Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи підвищення пропускної здатності каналів комп’ютерних мереж

(тема)

Виконав:

студент II курсу, групи КСМм-23-1

Сокирко М.А.
(прізвище, ініціали)

Спеціальність

123 «Комп’ютерна інженерія»
(код і повна назва спеціальності)

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

Освітня програма

Комп’ютерні системи та мережі
(повна назва освітньої програми)

Керівник: доц. Янковський О.А.
(посада, прізвище, ініціали)

Допускається до захисту

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

2025 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-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Комп’ютерні системи та мережі
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
“____” _____ 20__ р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Сокирці Максиму Анатолійовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Методи підвищення пропускної здатності каналів комп’ютерних мереж

затверджена наказом по університету від “ 22 ” листопада 2024 р. № 1237 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 20 січня 2025 р.

3. Вхідні дані до роботи

1) Моделі та методи для керування мережевими інформаційними потоками

2) Сучасні вимоги до мережних показників

3) Стек протоколів TCP/IP

4) Перелік використаних програмних та апаратних засобів: OpNet 14, NS-3

4. Перелік питань, що потрібно опрацювати у роботі

1) Аналіз сучасного стану проблеми

2) Огляд технологій управління перевантаженням в протоколі TCP

3) Моделі управління мережним трафіком

4) Вибір програмних засобів моделювання

5) Проведення експериментальних досліджень

6) Висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) 23 слайда

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз стану проблеми та сучасних методів її	26.11.24–29.11.24	
	вирішення		
2	Огляд технологій управління перевантаженням	30.11.24–04.12.24	
3	Розробка моделі управління мережним	05.12.24 –16.12.24	
	трафіком		
4	Вибір програмних та апаратних засобів	17.12.24 –19.12.24	
	моделювання		
5	Тестування запропонованого метода	20.12.24–25.12.24	
6	Оформлення пояснювальної записки	26.12.24–-05.01.25	

Дата видачі завдання 25 листопада 2024 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Янковський О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 103 с., 33 рис., 2 табл., 1 дод., 36 джерел.

AQM, ECN, TCP, RTT, UDP, АЛГОРИТМ, ВТРАТА ПАКЕТІВ, ПЕРЕВАНТАЖЕННЯ, ПОВІЛЬНИЙ СТАРТ, ПОВТОРНА ПЕРЕДАЧА, ПРОТОКОЛ, ТАЙМ-АУТ.

Метою кваліфікаційної роботи є поліпшення характеристик функціонування комп’ютерних мереж у сенсі зниження ймовірності виникнення перевантажень в умовах інтенсивного трафіку.

У ході виконання кваліфікаційної роботи представлені нові наскрізні механізми управління перевантаженням в протоколі TCP, включаючи популярний високошвидкісний наскрізний алгоритм на основі втрат.

За допомогою моделювання отримано обнадійливі результати, що означає, для змодельованих мережевих середовищ, можна мати розумні наскрізні алгоритми контролю перевантаження. За допомогою цих методів складність управління перевантаженням розподіляється по мережі, також підтримується сумісність з існуючими версіями протоколу TCP.

ABSTRACT

Master’s thesis: 103 pages, 33 figures, 2 tables, 1 appendices, 36 sources.

AQM, ALGORITHM, ECN, OVERLOAD, PACKET LOSS, PROTOCOL, RETRANSMISSION, RTT, SLOW START, TCP, TIMEOUT, UDP.

The purpose of the qualification work is to improve the characteristics of the functioning of computer networks in the sense of reducing the probability of overloads in conditions of intensive traffic.

In the course of the qualification work, new end-to-end congestion control mechanisms in the TSR protocol are presented, including the popular high-speed end-to-end loss-based algorithm.

The simulations yield encouraging results, which means that intelligent end-to-end congestion control algorithms are possible for simulated network environments. With the help of these methods, the complexity of congestion management is distributed over the network, and compatibility with existing versions of the TCP protocol is also supported.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ACK – підтвердження нового пакету TCP (англ., New TCP packet acknowledgment)

AIMD – адитивне збільшення/мультиплікативне зменшення (англ., Additive increase multiplicative decrease)

DACK – подвійне підтвердження пакета TCP (англ., Duplicate TCP packet acknowledgment)

BDP – продукт затримки смуги пропускання (англ., Bandwidth-Delay Product)

Cwnd – перевантажувальне вікно TCP (англ., TCP congestion window)

IP – Інтернет-протокол (англ., Internet protocol)

MIMD – мультиплікативне збільшення/мультиплікативне зменшення (англ., Multiplicative Increase Multiplicative Decrease)

MSS – максимальний розмір сегменту (англ., Maximum Segment Size)

MTU – максимальний передаваний блок (англ., Maximum Transmission Unit)

PACK – часткове підтвердження (англ., Partial acknowledgment)

RTA – дія тайм-ауту повторної передачі (англ., Retransmission timeout action)

RTO – очікування повторної передачі (англ., Retransmission timeout)

RTT – час подвійного оберту (англ., Round trip time)

Ssthresh – поріг повільного запуску (англ., Slow start threshold)

TCP – протокол керування передачею (англ., Transmission Control Protocol)

UDP – протокол дейтаграм користувача (англ., User Datagram Protocol)

ВСТУП

TCP – це протокол транспортного рівня, який зараз широко використовується в Інтернеті. Це надійний і впорядкований мережевий протокол, орієнтований на підключення.

Згідно зі статистикою, більшість трафіку даних у сучасному Інтернеті передається через TCP.

Таким чином, якість протоколу передачі TCP має великий вплив на загальну продуктивність мережі. Дослідження показують, що перевантаження мережі є однією з основних причин, які впливають на ефективність передачі TCP.

З постійним розвитком соціальної інформатизації через мережу потрібно передавати велику кількість даних у різних сферах суспільства, що робить проблему ефективності передачі мережі та перевантаження мережі все більш актуальною.

У 1988 році Ван Якобсон вказав на недоліки протоколу TCP у контролі перевантажень і запропонував алгоритми «повільного старту», «уникнення перевантажень» і «швидкої повторної передачі».

Тоді також, до алгоритму TCP Reno було додано механізм швидкого відновлення [1] . Розвиваючись протягом багатьох років, TCP розвинувся від оригінального алгоритму TCP Tahoe до поточного широко використовуваного алгоритму TCP CUBIC [2]. Алгоритм контролю перевантажень також був додатково вдосконалений, що також певною мірою покращило продуктивність передачі TCP.

На початку TCP-з’єднання засіб контролю перевантаження спочатку виконує алгоритм повільного старту, щоб запобігти перевантаженню мережі. Основною функцією алгоритму повільного старту є визначення доступної пропускної здатності невідомої мережі на початковому етапі встановлення TCP-з’єднання.

Традиційний стандартний TCP використовує стратегію віконного коригування алгоритму AIMD (Additive Increase, Multiplicative Decrease), завдяки якому дані раптово збільшуються або різко зменшуються під час процесу передачі, а передача є нестабільною. У результаті передача даних у високошвидкісній мережі буде генерувати велике тремтіння затримки [3,4].

Завдяки вдосконаленню початкового вікна перевантаження та режиму зростання аналізу в реальному часі та вікна корекції перевантаження зменшується втрата даних і підвищується ефективність передачі даних мережі.

1 МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Протокол керування передачею TCP відіграє вирішальну роль в інфраструктурі сучасних мереж. Надаючи орієнтовану на з'єднання, надійну службу байтового потоку, TCP дозволив Інтернету процвітати в багатьох типах мереж від високошвидкісних, високонадійних волоконно-оптичних з'єднань до безпровідних каналів, схильних до помилок.

Незважаючи на свою довгу історію, контроль перевантажень TCP залишається гарячою темою донині, оскільки TCP досі залишається єдиним стандартним мережевим протоколом, здатним забезпечити наскрізну надійну передачу та контролювати перевантаження.

Метою магістерської кваліфікаційної роботи є розробка новітніх методів контролю перевантаження алгоритмами протоколу TCP та поліпшення характеристик функціонування комп'ютерних мереж у сенсі зниження ймовірності виникнення перевантажень в умовах інтенсивного трафіку. Необхідно розглянути поточні та запропонувати нові методи, засновані на затримці, з метою вдосконалення існуючих наскрізних алгоритмів контролю перевантажень на основі втрат пакетів.

2 СТЕК ПРОТОКОЛІВ TCP/IP

2.1 Інтернет-протокол

IP – це комунікаційний протокол для передачі та ретрансляції пакетів даних у мережі за допомогою IP-мереж. IP може маршрутизувати пакети через мережі до місця призначення за допомогою своєї адресної схеми. TCP/IP стек – це загальна назва комбінації IP і TCP або UDP. Модель TCP/IP представлений на рисунку 2.1.

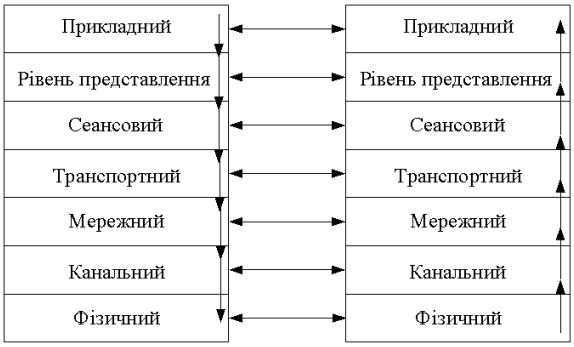

Рисунок 2.1 – TCP/IP модель взаємодії

Сам IP-пакет складається із заголовка та корисного навантаження даних. Заголовок IPv4-пакету показано на рисунку 2.2. Цей рисунок також оновлено з урахуванням найновіших модифікацій у стандартах диференційованих послуг і явного сповіщення про перевантаження.

2.2 Протокол TCP

У травні 1974 року Вінт Серф і Боб Кан опублікували статтю під назвою «Протокол для пакетного мережевого з'єднання» [5]. У цій статті описується протокол комутації пакетів для спільного використання ресурсів

у мережі. У цьому документі програма керування передачею описана як центральний компонент, який був протоколом, орієнтованим на з’єднання, і службою дейтаграм. Програма керування передачею вказана в RFC675 [6].

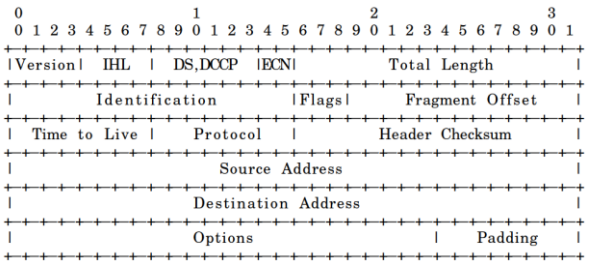

Рисунок 2.2 – Заголовок IP-пакету

Монолітну програму керування передачею пізніше було розділено на TCP та IP, що дало більш модульну архітектуру та називається пакетом протоколів Інтернету. TCP – це транспортний рівень, побудований поверх IP рівня, і тому його часто називають TCP/IP. Там, де IP є службою дейтаграм без з’єднання, TCP розширює IP, забезпечуючи надійну та впорядковану доставку байтового потоку від одного процесу до іншого через мережу.

TCP є наскрізним протоколом, що означає, що взаємодіючі вузли відповідають за правильну передачу трафіку між собою.

Ієрархічна система розрівнювання дає змогу розмістити будь-який протокол прикладного рівня поверх TCP. Це надає додатку функції та надійність доставки, які пропонує TCP.

2.2.1 Встановлення з’єднання TCP

Рукостискання TCP ініціює з’єднання між двома хостами. Комп'ютер, що підключається (A), посилає TCP-сегмент з бітами SYN=1 і ACK=0. Порядковим номером для сегментів є початковий порядковий номер (ISN) (M), який обирає комп’ютер A. SYN-біт повідомляє про спробу створити з’єднання.

Комп’ютер (B) отримує сегмент із увімкненим бітом SYN і відповідає SYN=1 і ACK=1, що підтверджує сегмент SYN. Порядковий номер буде ISN (N) для комп’ютера B, а номер підтвердження – це ISN комп’ютера A плюс один (M+1).

Після цього комп’ютер A завершує рукостискання, підтвердивши ISN, надісланий B, SYN=0 і ACK=1 для цього сегмента, оскільки частина синхронізації завершена. Порядковий номер для цього сегмента M+1, а номер підтвердження N+1.

У цьому процесі рукостискання інша інформація синхронізується між вузлами. Протягом цього періоду відбувається обмін параметрами передачі, часовими мітками та контрольними сумами.

2.2.2 Заголовок TCP

Заголовок TCP містить необхідну інформацію для надсилання даних правильному процесу та підтримки впорядкованості та надійності потоку даних (рисунок 2.3). Номери портів у заголовку повідомляють операційній системі, куди доставляти дані, щоб отримати правильний процес. Оскільки процес прив’язується до певного порту, поле порту з’єднує процес на одержувачі та відправнику з портом джерела та портом призначення.

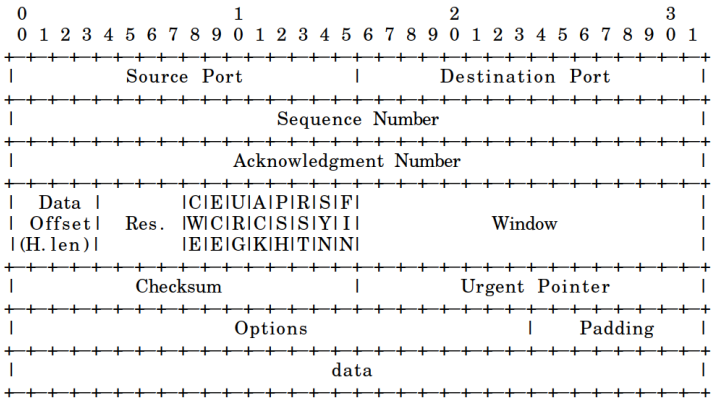

Рисунок 2.3 – TCP-заголовок

Поле опції – це кількість не зарезервованих байтів, яка використовується для сигналізації між комп’ютерами, що зв’язуються через TCP-потік.

