla і TCP NewReno, що відрізняються один від одного використанням різних підходів до визначення факту настання перевантаження.

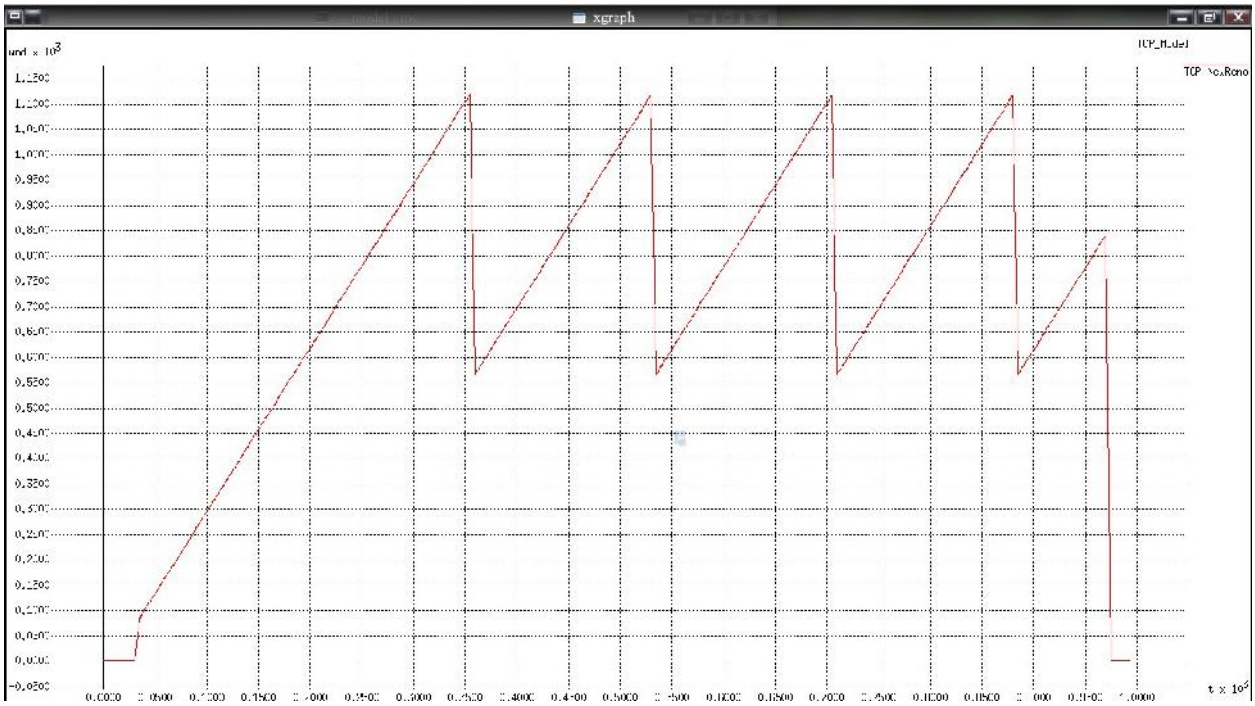

Рисунок 3.3 – Реализація TCP NewReno

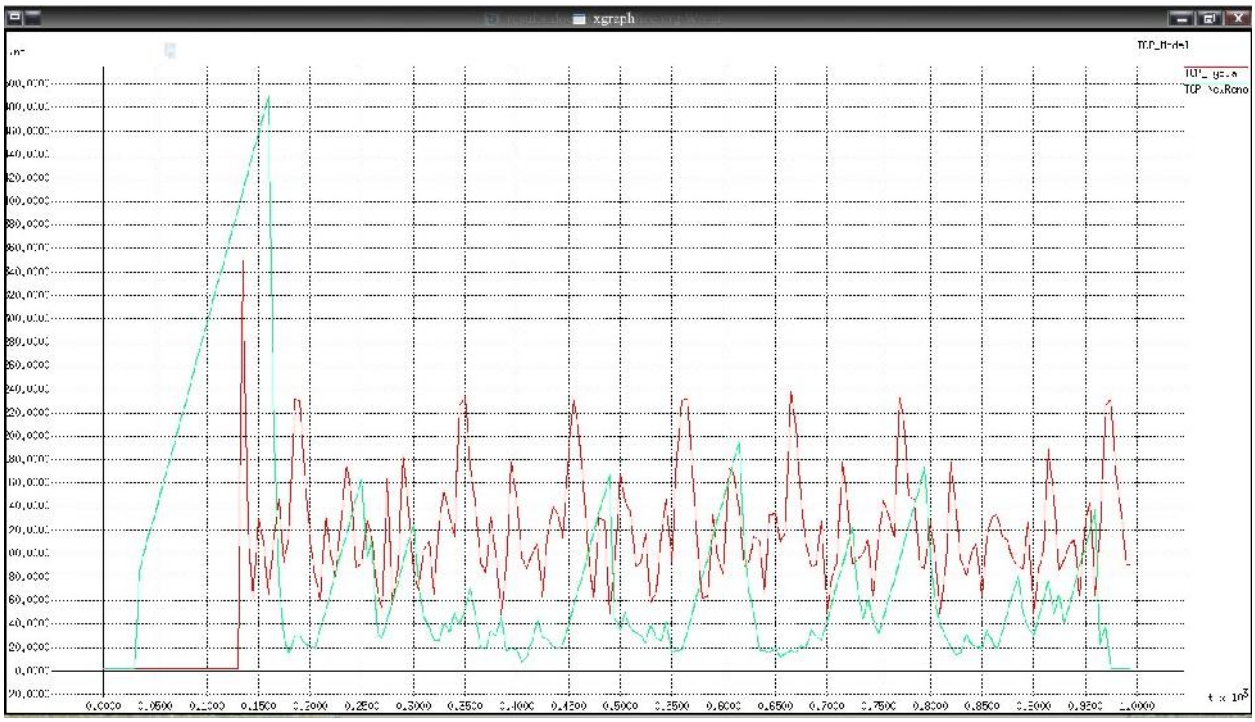

Рисунок 3.4 – Реализації TCP NewReno і TCP Hybla

В інтервалі часу 70-970 с з відбувається передача трафіку FTP

протоколом TCP NewReno. У момент часу t = 130 с в цьому ж каналі створюється з’єднання протоколом TCP Hybla і починається генерація трафіку. Після переходу цих протоколів в фазу запобігання перевантажень спостерігається нерівномірність розподілу доступного розміру смуги пропускання каналу між одночасно існуючими з’єднаннями, поряд зі значними флуктуаціями розмірів їх вікон, причому набагато більша і нерівномірна амплітуда змін спостерігається в протоколі TCP NewReno.