Поле параметрів використовується для підтримки «надбудов» до протоколу TCP. В ACKCC поле параметрів використовується для встановлення зв’язку та для оновлення значень співвідношення ACKCC TCP-потоку.

Параметри в полі параметрів мають номери, призначені органом з присвоєння номерів Інтернету (IANA). IANA не має номерів, зарезервованих для ACKCC, але з метою тестування та розробки призначив два значення для використання. Експериментальні значення 253 і 254 використовуватимуться як значення для типу опції в цій реалізації ACKCC.

Перший протокол TCP, визначав лише три номери параметрів:

- кінець параметрів;
- відсутність операції;
- максимальний розмір сегмента.

Використання поля параметрів є популярним для розширення TCP функціями. TCP-заголовок має 4-бітове поле для представлення довжини заголовка. Поле вказує, із скількох 32-розрядних слів складається заголовок. 4 біти можуть представляти лише 16 різних значень від 0 до 15. Максимальна довжина заголовка становить 15*32біти=480біт або 60 байтів. З усіма обов’язковими полями в заголовку заголовок має принаймні 20 байтів без будь-яких параметрів. Тому заголовок може містити 40 байт опцій, які повинні бути доповнені до 32-бітового слова.

У стандартному зв'язку TCP 20 байт використовуються для опцій у пакеті SYN. Мітка часу, максимальний розмір сегмента (MSS), вибіркове підтвердження (SACK), масштаб вікна (WSCALE), відсутність операції (NOP) і кінець рядка (EOL). Для SYN-ACK із загальних 40 доступних байтів використовується ще 4 байти, тобто 24 байти. Оскільки параметри мають бути вирівняні на 32-бітовому слові, поле параметрів доповнюється 0.

Коли приймач зчитує доповнені нулі, він припиняє аналіз параметрів. Доповнені нулі також займають додатковий простір у 24 байтах SYN. Оскільки TCP-заголовок має проблеми з кількістю параметрів, які може містити поле, можна розподілити параметри. Наприклад, у TCP-рукостисканні вже використовується багато місця для опцій, і розвиток TCP вимагатиме надсилання більшої кількості опцій. Тому опцію для ACKCC можна надіслати в першому пакеті даних.

Кожна опція має своє поле типу, що ідентифікує опцію. Тип ідентифікується 8-бітним числом. Довжина в один октет обмежує типи опцій до 255 різних значень. Якщо пропонується багато різних типів опцій, діапазон значень може бути вичерпаний.

Порядковий номер – це 32-розрядне ціле число, розташоване в заголовку TCP-пакета. Номер описує місце розташування пакета в послідовності TCP-потоку. Порядковий номер обчислюється з кількості надісланих байтів, починаючи з ISN.

У мережі з низькою пропускною здатністю порядковий номер не використовуватиметься повторно, оскільки пропускна здатність гарантуватиме, що час для обгортання всього простору порядкового номера буде занадто довгим. Ця проблема може виникнути з більшою пропускною здатністю.

Для 10-мегабітного зв’язку час, необхідний для обгортання порядкового номера, становить приблизно 30 хвилин. Цю проблему вирішує параметр «timestamps», який використовує значення timestamp для усунення неоднозначності порядкових номерів.

Номер підтвердження працює подібно до порядкового номера, але замість збільшення кількості надісланих даних, воно підтверджує отримані дані, надсилаючи назад наступний очікуваний порядковий номер відправнику в ACK-пакеті.

Потім відправник може обчислити, які дані отримав одержувач, і визначити, які пакети він може відправитиабо надіслати повторно.

2.2.3 Трафік TCP

Стандартний TCP-потік складається з пакетів даних у прямому напрямку з відповідним ACK для кожного пакету, отриманого у зворотному напрямку.

Коли отримувач надсилає ACK для кожного другого пакета, він використовує "затримані ACK". Відкладені ACK були реалізовані для збереження ресурсів мережі та хосту.

У перших версіях TCP надсилав ACK для кожного пакета даних. З відкладеними ACK така поведінка спостерігається лише під час події зміни порядку. Затримка ACK обмежена 500 мс, і ACK має надсилатися для другого пакета даних [7].

Контроль перевантаження керує, як контролем перевантаження мережі, так і обмеженнями приймача, що називається керуванням потоком.

З’єднання TCP у будь-який час відстежує як вікно відправника, так і вікно отримувача, яке складається з надісланих і отриманих даних. TCP-з’єднання надсилає ціле вікно сегментів до місця призначення одночасно, замість того, щоб чекати підтвердження для кожного пакета. Це називається протоколом ковзного вікна.

Вікно перевантаження підтримується відправником. Відправник відстежує поточне вікно відправника та вікно отримувача на підключеному хості. Відстежуючи цю інформацію, відправник вибирає найнижче значення вікна перевантаження та вікна отримувача. Відправник зберігає значення розміру вікна перевантаження. На це значення впливає перевантаженість мережі. Під час надсилання пакету даних відправник вибирає найменше значення з вікна перевантаження та вікна одержувача. Таким чином відправник не буде насичувати одержувача через контроль потоку.

Вікно перевантаження snd_cwnd постійно оновлюється за допомогою зворотного зв’язку з мережі. snd_cwnd намагається наблизити себе до поточного продукту затримки пропускної здатності BDP.

Відправник керує трьома змінними: snd_wnd, rcv_wnd і snd_cwnd. snd_wnd, як описано вище, встановлюється відповідно до двох інших. Обсяг даних, що передаються через мережу, обмежується параметром snd_wnd. Якщо значення snd_wnd завжди буде менше, ніж добуток затримки пропускної здатності, це забезпечить відсутність перевантаження в мережі.

Коли відправник перебуває в режимі повільного старту, вікно перевантаження (cwnd) збільшується для кожного ACK, починаючи з початкового вікна (зазвичай) 1-4 пакетів до порогового значення повільного старту. Використовується на початку підключення. Якщо протягом певного часу не надходить ACK, виникає тайм-аут, і контроль перевантаження TCP також переходить у цю фазу.

Уникнення перевантаження – це алгоритм, який зменшує вікно відправника, коли виникає втрата пакету. Вікно відправника зменшується вдвічі, а потім збільшується на один раз за час RTT.

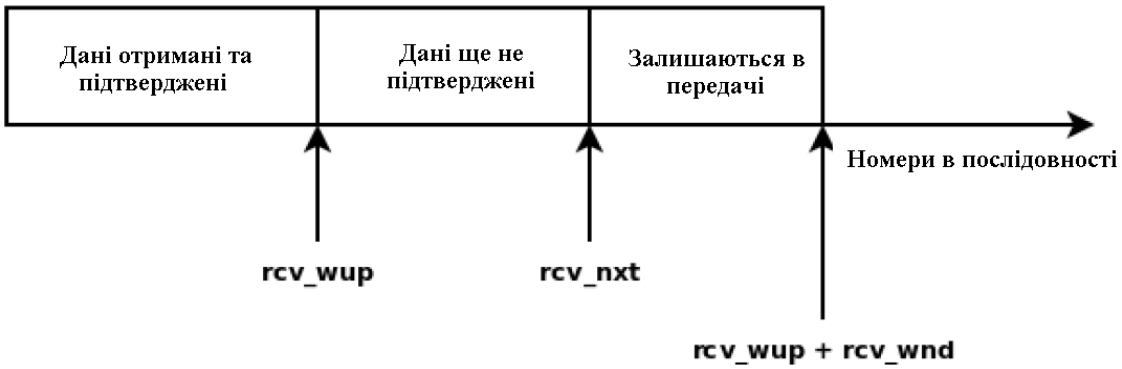

Рисунок 2.4 – Вікно на стороні адресата

Швидка повторна передача ініціюється, коли до відправника надходять три пакети DUPACK, що свідчить про втрату пакету. Цей алгоритм повторно передає пакет до закінчення часу очікування його повторної передачі. Коли пакет повторно передається, відправник переходить у фазу уникнення перевантажень, яка зменшує вікно вдвічі та встановлює це значення як новий поріг повільного старту.

Явне сповіщення про перевантаження (ECN) було представлено в RFC3168 [8] у 2001 році. ECN представляє для мережі метод сповіщення відправника даних про те, що мережа перевантажена, без відкидання пакетів. ECN – це наскрізне сповіщення, яке має бути реалізоване у відправнику, одержувачі та маршрутизаторах на мережевому шляху.

Щоб мати можливість надсилати дані про стан, ECN перевизначає деякі поля заголовка, які використовуються диференційованими послугами (DS) в IP, і використовує деякі зарезервовані біти в заголовку TCP.

ECN не використовується широко в Інтернет. У версіях Windows до Windows 7 ECN вимкнено за замовчуванням. У Linux ECN використовується, якщо інша сторона запитує про це.

Черга випадкового раннього виявлення (RED), яка може використовуватися в маршрутизаторах, використовує ECN. Замість скидання випадкового пакета, щоб повідомити відправника про необхідність уповільнити швидкість надсилання, пакет позначається. Наступний вихідний ACK буде позначений одержувачем, сигналізуючи відправника про поточний стан мережі. TCP реалізує лише контроль перевантаження пакетів даних. Оскільки TCP не контролює перевантаження керуючих пакетів (чисті ACK, SYN, FIN-сегменти), контрольні пакети зазвичай не позначаються як сумісні з ECN. Перевантаження на ACK-шляху]. Оскільки немає механізму для адаптації потоку керуючих пакетів, немає жодних переваг для їх маркування.

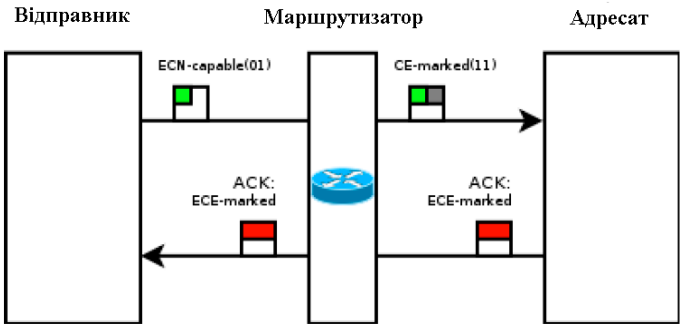

Рисунок 2.5 – Пакет з позначкою ECN

3 ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

3.1 Виникнення проблеми перевантаження

За останні двадцять років Інтернет суттєво зріс. Сьогодні він об’єднує мільйони людей і комп’ютерів та вплинув на численні аспекти нашого життя. Споживчі мережі, такі як житлові волоконно-оптичні мережі та хмарні технології розширюють межі того, що вважалося неможливим лише кілька років тому. Наприклад, такі послуги з постійно зростаючими вимогами до пропускної здатності, як-от відео високої чіткості на вимогу, онлайн-музика та хмарні служби зберігання даних, швидко змінюють аспекти використання Інтернету.

Високошвидкісні трансатлантичні мережі дозволяють лабораторіям, таким як CERN і Fermilab, щодня обмінюватися величезними обсягами експериментальних даних із науковими дослідницькими установами по всьому світу.

Протокол керування передачею TCP [10] відіграє вирішальну роль в інфраструктурі Інтернету. Надаючи орієнтовану на з’єднання, надійну службу байтового потоку, TCP дозволив Інтернету процвітати в багатьох типах мереж; від високошвидкісних, високонадійних волоконно-оптичних з’єднань до безпровідних каналів, схильних до помилок. Крім того, TCP забезпечує фундаментальний сервіс для забезпечення стабільності Інтернету, який називається контролем перевантажень.

Контроль перевантаження є ключовим питанням у будь-якій мережі. Інтернет, який складається з багатьох різнорідних мереж із різними можливостями, особливо сприйнятливий до перевантажень. Маршрутизатори повинні буферизувати пакети в чергах до і після обробки. Якщо швидкість надходження пакетів перевищує пропускну здатність мережі, маршрутизатор повинен зберігати їх у черзі, доки не зможе переслати пакети до наступного

переходу. Зрештою, якщо швидкість надходження продовжує перевищувати швидкість відправлення, черга зростає до такого розміру, що маршрутизатор не має іншого вибору, окрім як відкидати пакети, порушуючи мережеві потоки. Таким чином, із збільшенням коефіцієнта втрати пакетів пропускна здатність зменшується.

Рішення полягає в тому, щоб зменшити зайнятість черги маршрутизатора та, як наслідок, рівень втрати пакетів. Щоб досягти цього, джерела повинні знизити швидкість надсилання.

Очікується, що TCP, будучи надійним транспортним протоколом, повторно передаватиме втрачені сегменти, але його початкова поведінка повторної передачі є надто агресивною, ставлячи мережу під загрозу збою перевантаження. Перевантаження відбувається, коли сильно перевантажена мережа досягає стійкого стану, де фактична корисна пропускна здатність дуже низька.

У жовтні 1986 року Інтернет зазнав свого першого колапсу перевантаження. Ці події захопили дослідників, спонукавши їх дослідити основні причини збоїв [11]. Було показано, що протоколу TCP бракує фундаментальних механізмів контролю, щоб запобігти створенню перевантажень. Було запропоновано декілька алгоритмів разом із критичними вдосконаленнями таймерів TCP.

Алгоритми контролю перевантажень TCP були успішними з моменту їх створення в 1988 році і дозволили масштабувати Інтернет багаторазово. Однак поширення мереж нового покоління, таких як з’єднання з продуктами з великою затримкою пропускної здатності або безпровідними мережами, зробило обмеження традиційного контролю перевантажень TCP більш очевидними.

Незважаючи на свою довгу історію, контроль перевантажень TCP залишається гарячою темою донині, оскільки TCP досі залишається єдиним стандартним протоколом Інтернету, здатним забезпечити наскрізну надійну передачу та контролювати перевантаження.