На рисунку 3.5 наведено сценарій, коли першим встановлюється з’єднання протоколом TCP Hybla. Протоколу TCP NewReno не вдається досягти розміру вікна, порівнянного з перевантажним вікном протоколу TCP Hybla, і це відбувається на всьому інтервалі часу 200-970 с.

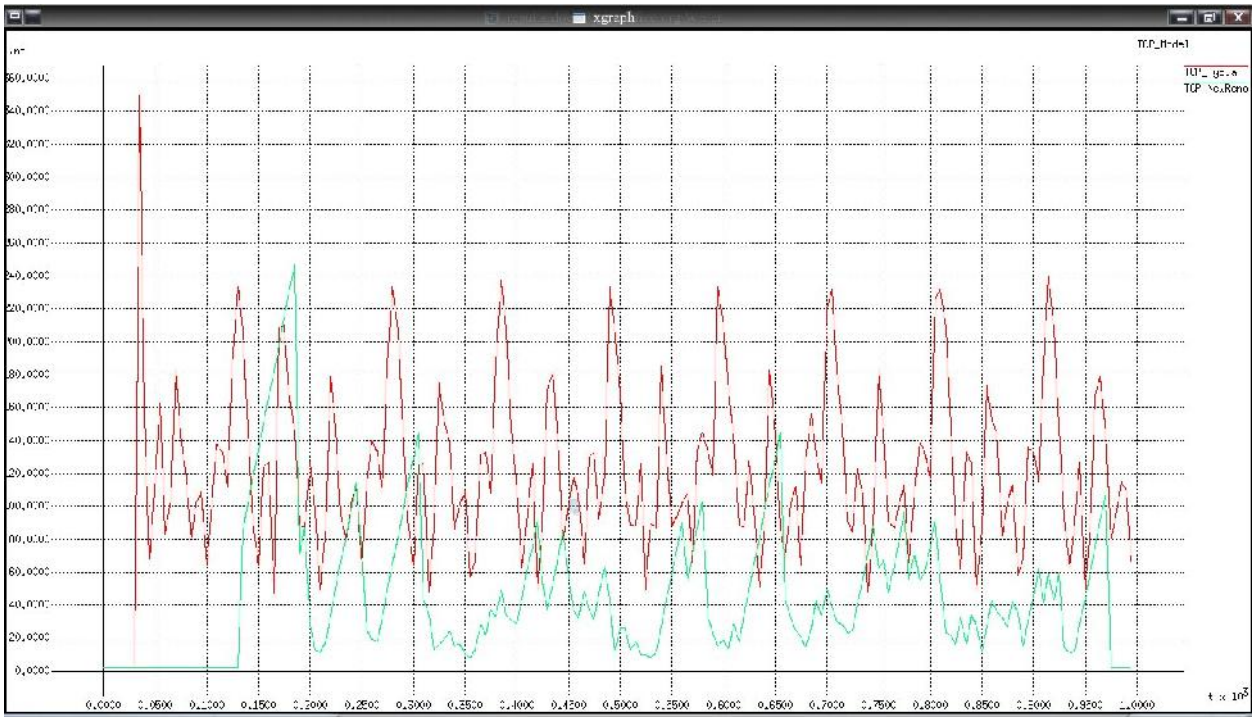

Рисунок 3.5 – Реализації TCP Hybla і TCP NewReno

На рисунку 3.6 представлено сценарій аналогічний наведеному на рисунку 3.4, з тією лише відмінністю, що зменшено в два рази розмір черги маршрутизаторів і прийнято рівним 10 пакетам.