3.2 Поточні проблеми TCP

Основна проблема, з якою зараз стикається TCP – це ефективність з’єднання. Ефективність зв'язку означає, що протокол TCP повинен мати можливість повністю використовувати пропускну здатність мережі. Фактично, багато останніх внесків у контроль перевантажень TCP є побічним ефектом необхідності покращити продуктивність TCP на з’єднаннях із високим BDP (Bandwidth Delay Product), тобто максимальним об’ємом даних у мережі в будь-який момент часу – даних, які були передані, але ще не підтверджені.

Щоб отримати уявлення про цю проблему, потрібно зрозуміти динаміку, пов’язану з керуванням вікном перевантаження. TCP використовує механізм ковзного вікна як форму керування потоком. Це вікно називається вікном перевантаження. Відправник TCP регулює розмір вікна, коли надходить підтвердження від одержувача або коли в мережі буде виявлено перевантаження.

Час між надсиланням сегмента та отриманням його підтвердження називається часом зворотного проходження (RTT). Таким чином, швидкість відправника TCP (тобто швидкість, з якою він може адаптувати своє вікно) залежить від його RTT.

Ця тенденція до потоків із меншими RTT створює різкі відмінності у зростанні вікна TCP. Наприклад, відправник із 10 мс RTT може збільшити розмір вікна перевантаження до 100 сегментів лише за 1 секунду; з іншого боку, для відправника із 100 мс RTT це займе 10 секунд.

Досягнення фізичних мережевих технологій зробили міжміські мультигігабітні мережі звичним явищем. Однак, хоча пропускна здатність мережі постійно зростає, затримка залишається постійною. Крім того, відправник TCP регулює своє вікно відповідно до алгоритму адитивного збільшення мультиплікативного зменшення (AIMD) із досить консервативними параметрами.

Ця конструкція, хоч і безпечна, призводить до дуже поганого використання каналу зв’язку в мережах із великим BDP и з великою пропускною здатністю.

Цей стан погіршується, оскільки добуток пропускної здатності на затримку зв’язку збільшується. Для ілюстрації розглянемо зв’язок із доступною пропускною здатністю 5 Гбіт/с, MTU 1500 і 100 мс RTT. TCP знадобиться вікно з приблизно 42 000 сегментів, щоб повноцінно використовувати канал.

Після виявлення першої події перевантаження вікно перевантаження буде зменшено до 21 000 сегментів. Якщо припустити, що не відбудеться жодної іншої події втрати, що є малоймовірним через динаміку самого каналу, наприклад інші потоки TCP, що конкурують за пропускну здатність і частоту помилок каналу, TCP знадобиться 35 хвилин, щоб досягти повного використання каналу.

Підсумовуючи, можна сказати, що сучасний TCP невідповідний для високошвидкісних мереж із великою затримкою.

Справедливість також є важливим питанням. Розглядаємо два різних типи справедливості. Справедливість TCP, тобто те, наскільки чесною буде нова пропозиція TCP щодо стандартного TCP, і чесність внутрішнього протоколу, яка пов’язана зі справедливим розподілом пропускної здатності між потоками, на яких працює одна версія TCP. Що стосується справедливості застарілого протоколу TCP, розподіл пропускної здатності для потоків, що використовують варіанти TCP, не повинен домінувати над стандартними потоками TCP, щоб забезпечити менш руйнівне розгортання в мережі.

Що стосується внутрішньопротокольної справедливості, будь-який алгоритм TCP, який має на меті забезпечити ефективне використання каналів у високошвидкісних мережах, повинен з часом досягти справедливого розподілу пропускної здатності між конкуруючими потоками. Крім того, алгоритм повинен враховувати різні аспекти, які можуть вплинути на

поведінку окремого потоку, і, отже, на його загальну справедливість. Наприклад, було помічено, що такі властивості потоку, як час проходження туди й назад, час початку надходження потоку та агресивність фактора збільшення, можуть серйозно погіршити справедливість багатьох нових потоків TCP [12].

Нарешті, перехідний період, коли розподіл пропускної здатності між потоками є несправедливим, має бути якомога коротшим без шкоди для стабільності та ефективності алгоритму.

За останні кілька років безпровідні мережі стали дуже популярними. З появою мобільних комп'ютерів кількість пристроїв, які використовують TCP через безпровідні з'єднання, різко зросла.

Зараз дуже поширені мобільні телефони з доступом до Інтернету, безпровідний широкосмуговий Інтернет і WLAN у житлових мережах. Вони служать свідченням гнучкості TCP для адаптації до нових фізичних носіїв.

Однак важливо відзначити, що TCP спочатку був розроблений з урахуванням провідних мереж, де частота помилок пакетів мала, а характеристики мережі досить постійні. Це, звичайно, контрастує з реальністю безпровідних мереж.

У безпровідних мережах частота помилок може різко зрости через перешкоди або конфлікти кадрів, спричинені тим, що кілька відправників намагаються передавати дані одночасно. Доступна смуга пропускання може раптово впасти через погіршення сигналу. TCP розроблено для інтерпретації втрати пакетів як ознаки перевантаження. Отже, у безпровідній мережі, коли TCP виявляє, що сегмент втрачено, він часто й помилково припускає, що це сталося через перевантаження. У відповідь на подію втрати TCP зменшить своє вікно перевантаження вдвічі, без потреби вплинувши на пропускну здатність.

Це, мабуть, найскладніша проблема TCP, оскільки він не має засобів для розрізнення втрат, спричинених перевантаженням мережі, і втрат, спричинених помилками зв'язку.

Обмеження в контролі перевантажень TCP спонукали дослідницьке співтовариство запропонувати нові рішення. Було запропоновано багато внесків, але всі вони можуть бути оформлені в одному з двох підходів: наскрізний контроль перевантаження та контроль перевантаження за допомогою маршрутизатора.

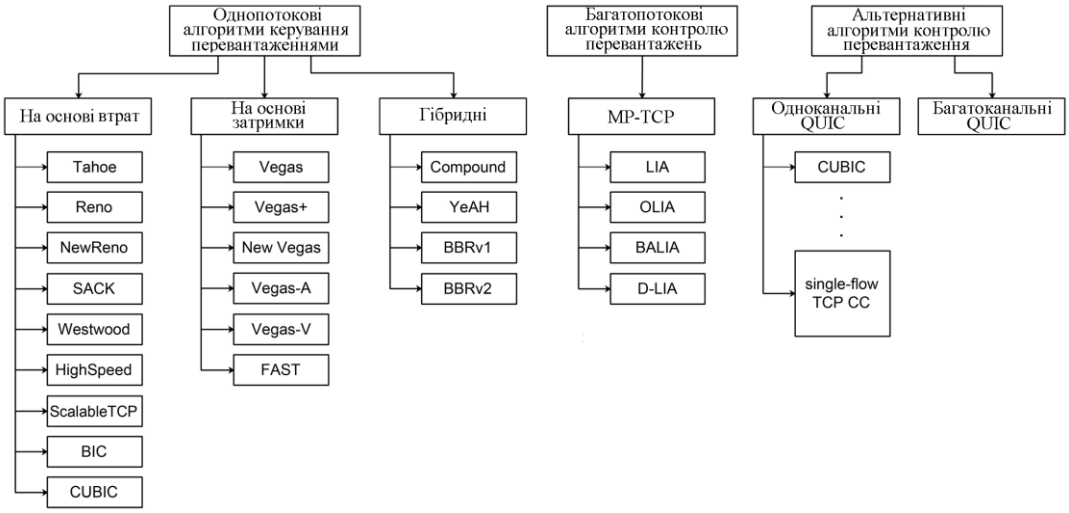

Рисунок 3.1 – Основні категорії та типи алгоритмів TCP

Наскрізна стратегія – це класичний підхід до контролю перевантажень. У цій моделі відповідальністю TCP-відправника є виявлення перевантаження та реагування на нього. Наскрізні алгоритми мають перспективу чорної скриньки мережі. Таким чином, усі наскрізні механізми перевантаження можуть покладатися лише на неявні сигнали перевантаження, тобто втрати пакетів і зміни затримки. Цей підхід виявився успішним протягом багатьох років. Тим не менш, він має деякі обмеження. Схеми, засновані на втратах, спричиняють перевантаження мережі, оскільки це єдиний спосіб перевірити доступну пропускну здатність, що робить їх реактивними, а не проактивними.

Схеми, засновані на затримці, можуть проактивно реагувати на перевантаження мережі, і, таким чином, є більш дружніми до мережі в тому сенсі, що вони можуть підтримувати низькі показники втрати пакетів і

низьку затримку в черзі. Перевіряючи коливання виміряних значень RTT, вони можуть зробити висновок про зайнятість черги маршрутизатора, але стикаються з багатьма проблемами з точною оцінкою RTT і недостатнім використанням каналу через наявність алгоритмів, що базуються на втратах.

Крім того, ще належить визначити, чи можуть алгоритми, засновані на чистій затримці, правильно працювати в багатьох мережевих сценаріях. Зовсім недавно гібридні (тобто такі, що поєднують втрати та затримку) підходи пропонуються як можливі рішення деяких внутрішніх проблем із механізмами, заснованими на чистих втратах і чистих затримках. Гібридні алгоритми демонструють покращені результати в деяких сценаріях, але вони все ще зберігають більшість проблем, пов'язаних з алгоритмами на основі втрат і затримок, оскільки вони все ще мають обмежену інформацію про справжній стан мережі. Незважаючи на свої обмеження, прихильники наскрізного підходу стверджують, що лише наскрізний алгоритм дотримується «золотого принципу», що складність мережі має бути на її межі, а не на її ядрі [13], дозволяючи масштабувати мережу. Ще один важливий аргумент полягає в тому, що наскрізний підхід дозволяє поступове розгортання таких алгоритмів. Дуже привабливою характеристикою наскрізного алгоритму контролю перевантажень є те, що потрібно модифікувати лише сторону відправника.

У той же час пропонується інша стратегія контролю заторів. Підхід із підтримкою маршрутизаторів робить маршрутизатори активним компонентом архітектури контролю перевантаження.

Однією з ранніх пропозицій є випадкове раннє виявлення (RED) [14] разом із явним сповіщенням про перевантаження (ECN) [15]. RED має на меті пом'якшити деякі проблеми з відкиданням пакетів у чергах маршрутизаторів.

У хвостових чергах відкидання пакети відкидаються у маршрутизаторі з вузьким місцем після того, як відбувається переповнення буфера черги. У чергах, керованих RED, пакети відкидаються раніше на основі результатів

алгоритму ймовірності відкидання. Якщо зайнятість черги низька, пакети майже не відкидаються; навпаки, якщо черга почне заповнюватися, ймовірність випадання зросте пропорційно.

Цей підхід показав хороші результати в контролі відкидання пакетів, затримки в черзі та поведінки блокування. Однак відправники TCP все ще виявляють перевантаження мережі, спостерігаючи за втратою пакетів, і успадковують усі проблеми, пов’язані з неявною сигналізацією про перевантаження. Таким чином, було запропоновано ECN. ECN використовує вихідні дані функції ймовірності RED, але замість того, щоб скинути пакет, він явно сповіщає відправника TCP, дозволяючи йому адаптуватися до перевантаження ще до його виникнення та уникнути непотрібних повторних передач.

Також були запропоновані більш складні алгоритми для контролю перевантажень, які називаються алгоритмами зворотного зв’язку з явною швидкістю. Прикладами таких пропозицій є XCP [16]. По суті, алгоритми зворотного зв’язку з явною швидкістю дозволяють маршрутизаторам надавати джерелам інформацію про те, скільки даних вони повинні надсилати в певний момент часу.

Таким чином, вони демонструють переваги перед наскрізними аналогами, вирішуючи більшість їхніх проблем. Це включає в себе ефективне використання в каналах з високою пропускною здатністю із затримкою, справедливість потоку та неоднозначність втрати сегментів у гетерогенних мережах.

Основні недоліки полягають у тому, що, окрім джерел, усі маршрутизатори в мережі повинні брати участь у протоколі, і що потрібні потужніші маршрутизатори, щоб впоратися зі збільшеною складністю таких алгоритмів.

Підхід із підтримкою маршрутизатора, незважаючи на те, що він є кращим, все ще обговорюється, оскільки його впровадження в поточних мережах є дуже складним як через технічні, так і політичні аспекти. Цей та

інші фактори поки що перешкоджають прийняттю підходу за допомогою маршрутизаторів. Алгоритм наскрізного контролю перевантаження TCP є найпростішою та найбільш масштабованою формою контролю перевантаження в Інтернеті. За винятком відправника і, можливо, одержувача, наскрізний підхід до контролю перевантаження розглядає всі компоненти архітектури Інтернету як чорні ящики. Перевантаженість визначається лише двома неявними сигналами: втратою пакетів і змінами затримки. Втрати пакетів виявляються за допомогою таймерів TCP і дублікатів підтвердження. Затримка пакетів виявляється за допомогою вимірювань RTT і аналізу потоку підтверджень.

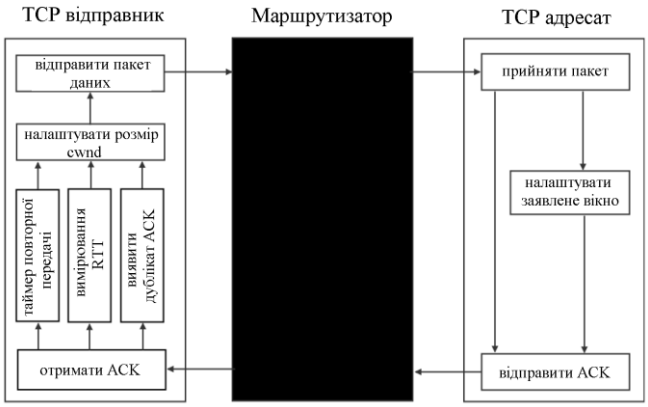

Рисунок 3.2 – Архітектура наскрізного контролю перевантажень TCP

Відповідно до цих обмежень можна розробити нові наскрізні алгоритми контролю перевантажень. На рисунку 3.2 проілюстровано архітектуру наскрізного контролю перевантаження. Слід зауважити, що деякі компоненти є необов’язковими, наприклад, в алгоритмі на основі втрат компонент вимірювання RTT не впливає на розмір вікна перевантаження.