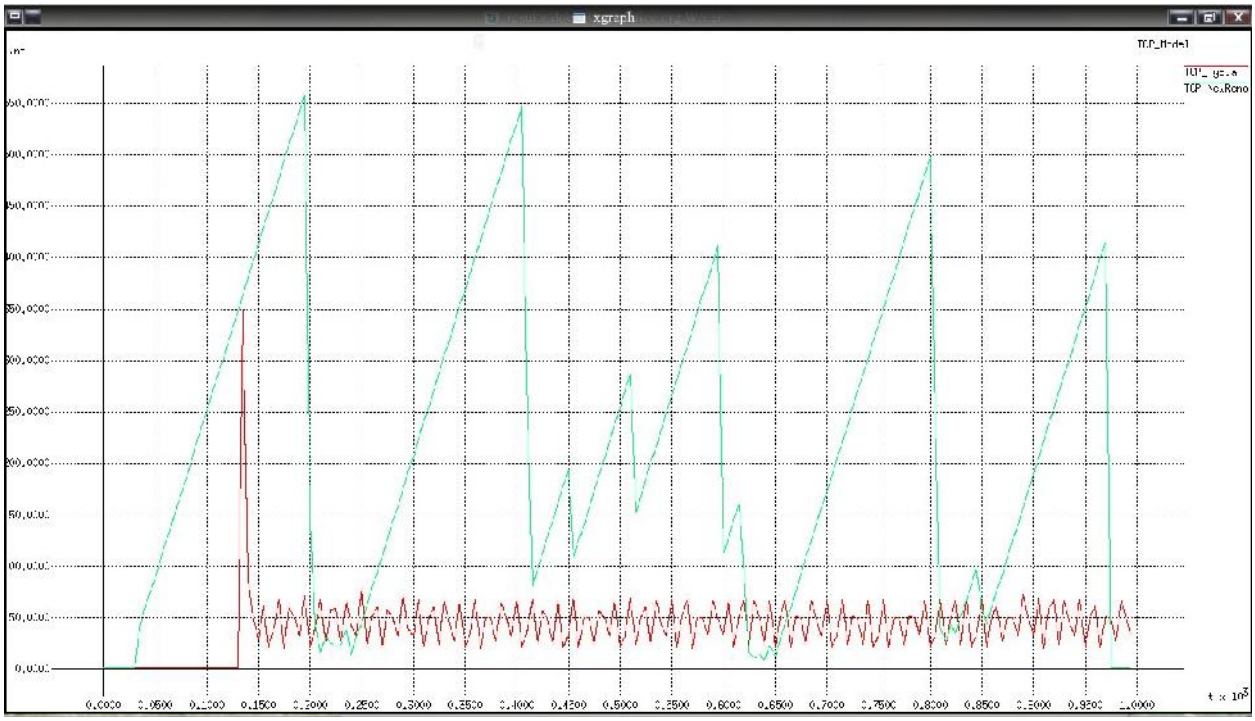

Рисунок 3.6 – Реализації TCP Hybla і TCP NewReno при зменшеному розмірі черги в маршрутизаторі

В даному випадку спостерігається кількісне переважання протоколу TCP NewReno, однак флуктуації вікна приблизно на 30% перевищують його середній розмір, що може привести до катастрофічного зниження його продуктивності. У той же час розмір вікна TCP Hybla залишається невеликим, хоча стабільним (середній розмір вікна близько 50 пакетів), що обумовлено низьким значенням розміру буферної пам’яті маршрутизатора. Таким чином, можна зробити висновок, що продуктивність протоколу TCP Hybla залежить від розміру буферної пам’яті і знижується при її зменшенні.

На рисунку 3.7 представлено сценарій, який полягає в збільшенні часу передачі пакету в бездротовому каналі зв’язку з 200 мс до 300 мс.

Тут спостерігається значна зміна розміру вікна протоколу TCP NewReno (більш ніж в 20 разів), що може говорити про його непристосованості до затримок такого порядку в каналі, і може бути обумовлено опцією вибіркових підтверджень відразу декількох вікон. Розмір вікна для протоколу TCP Hybla хоча і стабільний, але невеликий (близько 90-

95 пакетів), що може бути обумовлено штучно зменшеним розміром буферної пам’яті. Таким чином, тут спостерігаються безперечні переваги протоколу TCP Hybla в характерному для бездротових каналів зв’язку мережевому середовищі.

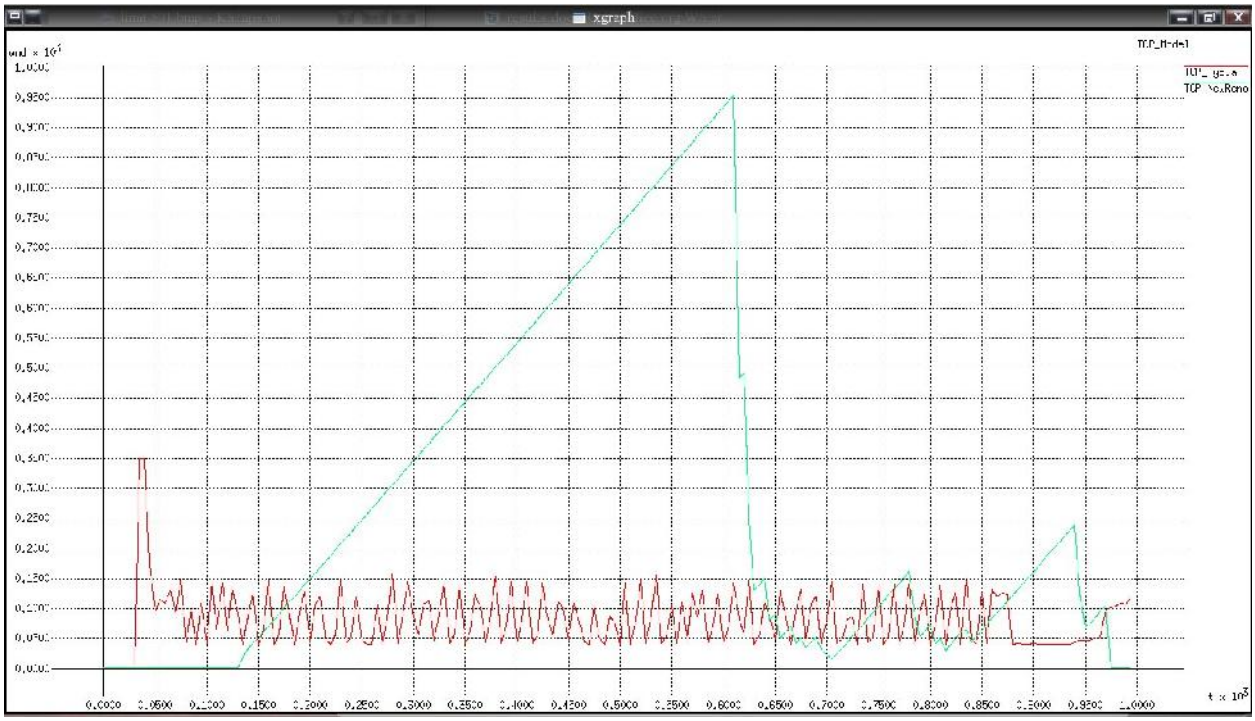

Рисунок 3.7 – Реалізації TCP Hybla і TCP NewReno при збільшенні часу передачі сигналу в бездротовому каналі

При збільшенні буферної пам’яті, але значному зменшенні пропускної здатності бездротового каналу зв’язку (рисунок 3.8) картина протилежна попереднім сценаріям. Тут видно значні флуктуації вікна для протоколу TCP Hybla при практично незмінному його середньому значенні, що дорівнює приблизно 500-600 пакетів. В цей же час розмір вікна для TCP NewReno приблизно дорівнює 50 пакетам, що на порядок менше. Це ще раз доводить кращу адаптацію протоколу TCP Hybla до бездротових каналів зв’язку, навіть при таких штучно заданих обмеженнях.

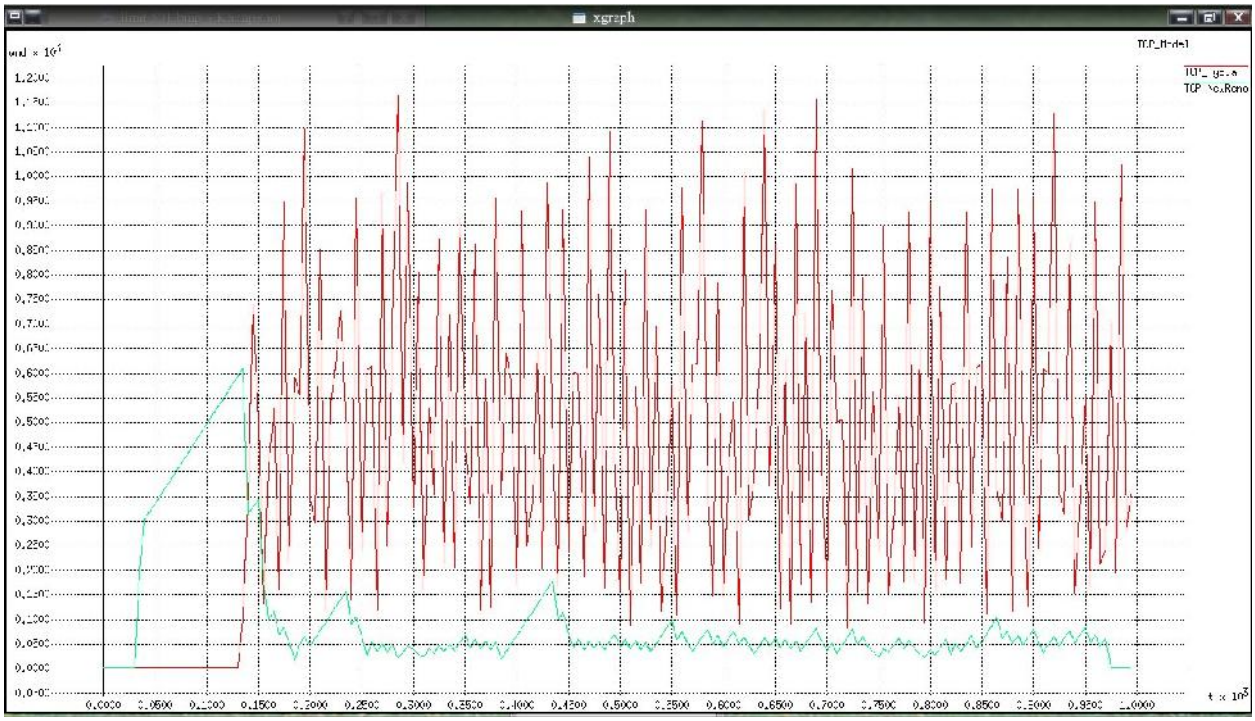

Рисунок 3.8 – Реалізації TCP Hybla і TCP NewReno при зменшенні пропускної здатності бездротового каналу зв’язку

Перейдемо до моделювання запропонованого методу і відповідної модифікації протоколу TCP. При моделюванні асиметричного бездротового каналу зв’язку (з симплексними лініями 30 Мб/с і 50 Мб/с) із затримкою в 700 мс, явну перевагу як за розміром вікна перевантаження (рисунок 3.9), так і за пропускною здатністю (рисунок 3.10) показує модифікований протокол TCP Hybla. Навіть при зменшенні затримки до 2 мс, ситуація не сильно змінюється на користь TCP NewReno (рисунки 3.11 і 3.12). Таким чином, час передачі пакета даних в установленому з’єднанні вже не є критичним при виборі транспортного протоколу.

За отриманими експериментальними даними можна зробити висновок, що на ефективність реалізацій протоколу TCP впливають такі параметри як пропускна здатність складеного каналу зв’язку, затримка поширення сигналу в каналі зв’язку, також розмір черги пакетів в проміжних пристроях маршрутизації.