4 КОНТРОЛЬ ПЕРЕВАНТАЖЕННЯ В ПРОТОКОЛІ TCP

Наскрізні алгоритми пропонують привабливий підхід до контролю мережевих перевантажень, як простотою, так і масштабованістю. Єдиними сигналами перевантаження мережі, доступними для наскрізного алгоритму, є втрати пакетів і варіації затримки. Тому аналіз був зосереджений на трьох типах алгоритмів:

- алгоритми на основі втрат покладаються лише на втрату пакетів, щоб реагувати на перевантаження мережі;
- алгоритми на основі затримки використовують лише вимірювання затримки, щоб зробити висновок про зайнятість черги маршрутизатора та діяти до того, як виникне сильна перевантаженість;
- гібридні алгоритми використовують методи як алгоритмів, що базуються на втратах, так і на основі затримки (їх обґрунтування полягає в тому, що з більшою кількістю інформації наскрізний алгоритм може точніше визначати стан мережі та приймати кращі рішення).

4.1 Алгоритми контролю перевантаження на основі втрат

Традиційно протокол TCP використовує втрату пакетів для виявлення перевантаження мережі. Зараз багато нових алгоритмів використовують інформацію про затримку, але алгоритми, засновані на чистих втратах, досі домінують в мережах.

4.1.1 TCP Tahoe

Tahoe був першою версією TCP, розробленою для контролю перевантаження [11]. Один з основних принципів його роботи полягає в тому, що швидкість нових пакетів, що надсилаються в мережу, повинна бути

близькою до швидкості, з якою одержувач повертає підтвердження. У версіях, раніших за Tahoe, джерела надсилають кілька сегментів у мережу, доки заявлене вікно приймача не заповниться. Ця надто агресивна поведінка може спричинити проблеми, коли між відправником і одержувачем є повільніші канали. Після досягнення вихідного маршрутизатора, підключеного до каналу з меншою пропускною здатністю, пакет повинен бути буферизований і чекати передавання.

Оскільки буферний простір обмежений, можливо, що маршрутизатор переповнить свою чергу, і кілька пакетів буде відкинуто. У [11] показано, як такий підхід може серйозно вплинути на пропускну здатність TCP-з'єднань.

Tahoe містить три алгоритми. Це повільний старт, уникнення перевантаження та швидка повторна передача. Tahoe також додає нове вікно до TCP відправника, яке називається вікном перевантаження.

TCP-відправник не повинен передавати більше, ніж мінімум вікна перевантаження та оголошеного вікна приймача, тобто.

$$\text{window} = \min(\text{cwnd}, \text{rwnd}), \tag{4.1}$$

де cwnd – вікно перевантаження, а rwnd – заявлене вікно приймача.

Коли встановлюється нове з'єднання з іншим хостом, вікно перевантаження встановлюється розміром в один сегмент. Кожного разу, коли отримується ACK, вікно перевантаження відправника збільшується на один сегмент, що призводить до експоненціального збільшення вікна. Це повільний старт. Через деякий час, якщо пропускна здатність мережі буде досягнута, проміжний маршрутизатор почне скидати пакети. Tahoe виявляє втрату пакетів за допомогою двох різних механізмів: тайм-ауту повторної передачі (RTO) і надходження трьох дублікатів підтвердження.

Якщо втрату пакета виявлено через три дублікати ACK, Tahoe переходить у фазу швидкої повторної передачі, де (імовірно) втрачений пакет надсилається повторно.

Згодом активується один із двох механізмів втрати. Уцьому випадку, відправник розуміє, що його вікно перевантаження стало занадто великим, і реагує відповідно.

Одним із фундаментальних припущень TCP Tahoe є те, що швидкість втрат пакетів дуже мала, тому втрату пакетів можна вважати сильним показником того, що в мережі виникла перевантаженість, і відправнику слід зменшити швидкість надсилання.

Після виявлення перевантаження в мережі поточне вікно перевантаження зменшується вдвічі та зберігається в змінній, що називається пороговим значенням повільного запуску (ssthresh). Відправник встановлює своє вікно перевантаження на один розмір сегмента та починає повільний запуск, експоненціально збільшуючи вікно.

Після того, як вікно перевантаження відправника досягає ssthresh, cwnd збільшується приблизно на один сегмент на RTT (1/cwnd кожного разу, коли отримано підтвердження). Це уникнення перевантаження. Алгоритм уникнення перевантаження намагається збільшити вікно перевантаження більш консервативним способом, ніж повільний запуск. Це призводить до адитивного збільшення на відміну від експоненціального збільшення при повільному запуску.

Tahoe, навіть з його обмеженнями, став великим проривом у контролі заторів. Він відіграв важливу роль у запобіганні застійним колапсам в Інтернеті та встановив керівні принципи, які практично кожна нова реалізація TCP повинна поважати або, принаймні, враховувати.

4.1.2 Reno

Reno [17] зберігає більшість характеристик Tahoe, але вносить дві зміни, які значно покращують продуктивність Tahoe. Reno змінює те, як Tahoe реагує на виявлення трьох повторюваних підтверджень і як він відновлюється після цього.

Ключова ідея полягає в тому, що Reno не повинен повертатися до повільного запуску, коли втрату пакетів виявляється через дубльовані ACK. Натомість він має продовжувати надсилати дані, хоча й повільніше. Основне розуміння полягає в тому, що оскільки відправник все ще отримує ACK, мережа все ще доставляє пакети, тому перевантаження ще не є сильним. Таким чином, він повинен мати можливість скористатися цією ситуацією та діяти обережно, але не повинен припинити надсилання пакетів взагалі. Таким чином, Reno представив новий механізм під назвою швидкого відновлення.

Отримавши три копії ACK, Reno активує швидку повторну передачу та негайно повторно надсилає втрачений сегмент. Така поведінка ідентична Tahoe. Однак після швидкої повторної передачі Reno активує швидке відновлення. Під час швидкого відновлення значення половини вікна перевантаження зберігається в ssthresh, а нове значення вікна перевантаження встановлюється як ssthresh плюс потрійний розмір сегмента. Це безпечно, тому що три дублікати ACK означають, що три пакети покинули мережу, і дотримується принцип збереження пакетів [11].

Кожного разу, коли надходить дублікат ACK, вікно перевантаження збільшується на розмір сегмента (знову ж таки, це означає, що інший пакет покинув мережу). Якщо нове значення вікна перевантаження це дозволяє, відправник передає пакет. Алгоритм виконується, доки не прийде новий ACK, який підтверджує отримання нових даних.

Коли швидке відновлення завершується, вікно перевантаження встановлюється на ssthresh, і Reno відновляє звичайне уникнення перевантаження. Можлива еволюція вікна перевантаження під TCP Reno показана на рисунку 4.1. На цьому рисунку показано лише фази повільного запуску та уникнення перевантаження.

Ефекти алгоритмів швидкої повторної передачі та швидкого відновлення у вікні перевантаження не відображаються. Крім того, якщо в будь-який час, якщо таймер повторної передачі закінчиться, Reno перейде у повільний старт, точно як Tahoe.

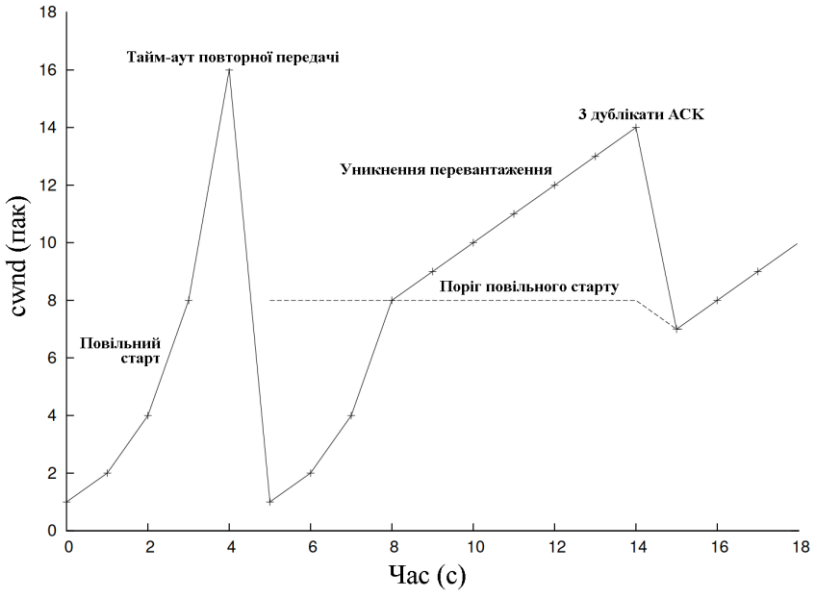

Рисунок 4.1 – Вікно перевантаження TCP Reno

Reno є значним покращенням порівняно з Tahoe щодо втрати одного пакета у вікні. Однак, як і Tahoe, він погано поводиться, коли в одному вікні даних відбувається втрата кількох пакетів.

NewReno пропонується в [18] для покращення алгоритмів швидкої повторної передачі та швидкого відновлення Reno.

NewReno припускає, що з вікна даних втрачається лише один сегмент. Однак пакети часто надходять на маршрутизатори пакетами, тому відкидання пакетів також відбуватиметься пакетно. Це поширене явище на маршрутизаторах із чергами відкидання. Отже, коли черга вузького місця маршрутизатора перевантажується, часто з вікна даних втрачається більше ніж один сегмент. За цього параметра Reno не справлятиметься належним чином із відкиданням кількох пакетів.

Фундаментальна проблема Reno, яка робить його непридатним для обробки скидання кількох пакетів, полягає в тому, що кожного разу, коли ACK підтверджує надходження нових даних, він залишає швидке відновлення. У сценарії, коли відкидається кілька пакетів, Reno кілька разів викличе швидку повторну передачу та швидке відновлення. Кожного разу,

коли це відбувається, вікно перевантаження та поріг повільного старту будуть ще більше зменшуватися. Це призводить до низької пропускної здатності та іноді може викликати RTO через брак ACK, змушуючи відправника повернутися до повільного запуску.

Нова ідея в NewReno полягає в запровадженні часткових підтвержень. Часткове підтвердження підтверджує лише деякі сегменти до того, як відправник виявив втрату пакета. У Reno перше часткове підтвердження призводе до припинення швидкого відновлення. З іншого боку, NewReno використовує часткове підтвердження як сигнал про втрату іншого пакета та реагує відповідно.

Після виявлення трьох дублікатів підтвердження NewReno зберігає порядковий номер останнього переданого сегмента у змінній під назвою recover. Далі йде поведінка, майже ідентична до Рено. NewReno вмикає швидку повторну передачу, надсилає втрачений пакет і запускає швидке відновлення. Однак, коли до відправника надходить часткове підтвердження, NewReno повторно передає сегмент, який відповідає частковому підтвердження, але продовжує швидке відновлення.

NewReno досі залишається одним із найпопулярніших варіантів TCP. Разом із Reno та SACK [19], який розширює можливості TCP для вибіркового підтвердження окремих сегментів і, отже, покращує його здатність відновлювати втрату багатьох сегментів, зазвичай розглядається як фактична версія TCP.

4.2 Високошвидкісні алгоритми контролю перевантажень TCP

Загальновідомо, що консервативна поведінка AIMD TCP погано масштабується в мережах із великим BDP. Однією з можливих стратегій підвищення ефективності з’єднання є зробити TCP більш агресивним. Тобто швидше збільшувати вікно перевантаження, а в разі втрати пакетів використовувати менший коефіцієнт зменшення.

Кілька нових високошвидкісних варіантів TCP були запропоновані як прості зміни в механізмі AIMD TCP. Багато високошвидкісних варіантів TCP мають два різних режими роботи. У високошвидкісному режимі для досягнення кращої ефективності використовується більш агресивна поведінка. У режимі сумісності стандартна поведінка TCP копіюється, щоб підтримувати справедливість із застарілими потоками TCP. Перехід між високошвидкісним і застарілим режимами зазвичай контролюється розміром вікна перевантаження.

4.2.1 HighSpeed TCP

HighSpeed TCP [20] модифікує механізм TCP AIMD, щоб зробити його більш придатним для високошвидкісних мереж. Високошвидкісний TCP розроблено для динамічної поведінки, що залежить від стану мережі. За умов низької втрати пакетів HighSpeed TCP поводиться агресивніше, ніж звичайний TCP, щоб досягти високого рівня використання мережі. Однак, коли коефіцієнт втрати пакетів перевищує 10^{-3}, поведінка HighSpeed TCP ідентична поведінці звичайного TCP. Таким чином, високошвидкісний TCP може підтримувати справедливість до звичайного TCP і не збільшує ризик збою перевантаження в мережах із середньою або високою швидкістю втрати пакетів. Крім того, якщо поточне вікно перевантаження менше попередньо визначеного порогу, Low_Window, тоді HighSpeed TCP повертається до звичайної поведінки TCP. Функція відповіді HighSpeed TCP визначається новими параметрами адитивного збільшення та мультиплікативного зменшення, α(cwnd) і β(cwnd), де cwnd – вікно перевантаження. На етапі уникнення перевантаження зростання вікна перевантаження визначається наступними рівняннями 4.2 та 4.3. Коли отримано ACK:

$$\text{cwnd} \leftarrow \text{cwnd} + \frac{\alpha(\text{cwnd})}{\text{cwnd}}. \qquad (4.2)$$

Коли зафіксовано перевантаження:

$$\text{cwnd} \leftarrow \text{cwnd} - \beta(\text{cwnd}) \cdot \text{cwnd}. \quad (4.3)$$

Якщо значення cwnd менше, ніж Low_Window, HighSpeed TCP поводиться точно так само, як TCP, тому α(cwnd)=1 і β(cwnd)=0,5. І навпаки, коли cwnd більше ніж Low_Window, α(cwnd) стає більшим, а β(cwnd) стає меншим із збільшенням cwnd.

Така поведінка дозволяє HighSpeed TCP збільшити вікно перевантаження та відновлюватися від втрат швидше, ніж TCP, що робить його більш придатним для високошвидкісних з'єднань.

На рисунку 4.2 показано еволюцію вікна перевантаження під HighSpeed TCP.

Слід звернути увагу на більш агресивну поведінку збільшення вікна перевантаження HighSpeed TCP. Крім того, коли відбувається скидання пакетів, вікно перевантаження HighSpeed TCP зменшується приблизно з 1300 пакетів до приблизно 950 пакетів замість 650 пакетів, як можна було б очікувати з NewReno.

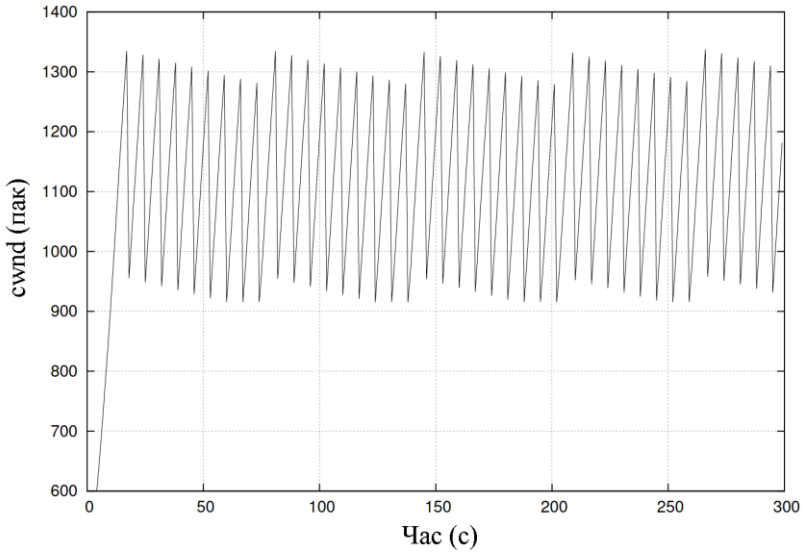

Рисунок 4.2 – Вікно перевантаження HighSpeed TCP

4.2.2 Scalable TCP

Scalable TCP [21] базується на роботі HighSpeed TCP. Його вікно перевантаження слідує за мультиплікативним збільшенням і мультиплікативним зменшенням (MIMD). Подібно до високошвидкісного TCP, вікно Scalable TCP зростає швидше після досягнення певного порогу. Проте, на відміну від HighSpeed TCP, Scalable TCP використовує статичні параметри збільшення та зменшення. Його високошвидкісне зростання вікна перевантаження визначається рівняннями 4.4 та 4.5.

Коли надходить АСК:

$$\text{cwnd} \leftarrow \text{cwnd} + 0.01. \tag{4.4}$$

Коли зафіксовано перевантаження:

$$\text{cwnd} \leftarrow \text{cwnd} - [0.125 \cdot \text{cwnd}]. \tag{4.5}$$

Високошвидкісний і масштабований TCP можуть як досягти гарної масштабованості, так і використання каналу. Однак вони викликають деякі проблеми щодо справедливості, коли потоки з різними RTT конкурують за пропускну здатність мережі. Недоліком обох протоколів є те, що їх більш агресивний характер робить їх більш схильними до несправедливості RTT.

4.2.3 H-TCP

H-TCP [22] прагне підтримувати сумісність із уже розгорнутими версіями TCP. Він базується на принципах високошвидкісного TCP, але вводить кілька ідей для зменшення несправедливості RTT, від якої він страждає. Як і HighSpeed TCP, він регулює свої параметри α і β, щоб краще використовувати доступну пропускну здатність мережі. Однак його

унікальна інноваційна ідея полягає в тому, щоб зробити функцію збільшення вікна перевантаження залежною від часу з моменту відкидання останнього пакета замість традиційного методу збільшення вікна перевантаження кожного RTT. Зробивши зростання вікна перевантаження незалежним від RTT, H-TCP значно покращує пропускну здатність і справедливість потоків із різнорідними RTT.

4.2.4 TCP-Westwood

TCP-Westwood запропоновано в [23]. Ключове розуміння полягає в тому, щоб використовувати швидкість надходження підтверджень для визначення поточної пропускної здатності мережі. На відміну від стандартного TCP, який удвічі скорочує вікно перевантаження при втраті пакетів, TCP-Westwood використовує оцінку пропускної здатності, щоб встановити вікно перевантаження та порогове значення повільного запуску на більш відповідні значення. Завдяки цьому TCP-Westwood працює значно краще, ніж Reno, у гетерогенних мережах.

Було запропоновано новий варіант під назвою TCP-Westwood з гнучким зондуванням і постійним виявленням незавантаженості (TCPW-A) [24]. TCPW-A може досягти високої ефективності без шкоди для справедливості та стабільності. Цікавою характеристикою є те, що вона зберігає властивості оригінального Westwood, тобто хорошу продуктивність у гетерогенних мережах. Таким чином, він може демонструвати кращу поведінку порівняно з іншими високошвидкісними варіантами TCP.

4.2.5 TCP BIC

Інший алгоритм під назвою Binary Increase Congestion (BIC) [25] використовує новий механізм збільшення вікна перевантаження, який розглядає контроль перевантаження як проблему пошуку.

Основна ідея алгоритму полягає в наступному. Мінімальне вікно визначається як вікно перевантаження, коли втрата пакетів ще не виявлена. Цільове вікно визначається як середина між мінімальним вікном (тобто поточним вікном перевантаження, за умови, що втрати ще не відбулися) і максимальним вікном.

Спочатку максимальне вікно є великою попередньо визначеною константою, інакше це вікно, у якому раніше було виявлено втрату пакетів. Якщо поточне вікно перевантаження далеко від цільового вікна, BIC збільшує вікно перевантаження швидше. Це називається фазою адитивного збільшення.

Коли вікно перевантаження наближається до цільового вікна, BIC переходить у бінарну фазу пошуку, де вікно має логарифмічне зростання. Якщо вікно перевантаження досягає максимального вікна, BIC входить у фазу максимального зондування, і вікно зростає експоненціально, доки не буде знайдено нове максимальне вікно. Така поведінка відрізняє BIC від інших масштабованих протоколів, таких як HighSpeedTCP і ScalableTCP, де швидкість зростання завжди найшвидша біля максимального вікна. Порівняно з TCP, BIC покращує ефективність з’єднання, зручність TCP, справедливість і стабільність протоколу.

4.2.6 TCP CUBIC

TCP CUBIC є менш складним і більш дружнім до мережі варіантом BIC, запропонованим після того, як виникли деякі занепокоєння щодо агресивності BIC у мережах з малим часом проходження в обидва кінці або низькою швидкістю з’єднання [26]. TCP CUBIC усуває три окремі фази зростання вікон BIC. Замість цього він використовує одну кубічну функцію, яка поводиться подібним чином. Крім того, CUBIC робить швидкість зростання вікна перевантаження незалежною від RTT, а натомість залежить від часу з моменту останньої події перевантаження.

Зокрема, CUBIC збільшує своє вікно перевантаження відповідно до:

$$W_{cubic} = C(t-K^3) + W\max, \tag{4.6}$$

де C – коефіцієнт масштабування, t – час, що минув від останнього зменшення вікна (тобто останньої події перевантаження), W_{max} – розмір вікна безпосередньо перед останнім зменшенням вікна, а $K = \sqrt[3]{W\max + \beta / C}$, де β – це коефіцієнт зменшення вікна перевантаження, застосований під час останньої події перевантаження.

На рисунку 4.3 приведена еволюція вікон контролю перевантажень BIC і CUBIC. Вікно перевантаження CUBIC прагне наблизитися до вікна BIC. Однак він досягає цього за допомогою простої кубічної функції для керування своєю поведінкою, на відміну від BIC, який є більш складним алгоритмом через три окремі фази.

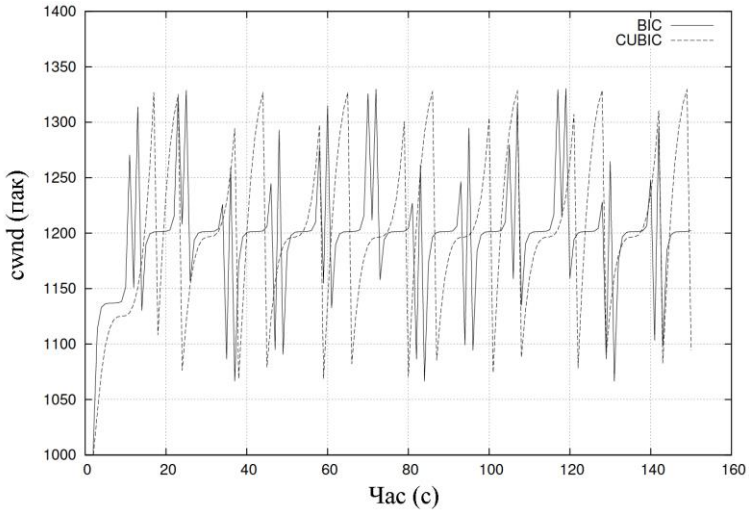

Рисунок 4.3 – Динаміка зміни вікна перевантаження TCP BIC і TCP CUBIC

У TCP CUBIC функція кубічного вікна забезпечує справедливість всередині протоколу, а функція збільшення вікна перевантаження, незалежного від RTT, покращує справедливість RTT.

Оскільки швидкість зростання вікна залежить від t, зрештою всі потоки збіжаться до одного значення, таким чином покращуючи справедливість RTT. Проте це ще не повністю вирішує проблему справедливості RTT.

CUBIC відрізняє себе від багатьох наскрізних алгоритмів контролю перевантажень тим, що він є типовим алгоритмом керування перевантаженнями в ядрі Linux, починаючи з версії 2.6.19. Як наслідок, це одна з небагатьох пропозицій, яка пройшла широке тестування в реальному світі.

Тим не менш, CUBIC не позбавлений недоліків. CUBIC може бути надто агресивним щодо перехресного трафіку (наприклад, викликів VoIP), мати дуже низький час конвергенції або взагалі не конвергентувати, демонструючи значну несправедливість пропускної здатності в деяких сценаріях поведінки мережі.

4.3 Алгоритми контролю перевантаження на основі затримки

Алгоритми на основі затримки мають значний відхід від традиційного підходу на основі втрат. Вони ґрунтуються на припущенні про те, що перед появою перевантаження мережі є деякі ознаки.

Таким чином, вони використовують інформацію про затримку мережі, як правило, у формі RTT, щоб визначити стан черг на маршрутизаторах. Одна з переваг підходу, заснованого на затримці, полягає в тому, що він набагато дружніший до мережі, ніж аналог, заснований на втратах, у тому сенсі, що перевантаження мережі можна завчасно уникнути. Таким чином, схеми на основі затримки можуть уникнути типового коливання вікна перевантаження алгоритмів на основі втрат, наприклад, знайомого «ефекту пили» TCP NewReno, і забезпечити краще використання пропускної здатності каналу.

Однак, досі залишаються сумніви щодо ефективності стратегій, заснованих на затримці.

4.3.1 TCP Vegas

TCP Вегас [10] є основоположною роботою з наскрізного контролю перевантаження TCP на основі затримки, яка безпосередньо вплинула на багато інших алгоритмів на основі затримки.

TCP Vegas вимірює швидкість надсилання, щоб контролювати розмір вікна перевантаження. Крім того, Vegas представляє нові ідеї щодо покращення відновлення втрачених пакетів і вносить зміни в механізм повільного запуску, щоб зменшити втрату пакетів.

Зокрема, Vegas регулює своє вікно перевантаження, порівнюючи очікувану швидкість надсилання з фактичною швидкістю надсилання. По-перше, він отримує очікувану швидкість надсилання за такою формулою:

$$\text{Expected} = \text{WindowSize}/\text{BaseRTT}, \tag{4.7}$$

де WindowSize – поточний розмір вікна перевантаження, а BaseRTT – мінімальний виміряний RTT для з'єднання на даний момент. По-друге, Vegas обчислює фактичну швидкість надсилання за RTT. Він записує час надсилання виділеного сегмента в момент часу T_I. Коли надходить підтвердження виділеного сегмента, у момент часу T_F, фактична швидкість надсилання обчислюється за рівнянням:

$$\text{Actual} = \text{SentData}/(T_F - T_I), \tag{4.8}$$

де SentData – це кількість даних, які передаються між T_I і T_F. По-третє, Vegas визначає два пороги, α і β ($\alpha<\beta$), які є кількістю мінімальних і максимальних пакетів, які повинні бути в черзі в маршрутизаторах. Нарешті, різниця між очікуваною та фактичною швидкістю надсилання дорівнює:

$$\text{Diff} = \text{Expected} - \text{Actual}, \tag{4.9}$$

Під час уникнення перевантаження, якщо Diff<α, вікно збільшується на один сегмент; якщо Diff>β, вікно зменшується на один сегмент; якщо α< Diff<β, вікно залишається незмінним. Як і Reno, Vegas починає з'єднання, викликаючи повільний старт, хоча Vegas використовує менш агресивний варіант. Vegas виходить із повільного старту, коли Diff падає нижче визначеного порогу.

Vegas може значно зменшити перевантаження мережі та підтримувати вікно перевантаження навколо оптимального значення для певного з'єднання. Його лінійне зростання, однак, робить Vegas непридатним для мереж із великим BDP. Крім того, продуктивність Vegas знижується, коли він співіснує з більш агресивними варіантами TCP на основі втрат, такими як Reno. Еволюцію вікна контролю заторів у TCP Vegas показана на рисунку 4.4. Параметри моделювання представлені в таблиці 4.1. Слід зазначити, що оскільки TCP Vegas не розроблений для мереж із великим BDP, потрібен деякий час, поки він повністю налаштується на використання каналу (приблизно при t = 130 с).