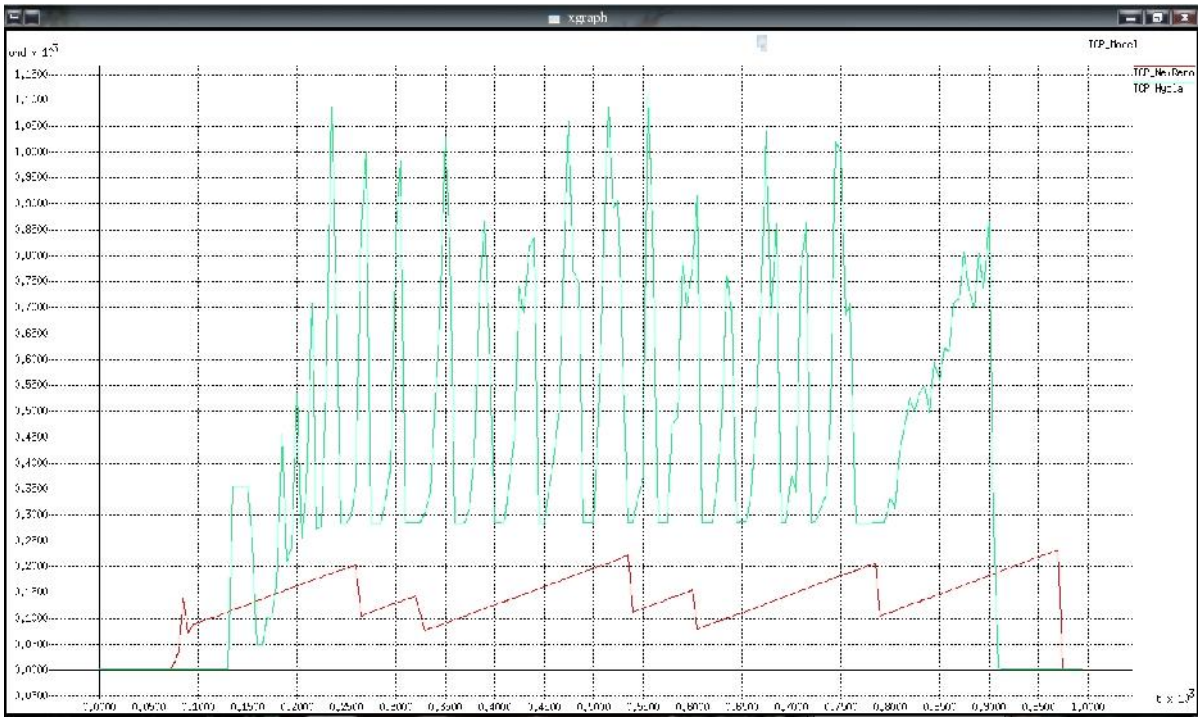

Рисунок 3.9 – Зміна розміру вікна при збільшенні затримки в бездротовому каналі до t = 700 мс

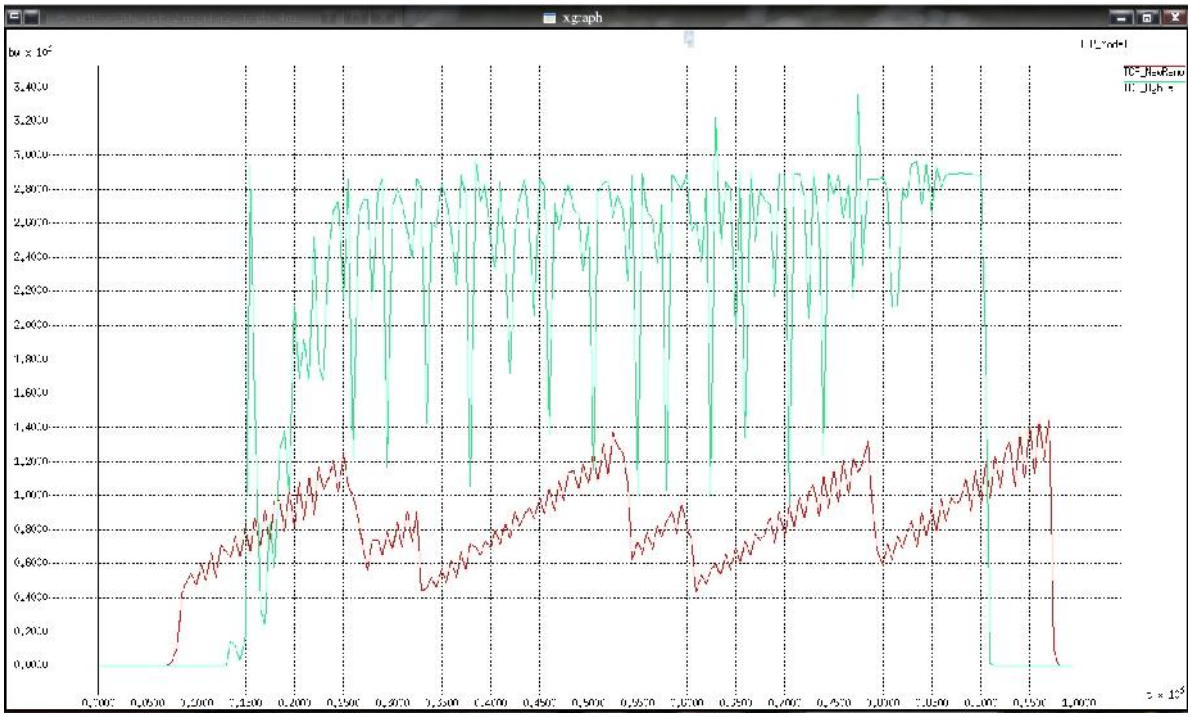

Рисунок 3.10 – Зміна пропускної здатності протоколів при збільшенні затримки в бездротовому каналі до t = 700 мс

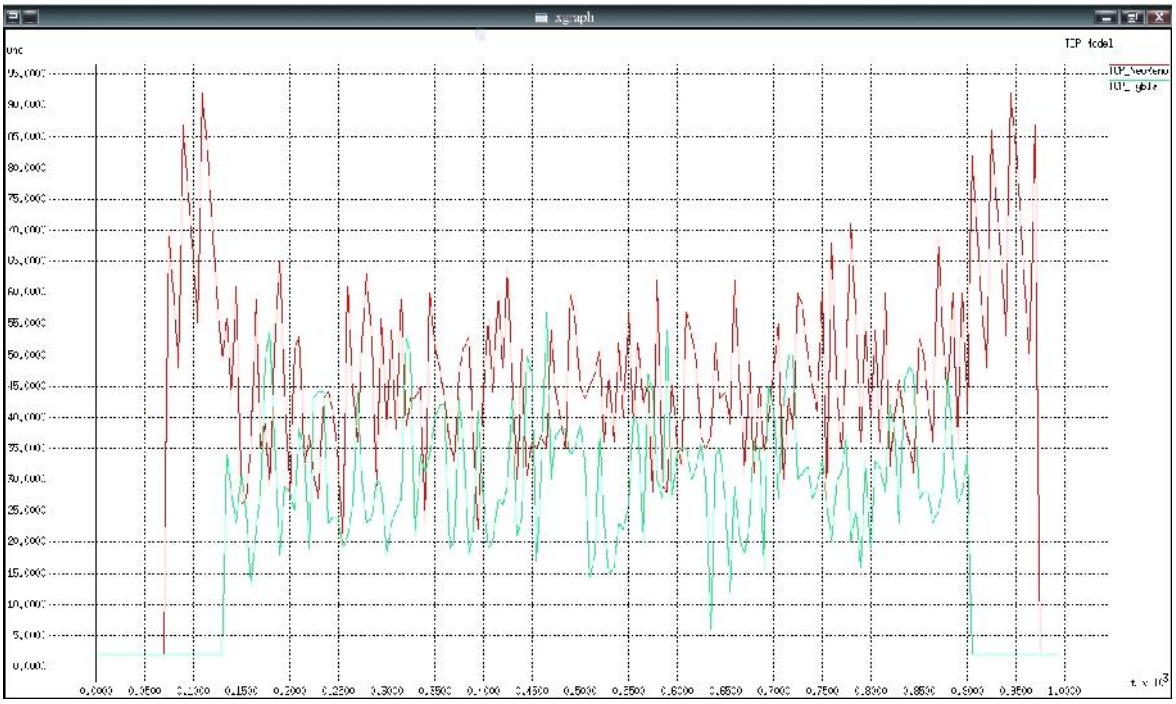

Рисунок 3.11 – Зміна розміру вікна при зменшенні затримки в бездротовому каналі до t = 2 мс

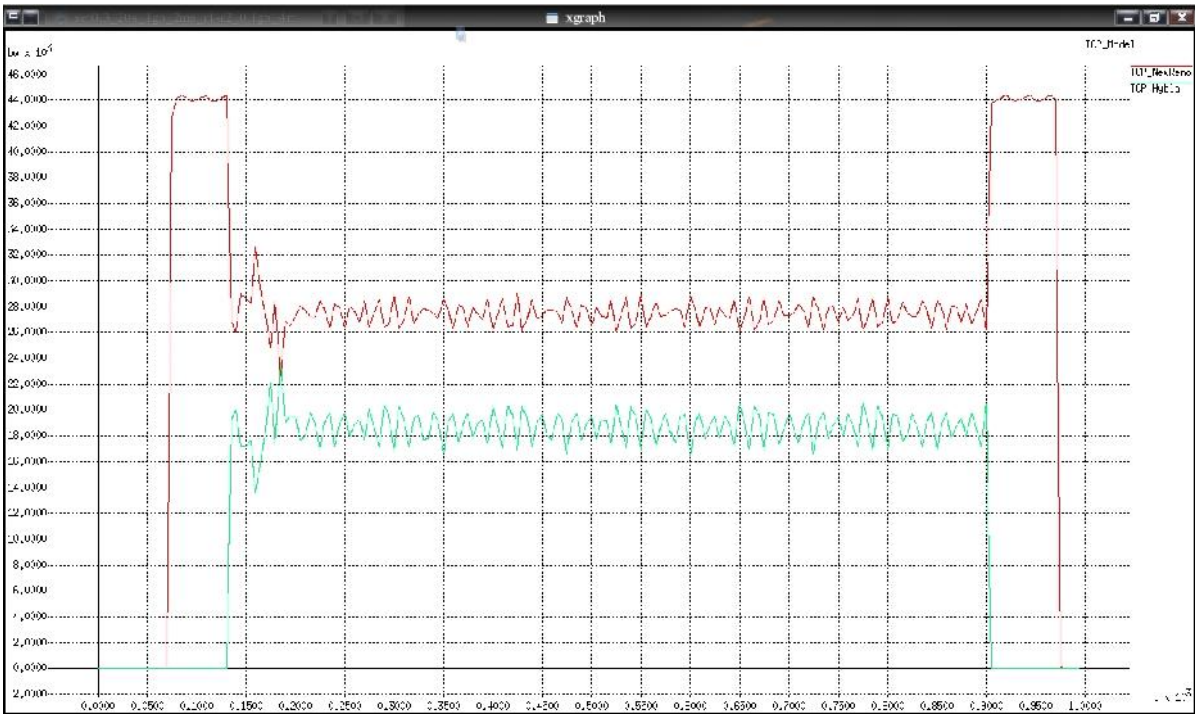

Рисунок 3.12 – Зміна пропускної здатності протоколів при зменшенні затримки в бездротовому каналі до t = 2 мс

Як критерій вибору реалізації протоколу TCP можна запропонувати використовувати затримку поширення сигналу в каналі зв’язку. В отриманих результатах, застосування реалізації TCP Hybla дає позитивний ефект при досягненні затримки поширення сигналу починаючи зі значення 30 мс і вище (рисунки 3.13, 3.14). При цьому зміна пропускної здатності асиметричного бездротового з’єднання в незначній мірі впливає на флуктуації вікна перевантаження розглянутих реалізацій протоколу TCP. З рисунка 3.13 видно, що флуктуації вікна перевантаження розглянутих реалізацій відбуваються практично синхронно, при цьому TCP Hybla переважає. На рисунку 3.14 показано пропускну здатність кожної з реалізацій. Тут також можна зробити висновок, що коли в каналі відбувається створення з’єднання TCP Hybla, продуктивність реалізації TCP NewReno різко знижується і канал починає використовуватися новим з’єднанням. Тому можна прийти до висновку, що в каналах з великими затримками більш доцільно використовувати реалізації протоколу TCP, такі як TCP Hybla, що враховують в логіці своєї роботи можливе велике значення часу передачі пакета поряд зі змінною пропускною спроможністю каналу зв’язку.

Як апаратних засобів, що дозволяють в повній мірі реалізувати переваги запропонованої модифікації протоколу TCP, можна рекомендувати наступні.