Таблиця 4.1 – Параметри імітаційного моделювання

Джерел	1
Пропускна здатність (Мб/с)	100
RTT	128
Черга маршрутизатора	100% BDP
MSS	1460

4.3.2 FAST TCP

FAST TCP, який можна вважати високошвидкісним нащадком Vegas, запропоновано в роботі [27]. Незважаючи на те, що FAST TCP базується більше на принципах Vegas, він більш агресивно збільшує вікно

перевантаження, щоб досягти високої ефективності у високошвидкісних мережах. FAST TCP оновлює своє вікно перевантаження відповідно до наступного рівняння:

$$\text{cwnd} \leftarrow \min\left\{2\cdot\text{cwnd}, (1-\gamma)\cdot\text{cwnd}+\gamma\left(\frac{\text{baseRTT}}{\text{RTT}}\cdot\text{cwnd}+\alpha\right)\right\}, \qquad (4.10)$$

де γ – специфічний параметр алгоритму, baseRTT – мінімальний RTT, виміряний на даний момент, RTT – середній RTT, а α – кількість пакетів, які мають бути поставлені в чергу в маршрутизаторах уздовж мережевого шляху.

По суті, FAST TCP швидко наближається майже до максимального вікна, а потім, у міру збільшення RTT, повільніше наближається до цільового вікна. Крім того, FAST TCP реалізує усуває ефекту вибуху, що спостерігається при дуже великих вікнах перевантаження.

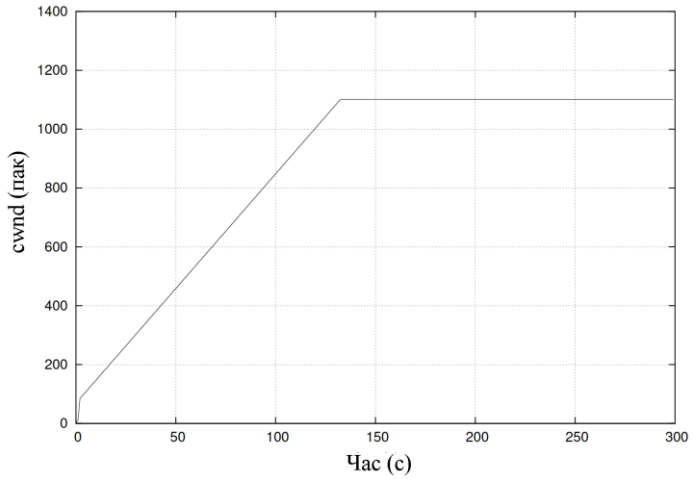

Рисунок 4.4 – Динаміка зміни вікна перевантаження TCP Vegas

FAST TCP може досягти відмінного використання каналів у високошвидкісних мережах, швидко конвергує до справедливості та є стабільним. Тим не менш, було помічено, що FAST TCP не має властивостей

справедливості в деяких мережевих сценаріях. FAST також може бути проблематичним, коли є багато потоків, які конкурують за пропускну здатність. Оскільки кожен потік має на меті поставити в чергу пакетів α у маршрутизаторі з вузьким місцем, буфера може бути недостатньо для зберігання всіх необхідних пакетів n×α, оскільки n збільшується.

4.4 Гібридний алгоритми

4.4.1 TCP-Illinois

TCP-Illinois [28] дотримується алгоритму AIMD, але використовує оцінки затримки для встановлення параметрів збільшення та зменшення вікна перевантаження. Якщо TCP-Illinois не виявляє затримку в черзі (тобто перевантаження мережі), для параметра збільшення встановлюється максимальне значення, завдяки чому вікно перевантаження швидко зростає. Коли затримка в черзі починає наростати, параметр збільшення поступово зменшується, уповільнюючи вікно перевантаження. У разі втрати пакетів вікно перевантаження оновлюється відповідно до рівняння:

$$\text{cwnd} \leftarrow (1-\beta)\cdot \text{cwnd}\,. \tag{4.11}$$

Якщо поточний RTT близький до максимального RTT, виміряного на даний момент, TCP-Illinois робить висновок, що перевантаження мережі спричинило втрату, і $\beta \approx 1/2$. Якщо RTT невелике, β матиме менше значення, що означає, що втрата пакетів, ймовірно, сталася через помилку зв’язку.

Тим не менш, деякі занепокоєння були підняті щодо масштабованості TCP-Illinois у мережах із великим BDP через його фіксований максимальний параметр збільшення вікна (10 пакетів на RTT), а також за наявності трафіку зворотного шляху. На рисунку 4.5 показано динаміку вікна перевантаження для TCP-Illinois.

Можна спостерігати, що початкова більш висока швидкість зростання, а потім більш повільна швидкість зростання (через збільшення затримки), створює характерну увігнуту криву зростання TCP-Illinois.

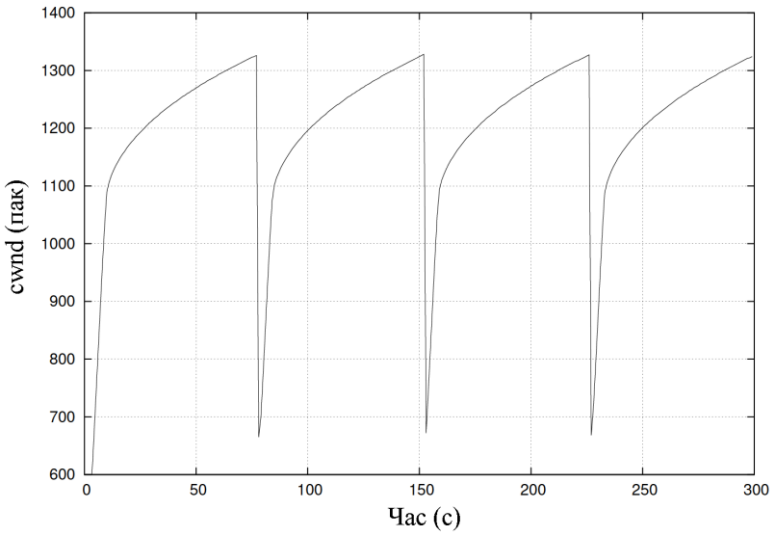

Рисунок 4.5 – Динаміка зміни вікна перевантаження TCP-Illinois

4.4.2 Compound TCP

Compound TCP пропонується в [29] для підвищення ефективності зв’язку та справедливості RTT. Він підтримує два допоміжні вікна контролю перевантаження. Традиційне вікно перевантаження на основі втрат, яке відповідає поведінці AIMD Reno, і масштабоване вікно на основі затримки, створене у TCP Vegas.

Отримане вікно перевантаження є сумою вікон на основі втрат і вікон на основі затримки. Якщо мережеве з’єднання використовується недостатньо, компонент на основі затримки швидко збільшить вікно перевантаження, щоб використовувати доступну пропускну здатність. Коли затори починають накопичуватися, компонент на основі затримки зменшить вікно. Під час сильного перевантаження мережі Compound TCP повертається до поведінки Reno.

Compound TCP на основі затримки дозволяє досягти високої ефективності з’єднання та справедливості RTT. Крім того, оскільки пропускна здатність є нижньою межею компонента на основі втрат, це вирішує проблеми несправедливості TCP Vegas зі схемами на основі втрат Проте було помічено, що Compound TCP може повернутися до поведінки масштабування, подібної до Reno, навіть за слабкого зворотного трафіку, і може страждати від проблем справедливості та масштабованості в каналах з дуже високою затримкою пропускної здатності. Незважаючи на те, що Compound TCP вимкнено за замовчуванням, він присутній у Windows Server 2008, Windows Vista та Windows 7 та інших операційних системах.

5 ПРОПОНОВАНІ МЕТОДИ БОРОТЬБИ З ПЕРЕВАНТАЖЕННЯМ

5.1 Вибір алгоритму

В кваліфікаційній роботі розглядаються поточні та пропонуються нові методи, засновані на затримці, з метою вдосконалення існуючих наскрізних алгоритмів контролю перевантажень на основі втрат пакетів.

Одним із таких варіантів є H-TCP, який вводить незалежне від RTT правило збільшення вікна та має передбачувану, залежну від вікна тривалість періоду перевантаження.

Ця остання характеристика робить H-TCP цікавим вибором для пом’якшення добре відомої проблеми справедливості розподілу пропускної здатності на основі затримок і втрат.

Іншим цікавим алгоритмом є TCP CUBIC. Як описано раніше, CUBIC також має незалежне від RTT правило збільшення вікна перевантаження, і, що найважливіше, це дуже популярний алгоритм, який пройшов широке тестування в реальному світі. Через його популярність було обрано CUBIC як тестовий стенд для вивчення пропонованих методів.

Пропонується два наскрізних алгоритма контролю перевантаження: Max-delay CUBIC, гібридний алгоритм контролю перевантаження, і Min-delay CUBIC, алгоритм контролю перевантаження, заснований на чистій затримці.

Обидва алгоритми базуються на CUBIC з метою покращення продуктивності CUBIC у показниках справедливості та зручності мережі, зберігаючи при цьому дуже високу ефективність.

Ключовим аспектом обох алгоритмів є простота їх реалізації та розгортання в поточних мережах, тому, що потрібно оновити лише сторону відправника. Таким чином, обидва алгоритми підтримують зворотну сумісність з усіма стеками TCP, які зараз розгорнуті в Інтернеті.

5.2 Проблеми та міркування проектування

Алгоритми контролю перевантажень на основі затримки, хоча теоретично є дуже привабливими, стикаються з додатковими перешкодами при реалізації в порівнянні з їхніми аналогами, заснованими на втратах. Таким чином, слід бути обережним при розробці алгоритму, який використовує інформацію про затримку.

5.2.1 Надання можливості конкуруючим потокам постійно виявляти перевантаження

У мережі з багатьма конкуруючими потоками і, отже, зі значним статистичним мультиплексуванням може бути низька кореляція між фактичною затримкою в черзі та затримкою в черзі, виміряною потоком. Таким чином, у певний момент деякі потоки можуть виявити, що мережа перевантажена, а інші – ні, в результаті чого вікно перевантаження деяких потоків зменшується, а інших – збільшується. Іншими словами, не всі конкуруючі потоки можуть виявити однакову динаміку черги на маршрутизаторі з вузьким місцем. Це призводить до зниження справедливості, оскільки потоки не бачать мережу в узгодженому стані. Деякі автори пов’язують це з проблемами вибірки і вибуховою природою віконних алгоритмів контролю перевантаження, таких як TCP.

Щоб отримати уявлення про проблему вибірки, розглянемо мережу з багатьма конкуруючими потоками. Зазвичай вибірка затримки складається з вимірювання часу проходження даного пакета туди й назад. Оскільки вікно перевантаження кожного конкуруючого потоку може стати досить малим, і, отже, передається небагато пакетів, кількість зразків затримки, зібраних кожним потоком, може бути недостатньою для виявлення накопичення затримки в черзі (це, по суті, явище зміщення). . Проблема погіршується версіями TCP, які вимірюють лише затримку черги в кінці RTT.

Одна з ключових ідей для вирішення цієї проблеми наведена в нещодавній пропозиції [30]. Його автори стверджують, що потоки, виявивши перевантаження в мережі, не повинні негайно зменшувати своє вікно перевантаження. Замість цього вони повинні почекати заздалегідь визначений проміжок часу, а потім зменшити вікно перевантаження. Наслідком цієї, здавалося б, простої ідеї в наведеному вище сценарії є те, що, хоча даний потік заморожує своє вікно перевантаження, таким чином утримуючи мережу перевантаженою, інші потоки також виявлятимуть, що перевантаження накопичується.

Таким чином, більшість потоків виявлять перевантаження та зменшать своє вікно перевантаження. По суті, потоки тримають мережу перевантаженою досить довго, щоб конкуруючі потоки могли мати послідовне уявлення про мережу. Крім того, в гібридних алгоритмах керування перевантаженням зазвичай вважається хорошою практикою перемикання в режим на основі втрат, коли вікно перевантаження нижче певного порогу. Таким чином, потік активує режим на основі затримки лише тоді, коли доступна достатня кількість зразків RTT для виявлення накопичення затримки в черзі.

Крім того, TCP, будучи віконним алгоритмом контролю перевантаження, зазвичай надсилає дані пакетами. Така поведінка сприяє агрегації пакетів з одного потоку на вузьких місцях в чергах маршрутизаторів. На рисунку 5.1 показано надходження пакетів від двох конкуруючих потоків з однаковим базовим часом RTT. На рисунку 5.1 (а) показано, що без пейсингу (регулювання) пакети з двох різних потоків матимуть різні погляди на динаміку черги на маршрутизаторі з вузьким місцем. Як обговорювалося раніше, така поведінка зрештою призведе до несправедливості між потоками.

Цю проблему можна пом’якшити за допомогою впровадження пейсингу [71]. Швидкість сприяє синхронізації потоку, і навіть якщо синхронізація може мати небажаний вплив на застарілий TCP, вона покращує

справедливість у високошвидкісних варіантах TCP і зменшує спалахи передачі даних. Це дві дуже важливі характеристики при розробці високошвидкісного протоколу TCP. Таким чином, пейсинг може відігравати важливу роль у покращенні справедливості та забезпеченні узгодженої оцінки прохідності, що має вирішальне значення в алгоритмах на основі затримки. На рисунку 5.1 (б) видно, що з пейсингом пакети двох конкуруючих потоків стають менш кластеризованими.

Це важливо, оскільки це дає змогу обом потокам мати узгоджене уявлення про мережу. Тим не менш, єдиними двома високошвидкісними варіантами TCP, які реалізують пейсинг, є FAST TCP і SyncTCP.

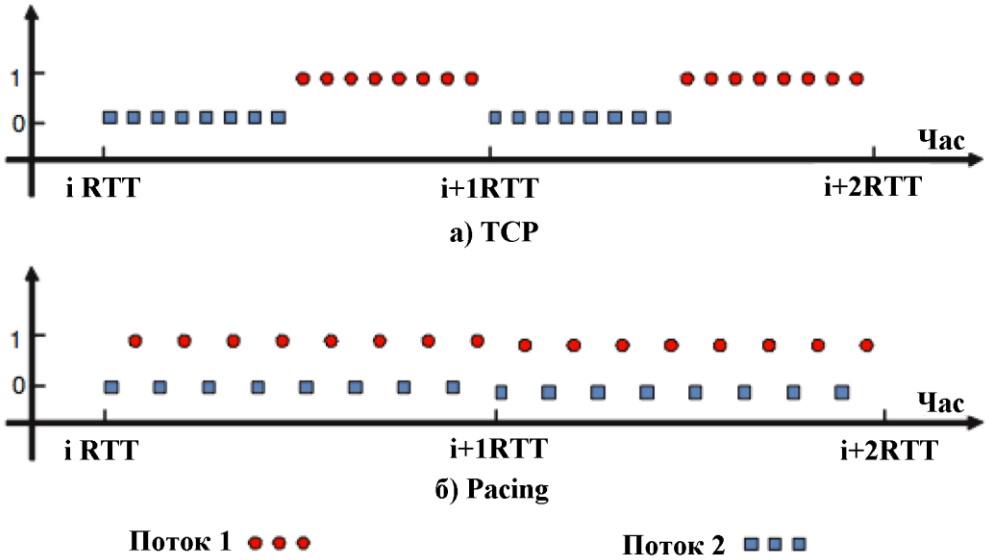

Рисунок 5.1 – Час надходження пакетів двох конкуруючих потоків

5.2.2 Правильна оцінка затримки RTT

Однією з головних проблем, з якою стикаються практично всі алгоритми на основі затримки, є те, як визначити базову затримку розповсюдження для кожного потоку. Оскільки всі потоки залежать від значення базового часу проходження для їх роботи, неправильна оцінка цього значення призведе до несправедливості.

Це дуже поширена проблема, з якою стикаються алгоритми на основі затримки, такі як TCP Vegas і FAST TCP.

Оскільки алгоритми керування перевантаженнями на базі TCP Vegas прагнуть утримувати певну кількість пакетів у черзі на маршрутизаторі з вузьким місцем, пізніші потоки, що надходять у мережу, вимірюватимуть більший базовий час проходження в обидві сторони, змушуючи їх отримувати більшу частину доступної смуги пропускання, тобто викликаючи суттєву несправедливість.

Ця проблема проілюстрована на рисунку 5.2. Слід зазначити, що порівняно з попередніми потоками пізніші потоки мають вищу пропускну здатність.

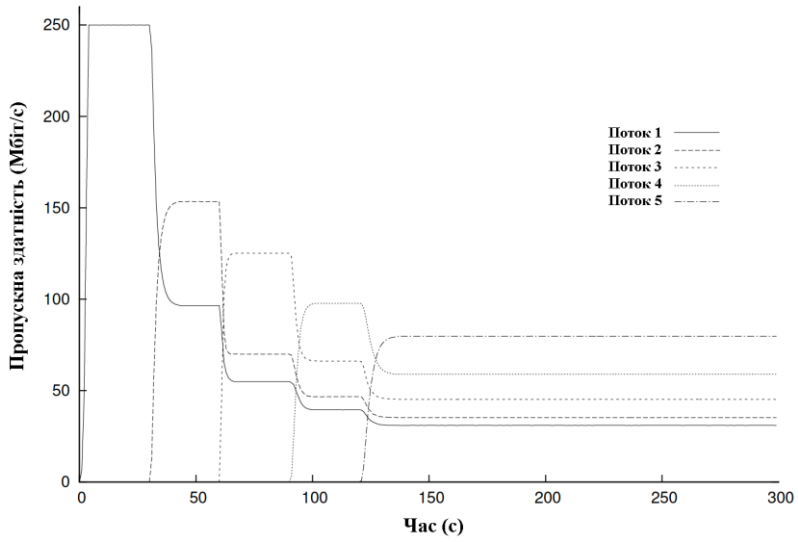

Рисунок 5.2 – Взаємодія 5 FAST TCP-потоків

Незважаючи на це, такі алгоритми залишаються корисними та можуть служити компонентом будь-якого алгоритму на основі затримки, який прагне бути дуже дружнім до мережі та готовий пожертвувати певною ефективністю, щоб досягти цього.

Крім того, використання максимального виміряного RTT, на відміну від мінімального RTT, має ряд цікавих переваг.

По-перше, максимальний RTT значно легше виміряти за допомогою нових потоків, оскільки він не потребує будь-якого механізму дренування черги. По-друге, він байдужий до наявності перехресного трафіку, такого як UDP. Нарешті, це дозволяє алгоритму постійно тримати пакети в черзі в маршрутизаторі з вузьким місцем, що забезпечує 100% ефективність з'єднання.

Потенційним недоліком цього підходу є початковий короткостроковий перехідний період несправедливості, спричинений необхідною фазою зондування зв'язку. Однак ця короткочасна несправедливість може, у деяких сценаріях, бути бажаною характеристикою, оскільки новим потокам надається тимчасово підвищений пріоритет.

5.2.3 Масштабованість

Традиційно алгоритми на основі затримки спрямовані на те, щоб зберегти певну кількість пакетів у черзі маршрутизатора з вузьким місцем. Ця поведінка має дві цікаві властивості. По-перше, це суттєво покращує справедливість RTT, оскільки постійна кількість пакетів ставиться в чергу, незалежно від RTT потоку. По-друге, він може постійно підтримувати хорошу ефективність з'єднання за умови використання високошвидкісного варіанту.

Однак такий підхід має свою ціну. Ця процедура лінійно масштабується залежно від кількості потоків, і зі збільшенням кількості потоків зростатиме зайнятість черги та затримка. Високошвидкісні варіанти Vegas, такі як FAST TCP, зберігають велику кількість пакетів у черзі на потік, щоб досягти високої ефективності. Небажаним побічним ефектом є те, що навіть за невеликої кількості потоків черга маршрутизатора може бути переповнена. Таким чином, може виникнути значна нестабільність і несправедливість мережі. Слід зазначити, що певною мірою ця проблема також впливає на багато гібридних алгоритмів контролю перевантажень.

Щоб проілюструвати цю проблему, на рисунку 5.3 показано пропускну здатність і скидання пакетів 32 конкуруючих потоків FAST TCP у мережі з пропускною здатністю 250 Мбіт/с. Розмір черги маршрутизатора з вузьким місцем становить 100% від BDP. MSS встановлено на 1460 байт, а RTT рівномірно розподілено між 25 мс і 160 мс. Крім того, на рисунку 5.4 показано стан черги на маршрутизаторі з вузьким місцем, де видно, що кожен новий потік сприяє додатковому зайняттю черги та затримці.

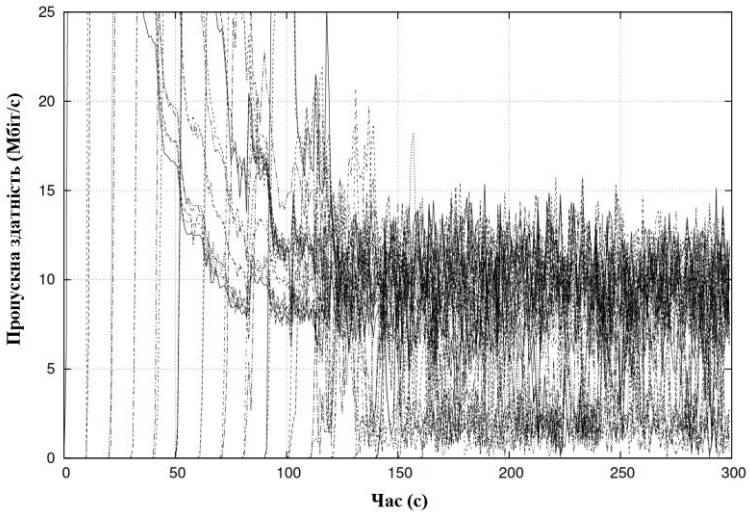

Рисунок 5.3 – Пропускна здатність для 32 FAST TCP-потоків

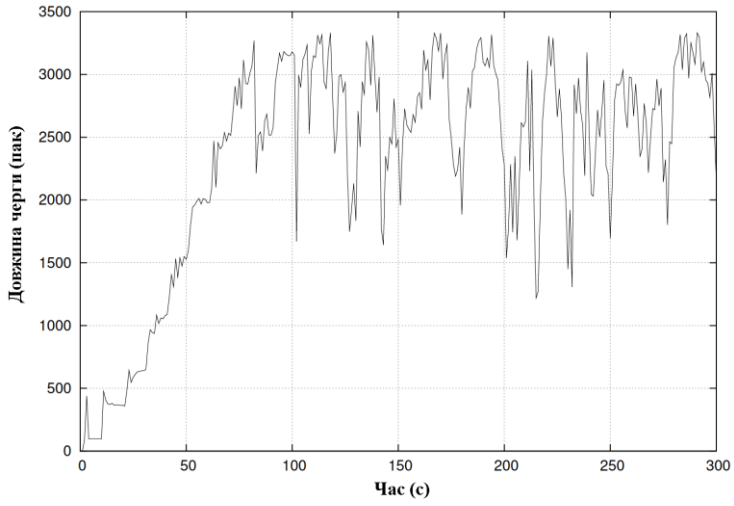

Рисунок 5.4 – Довжина черги маршрутизатора з 32 FAST TCP потоками

Підводячи підсумок, пропонується використати наступні методи:

- механізм пейсингу для покращення синхронізації потоку;
- простий механізм синхронізації на основі затримки для покращення справедливості потоків;
- Min CUBIC метод з використанням мінімальної затримки для оцінки затримки передачі;
- Max CUBIC метод з використанням максимальної затримки, де використовується максимальне значення RTT, усуваючи необхідність оцінювати затримку передачі;
- зменшення вікна перевантаження на основі порогового значення для покращення масштабованості;
- отримане емпіричним шляхом, залежне від RTT, правило зменшення вікна перевантаження, щоб покращити справедливість RTT.

6 АЛГОРИТМИ ЗАПОБІГАННЯ ПЕРЕВАНТАЖЕННЮ

6.1 Виявлення виснаження

Виявлення потоків на основі втрат дає змогу алгоритму CUBIC на основі затримки виявляти, що він конкурує з потоками, що не співпрацюють (наприклад, потоки на основі втрат), дозволяючи йому більш справедливо конкурувати за пропускну здатність.

Звісно, що в стаціонарному стані потік CUBIC з максимальною затримкою слідує дуже передбачуваному шаблону збільшення та зменшення вікна перевантаження. Після того, як потік зменшує своє вікно перевантаження, затримка в черзі також передбачувано зменшується. Невдовзі після події зменшення вікна очікується, що потік збільшить вікно перевантаження.

Однак, якщо затримка в черзі залишається високою, то потік зменшує вікно перевантаження вдвічі. Це вагомий показник того, що щось у мережі не реагує на сигнали затримки в черзі – зазвичай ознака наявності потоків на основі втрат.

Використовуючи цю інформацію, під час режиму на основі затримки алгоритм може відстежувати поведінку збільшення та зменшення вікна перевантаження. Якщо алгоритм виявляє, що він зменшив своє вікно перевантаження вдвічі, не збільшуючи його принаймні один раз між ними, він увійде в режим на основі втрат на певний період часу, оскільки це дуже сильний показник наявності в мережі потоків на основі втрат. Після закінчення цього часу потік повертається до режиму на основі затримки.

Для запобігання хибним спрацьовуванням і непотрібній агресивності потоку, потік активує цей механізм лише після того, як виявить, що він перебуває в стабільному стані. Стаціонарний стан можна легко підтвердити, спостерігаючи за двома змінними, присутніми в протоколі CUBIC.

Цими змінними є поточне вікно перевантаження та вікно перевантаження в кінці останньої епохи (тобто коли воно досягло максимального значення). Якщо поточне вікно перевантаження більше або дорівнює максимальному вікну перевантаження попередньої епохи, то потік досяг стаціонарного стану. Це додатково показано в лістингу 6.1 алгоритму виявлення потоків на основі втрат.

Лістинг 6.1 – Псевдокод алгоритму виявлення потоків на основі втрат

```
Ініціалізація:
increasedCwnd ← true stablePhase ← false

Для кожного ACK:
if cwnd > prevMaxCwnd and stablePhase = false then
  /* досяження стаціонарного стану */
  stablePhase ← true
end if

При збільшенні cwnd:
increasedCwnd ← true

При зменшинні cwnd:
if increasedCwnd = false and stablePhase = true then
  /* виявлено виснаження - не зменшувати cwnd і перехід у режим
на основі втрат */
  mode ← lossBased
  stablePhase ← false
else
/* зменшити cwnd */
increasedCwnd ← false
end if
```

6.2 Покращення справедливості RTT

Процедура складається з функції нормалізації на основі RTT, яка забезпечує покращену справедливість RTT серед конкуруючих потоків. Ключове розуміння випливає зі спостереження про те, що пропускна здатність потоку приблизно визначається як Throughput=cwnd/RTT. Таким чином, якщо потоки з довшими RTT сходяться до більшого вікна перевантаження, ніж потоки з коротшими RTT, справедливість RTT може

бути покращена. Слід зазначити, що цю процедуру можна використовувати лише з підходом максимальної затримки, оскільки вона конфліктує з механізмом спорожнення черги, необхідним для підходу мінімальної затримки.