Маршрутизатор з інтегрованими сервісами Cisco ISR 4431, що дозволяє підтримувати такі функції, як:

- одночасна робота різних сервісів (наприклад, забезпечення безпеки і голосового зв’язку) на швидкості фізичної лінії, а також розширених сервісів на повній швидкості T3/E3;
- підвищена модульність;
- хороший економічний ефект внаслідок підвищеної продуктивності та модульності;

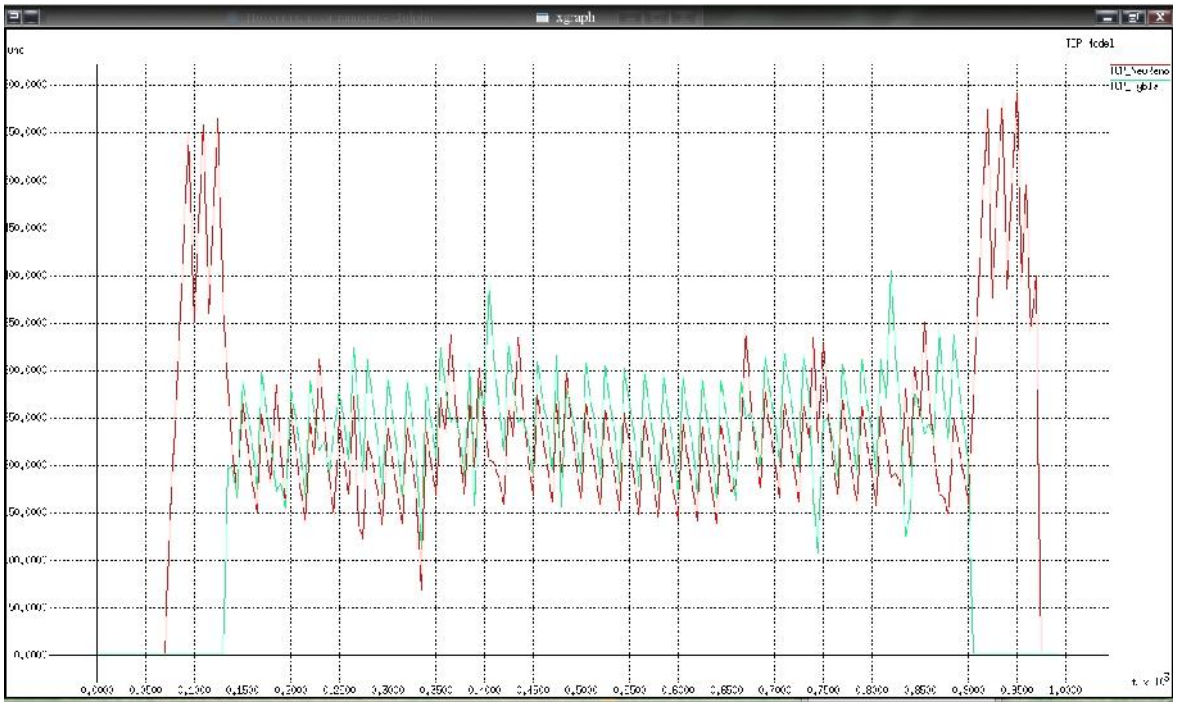

Рисунок 3.13 – Зміна розміру вікна при значенні затримки в бездротовому каналі t = 30 мс

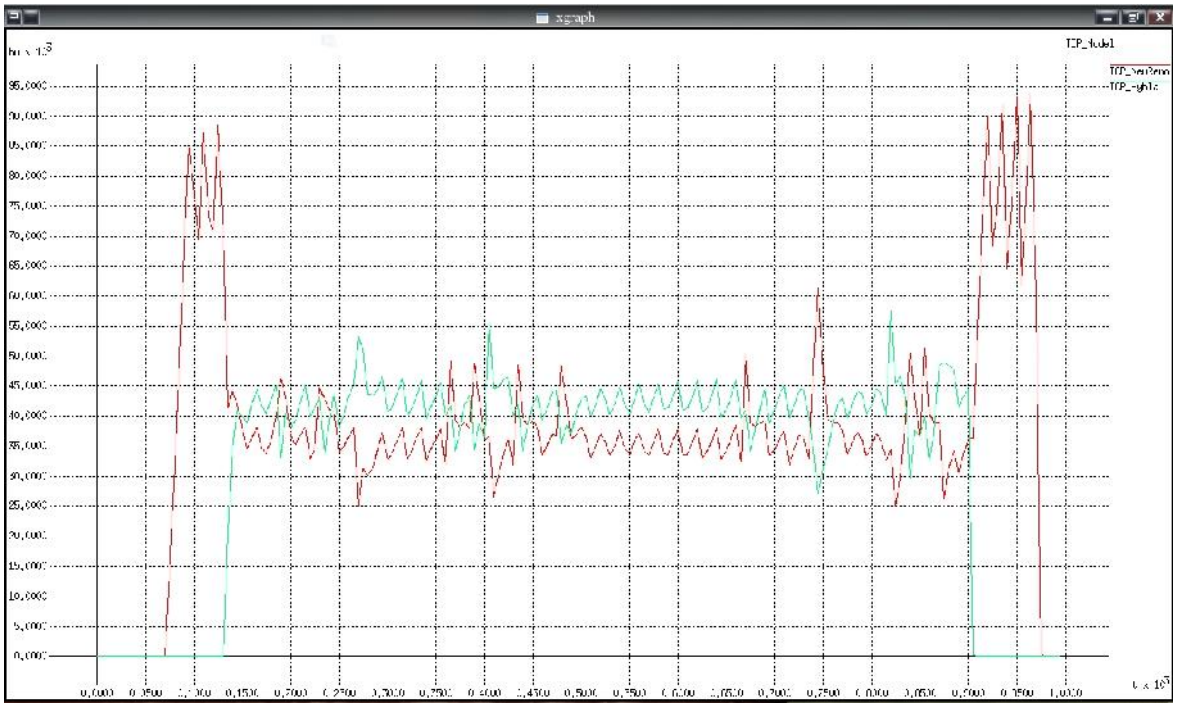

Рисунок 3.14 – Зміна пропускної здатності при значенні затримки в бездротовому каналі t = 30 мс

- збільшена щільність завдяки чотирьом слотам високошвидкісних інтерфейсних карт;
- слот розширеного мережевого модуля;
- підтримка більш 90 існуючих і нових модулів;
- підтримка більшості існуючих модулів AIM, NM, WIC, VWIC та VIC;
- вбудовані порти GE з підтримкою оптичних мереж і мереж на основі витої пари;
- можливість підтримки комутації 2-го рівня з функцією живлення по кручений парі (PoE) (як опція), підтримка 36-портового модуля Cisco EtherSwitch (NMD-36ESW);
- безпеку, включаючи вбудовану систему шифрування, підтримку до 2500 тунелів віртуальних приватних мереж, систему захисту від вірусів з використанням технології контролю доступу в мережу (NAC), систему запобігання вторгнень, підтримку брандмауера з повним контролем стану з'єднань Cisco IOS;
- передача голосу, включаючи підтримку аналогових і цифрових голосових викликів, підтримку функцій голосової пошти, підтримку Cisco CallManager Express для обробки голосових викликів (до 240 IP-телефонів), можливість підтримки телефонії для віддалених об'єктів з можливістю автономної роботи для обробки голосових викликів в філіях невеликих підприємств (до 720 IP-телефонів).

У свою чергу, можливе використання комутаторів Cisco Catalyst серії 9400 в разі переважної обробки даних. Такі комутатори мають наступні можливості:

- максимальна продуктивність на швидкості середовища передачі;
- мала затримка при комутації на рівнях 2-4;
- розвинені функції безпеки;
- наявність висхідних портів Gigabit Ethernet та 10 Gigabit Ethernet;
- можливість підключення опціональних внутрішніх джерел живлення

змінного або постійного струму з можливістю «гарячої» заміни і резервуванням за схемою «1 + 1»;

- вентиляторний модуль з можливістю «гарячої» заміни і резервованими вентиляторами.

Комутатор Catalyst 9400 поставляється в корпусі висотою 1RU і включає в себе 48 портів Gigabit Ethernet, комутованих на швидкості середовища передачі, а також висхідні порти Gigabit Ethernet або 10G Ethernet.

Модель комутатора Catalyst 9400M є полумодульний комутатор висотою 2RU з 8 висхідними портами 10G Ethernet, що працюють на швидкості середовища передачі, з можливістю підключення серверів на швидкостях Gigabit Ethernet і 10G Ethernet.

ВИСНОВКИ

В результаті виконання атестаційної роботи було проведено аналіз функціонування сучасних мережевих транспортних протоколів в бездротових з’єднаннях. Проаналізовано методи управління перевантаженнями, що використовуються в них, їх взаємодія між собою та іншими протоколами стеку TCP/IP, а також запропоновано відповідні модель та метод для управління передачею даних бездротового з’єднання в ОС Linux.

На основі запропонованого методу розроблено відповідну модифікацію протоколу TCP Hybla, що дозволяє підвищити його продуктивність в бездротових мережах. Розглянуто існуючі програмні засоби для вивчення ефективності мережевих протоколів і зроблено вибір системи імітаційного моделювання Ns-2, за допомогою якої було промодельовано роботу різних реалізацій протоколу TCP, у тому числі розробленої на основі запропонованого методу.

Проведено аналіз результатів, отриманих на основі імітаційного моделювання моделі запропонованої модифікації протоколу. Зроблено висновки про можливість застосування сучасних реалізацій протоколу TCP, як окремо, так і разом із запропонованою модифікацією протоколу.

Згідно з результатами моделювання запропонованої реалізації протоколу в бездротових каналах зв’язку було встановлено, що:

- запропонована модифікація дозволяє протоколу TCP Hybla досягати більшої продуктивності (приблизно в 1,2 рази) в ОС Linux в порівнянні з його немодифікованої версією за рахунок можливості відновлення більшого числа втрачених пакетів в одиницю часу;

- запропонована модифікація повторює динаміку оригінального протоколу TCP Hybla в ОС Linux за умови відсутності втрат в бездротовому каналі.

Практичне значення отриманих результатів полягає в тому, що розглянуті методи дослідження протоколів можуть застосовуватися для оцінки наявної організації комп’ютерних мереж, і розробки рекомендацій по збільшенню їх надійності і продуктивності.

Також матеріали даної атестаційної роботи можуть бути використані в освітніх цілях, оскільки детальним чином відображають механізми функціонування протоколу TCP і використання розглянутих інструментальних засобів з оцінки його ефективності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Коваленко А.А., Рубан І.В., Кучук Г.А., Помінчук Ю.В. Метод управління передачею даних бездротових 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. 78.

2. 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.

3. Thompson K., Miller G.J., Wilder R. Wide-area Internet traffic patterns and characteristics // IEEE Networks. – 2012.

4. Нанс Б. Компьютерные сети / Б. Нанс. – М.: Бином, 2015. – 394 с.

5. Request for Comments: 2581. 2011. – 11 p.

6. The ns Manual / A Collaboration between researchers at UC Berkeley, LBL, USC/ISI, and Xerox PA., 2013. – 392 p.

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

8. Johanna Antila TCP Performance Simulations Using Ns2. – 2010. – 28 p.

9. Fall, K. and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", Computer Communication Review, July 2006.

10. Гладун А.Я. Аналіз моделей процедур хендовера в бездротових мережах, базованих на технології ATM / А. Я. Гладун, О. І. Проценко // Наукові записки НаУКМА. – 2013. – Т. 21: Комп’ютерні науки.

11. Столлингс В. Современные компьютерные сети / В. Столлингс. – 4-е изд. – СПб.: «Питер», 2014. – 385 с.

12. Brakmo L. TCP Vegas: new techniques for congestion detection and

avoidance / L. Brakmo, S. O’Malley, L. Peterson // Proceedings of ACM SIGCOMM. – 2014. – Pp. 24-35.

13. Вегенша Ш. Качество обслуживания в сетях IP / Ш. Вегенша. – М.: Издательский дом “Вильямс”, 2013. – 386 с.

14. Bernet Y. The Complementary Roles of RSVP and Differentiated Services in the Full-Service Qos Network / Y. Bernet // IEEE Communications Magazine. – 2010. – Pp. 56-63.