Можливі стратегії вирішення цієї проблеми мають два попередні рішення. Перший підхід полягає у зміні правила збільшення вікна перевантаження. Таким чином, потоки з коротшими RTT збільшують перевантаження повільніше, а потоки з довшими RTT збільшують вікно перевантаження швидше. Другий підхід розглядає проблему під протилежним кутом: замість того, щоб намагатися маніпулювати вікном під час фази збільшення, робиться це під час фази зменшення, просто маніпулюючи значенням параметра зменшення, β. За замовчуванням CUBIC використовує параметр β зі значенням 0,2, що означає, що вікно перевантаження зменшується на 20% у кожній події перевантаження. Мета полягає в динамічному маніпулюванні цим значенням. По суті, щоб підвищити справедливість, потоки з довшими RTT мають менший коефіцієнт зменшення, а потоки з коротшими RTT мають більший коефіцієнт зменшення. Більш привабливий другий підхід. Основна причина полягає в тому, що перший підхід передбачає зміну основного протоколу CUBIC, тобто його правила збільшення вікна перевантаження. Роблячи це, ми фактично створили б інший алгоритм, який перестав би бути КУБІЧНИМ. При виборі другого підходу алгоритм CUBIC залишається незмінним, оскільки коригується лише параметр зменшення.

Динамічний діапазон значень RTT простирається від 25 мс до 160 мс базової затримки поширення в обидві сторони. Цей набір визначає 25 мс і 160 мс як низьке RTT і як високе RTT відповідно. Рівняння, яке, найкраще покращує спостережувану несправедливість RTT виглядає наступним чином:

$$\text{adjust}(x) = 1 - \frac{x^2}{x^2 + 10}. \qquad (6.1)$$

Для покращення справедливості RTT параметр зменшення, β, має динамічно обчислюватися відповідно до RTT наступним чином:

$$\beta=\text{adjust}(\text{scale}(\gamma)), \tag{6.2}$$

де γ – максимальний час проходження туди й назад, виміряний на даний момент кожним потоком. $\gamma=Q_{delayMax}+baseRTT$, де $Q_{delayMax}$ – максимальна затримка в черзі маршрутизатора з вузьким місцем. Якщо припустити, що буфери маршрутизатора забезпечені, то $Q_{delayMax}$ = 160 мс, оскільки розмір буфера встановлено на 100% від BDP, з D=160 мс. baseRTT∈[25, 160] мс, є базовою затримкою поширення потоків туди й назад. Таким чином, γ_{min}=185 мс і γ_{max}=320 мс. β обмежений між 0,15 і 0,40. scale(. . .) – це допоміжна функція для перетворення параметра γ на значення для функції adjust(. . .). По суті, масштабується параметр γ, щоб результуюче значення, β=adjust(scale(γ)), містилося в заданому діапазоні [0,15, 0,40].

$$\begin{gathered}\{x_1 : \text{adjust}(x_1)=\beta_{max}\}\\ \{x_2 : \text{adjust}(x_2)=\beta_{min}\}\\ \text{scale}(\gamma)=\frac{x_2-x_1}{\gamma_{max}-\gamma_{min}}\cdot(\gamma-\gamma_{min})+x_1\end{gathered}, \tag{6.3}$$

де β_{max} – бажане максимальне значення β, а β_{min} – бажане мінімальне значення β. Оскільки механізм динамічно регулює значення β між 0,15 і 0,40, відповідно до γ, потоки з коротшими RTT можуть стати менш агресивними, ніж вихідний протокол TCP CUBIC (β=0,4). І навпаки, потоки зі значно довшими RTT можуть стати трохи агресивнішими, ніж вихідний протокол CUBIC (β=0,15). Цей діапазон значень утримує протокол достатньо агресивним, щоб підтримувати хорошу ефективність з'єднання у випадку потоків із коротшими RTT, і в той же час не робить протокол надто агресивним для мережі з потоками з довшими RTT.

6.3 Обмежений повільний старт

Обмежений повільний старт пропонується в [31], щоб покращити продуктивність TCP-з'єднань із великими вікнами перевантаження. Вікно перевантаження потоку в мережах із величезними продуктами затримки пропускної здатності (BDP) може бути збільшено надто агресивно під час повільного старту. З цієї причини кілька тисяч пакетів можуть бути втрачені з вікна даних, що призведе до зниження продуктивності потоку та збільшення навантаження на саму мережу.

Обмежений повільний запуск – це, по суті, більш м'який повільний запуск, спрямований на зменшення кількості відкинутих пакетів під час процедури повільного запуску. Його головна ідея полягає в тому, щоб повільніше збільшувати вікно перевантаження TCP під час фази повільного запуску та, таким чином, покращувати його продуктивність за рахунок зменшення кількості викинутих пакетів.

6.4 Режим на основі втрат для малих вікон перевантаження

Щоб покращити проблеми вибірки, які можуть вплинути на алгоритми керування перевантаженням на основі затримки, пропонується використовувати механізм, який дозволяє алгоритму перемикатися в режим на основі втрат, після того як його вікно перевантаження впало нижче певного порогу.

Це означає, що потік поводитиметься як вихідний протокол TCP CUBIC, якщо його вікно перевантаження менше 40 сегментів. Додаткова перевага цього механізму полягає в тому, що, зберігаючи мінімальний поріг вікна, потоки можуть уникнути повного зниження пропускної здатності через конкуруючі потоки на основі втрат, оскільки сигнали затримки ігноруються, коли вікно перевантаження падає нижче low_window. Однак, коли велика кількість конкуруючих потоків співіснує в каналі зв'язку, значення вікна

перевантаження зрештою падає нижче low_window для всіх потоків. За цих умов потоки контролюють свої вікна перевантаження за допомогою алгоритму CUBIC на основі втрат.

6.5 TCP CUBIC на основі максимальної затримки

6.5.1 Основний алгоритм

Підхід із максимальною затримкою складається з нової методики на основі затримки, де максимальне виміряне значення RTT використовується як порогове значення для координації поведінки алгоритму. Цей підхід контрастує з традиційним методом використання мінімального виміряного RTT, що робить його цікавим підходом. Максимальне значення RTT легше отримати, ніж мінімальне RTT, як обговорювалося раніше. Ще одна цікава характеристика полягає в тому, що алгоритм може постійно підтримувати пакети в черзі на маршрутизаторі з вузьким місцем, таким чином забезпечуючи високу ступінь використання каналу.

Основна ідея цього підходу полягає в тому, щоб змусити CUBIC реагувати на більш ранні ознаки перевантаження, ніж на втрату пакетів (тобто збільшення зайнятості черги маршрутизатора), і таким чином уникнути переповнення черги маршрутизатора. Однак, щоб виміряти RTT, при якому відбувається скидання пакетів, алгоритм повинен спочатку перевірити мережу на максимальний RTT. Це досягається шляхом збереження стандартного механізму повільного запуску на основі втрат, подібно до Compound TCP, у поєднанні з додатковим перехідним періодом, коли він залишається в режимі на основі втрат (лістинг 6.2).

Після початкового режиму на основі втрат алгоритм змінюється на режим на основі затримки. У режимі на основі затримки алгоритм регулює вікно перевантаження згідно з правилами, описаними в алгоритмі CUBIC з максимальною затримкою, а δ встановлено на 500 мс згідно рекомендацій.

Лістинг 6.2 – Псевдокод алгоритму Max CUBIC

```
Ініціалізація:
δ ← 500 /* період очікування в мілісекундах */
reduce ← false /* Якщо істина, то зменшити cwnd */
threshold ← 100 /* Поріг затримки в мілісекундах */
Для кожноого ACK:
if now > waitTime then
  if reduce then
    cwnd ← β · cwnd /* β – параметр зменшення */
    ssthresh ← cwnd
    waitTime = now + δ
    reduce ← false
  else if rtt > maxRTT – threshold then
    reduce ← true
    waitTime ← now + δ
  else
    /* збільшити вікно перевантаження за правилами CUBIC */
  end if
else
  /* залишити cwnd без змін */
end if

При втраті пакета:
cwnd ← β·cwnd /* β – параметр зменшення */
ssthresh ← cwnd
```

Для кожного пакета підтвердження Max-delay CUBIC перевіряє значення поточного RTT. Якщо значення RTT вище визначеного порогу, Max-delay CUBIC зменшує своє вікно перевантаження. Натомість він зберігатиме поточне значення вікна перевантаження протягом δ мс.

Після цього часу Max-delay CUBIC нарешті зменшує вікно перевантаження. Після зменшення вікна перевантаження він чекає додаткові δ мс, поки знову не почне перевіряти RTT. Це робиться для того, щоб дозволити маршрутизаторам зменшити свої черги та затримку, спричинену чергою. Окрім додаткового компонента керування на основі затримки, поведінка Max-delay CUBIC ідентична поведінці CUBIC.

По суті, алгоритм підтримує динаміку вікна CUBIC, хоча й раніше реагує на ознаки перевантаження, щоб запобігти втраті пакетів і зменшити затримку мережі.

Пороговий параметр (threshold) – це абсолютне значення затримки, яке сигналізує, коли потік повинен зменшити вікно перевантаження. Він являє собою мінімальну відстань, яку повинен підтримувати потік від максимального виміряного часу обертання. Наприклад, якщо максимальне виміряне значення RTT становить 300 мс, а порогове значення встановлено на 100 мс, тоді потік зменшує своє вікно перевантаження, коли значення RTT дорівнює 200 мс.

Його значення слід вибирати відповідно до бажаного режиму роботи алгоритму. Якщо використовується більше значення, напр. 100 мс, черги залишаються меншими, покращуючи продуктивність чутливих до затримки потоків, таких як VoIP та інтерактивні сеанси віддаленого терміналу. Однак більш високе порогове значення погіршує короткострокову справедливість. Коли нові потоки входять у мережу та перевіряють максимальне значення RTT, старі потоки, які вже перебувають у режимі на основі затримки та припускають, що не використовується механізм покращення короткострокової справедливості, швидше реагуватимуть на перевантаження мережі та зменшуватимуть вікна перевантаження, перш ніж це зроблять нові потоки. І навпаки, якщо використовується менше значення, короткострокова несправедливість покращується, оскільки алгоритм більш точно імітує поведінку вихідного алгоритму CUBIC, водночас досягаючи значно кращої довгострокової справедливості, ніж CUBIC. Деякі втрати пакетів можуть статися, якщо вибраний поріг замалий. По суті, вибір порогового значення передбачає компроміс між короткостроковою справедливістю, яка може бути важливою, а може й ні, та зручністю мережі, тобто зайнятістю черги та затримкою.

Спочатку основна ідея алгоритму Max-delay CUBIC полягала у використанні оціненої максимальної затримки в черзі. Тобто обчислення різниці між максимальним і мінімальним виміряним RTT і зменшення вікна перевантаження, коли було досягнуто відсоток від максимального значення затримки в черзі.

Незабаром стало зрозуміло, що такий підхід призведе до непотрібних зусиль, оскільки для належної роботи для визначення фактичної затримки розповсюдження потрібна техніка виснаження черги.

Метод на основі порогового значення використовує лише максимальне значення RTT. Початковий підхід полягє у вимірюванні максимального RTT, а потім зменшення вікна перевантаження після того, як поточний RTT досяг певного відсотка від максимального RTT, наприклад, 90%. Ключова ідея полягає в тому, щоб зменшити вікно перевантаження до досягнення максимального RTT, щоб уникнути втрати пакетів. Цей підхід дуже добре працює з потоками з однаковою затримкою поширення. Однак він має низьку справедливість з потоками з різнорідними RTT. Ключ до розуміння цієї проблеми полягає в знанні того, що якщо використовувати відсоток від максимального RTT як порогове значення, потоки з різними затримками поширення оцінюватимуть різні цільові затримки в черзі. Це змусить потоки реагувати на різні значення затримки мережевої черги, спричиняючи несправедливість, оскільки потоки зменшують свої вікна перевантаження в різний час. Для досягнення справедливості потоки повинні використовувати однакову цільову затримку в черзі, незалежно від їхнього RTT. Але не бажано оцінювати затримку в черзі за допомогою мінімального RTT, оскільки це призводить до зниження ефективності. Таким чином, цільове порогове значення має бути абсолютним значенням, віднімається від максимального RTT.

TCP CUBIC з максимальною затримкою не покладається на базовий час RTT для координації своєї поведінки, що може бути важко визначити. Замість цього він використовує максимальний час проходження туди й назад і усуває необхідність оцінювати фактичну затримку поширення потоків. Таким чином, йому не потрібно використовувати будь-який механізм скидання черги, що робить алгоритм більш надійним за наявності перехресного трафіку, такого як UDP. Нарешті, довгострокова справедливість значно покращена порівняно з CUBIC.

6.5.2 Max CUBIC з NewReno

Інший підхід полягає в швидкому збільшенні вікна перевантаження за допомогою правила агресивного збільшення вікна (тобто CUBIC), в той час як затримка залишається нижче певного порогу затримки.

Після досягнення порогового значення це означає, що мережа використовується повністю (100%), і алгоритм змінює режим роботи, щоб імітувати NewReno.

Цей подвійний режим роботи дозволяє алгоритму ефективно використовувати мережу та підтримувати зворотну сумісність із застарілими потоками.

Ключова відмінність цього підходу від подібних алгоритмів полягає в тому, що, це гібридний алгоритм, який не використовує правило збільшення на основі затримки, засноване на TCP Vegas. Натомість він використовує більш масштабоване правило AIMD на основі затримки (тобто CUBIC), незалежно від кількості потоків, що робить його теоретично більш масштабованим.

По суті, він підтримує сумісність із застарілими потоками TCP, зберігаючи дуже високу ефективність мережі. Псевдокод алгоритму представлений у форматі Max CUBIC з алгоритмом NewReno в лістингу 6.3.

Лістинг 6.3 – Псевдокод алгоритму MAX CUBIC/NEWRENO

```
Для кожного ACK:
if rtt > maxRTT - threshold then
  cwnd ← cwnd + 1/cwnd /* NewReno правило */
else
  /* збільшити вікно перевантаження за правилами CUBIC */
end if

При втраті пакету:
cwnd ← β*cwnd /* β є параметром зменшення */
ssthresh ← cwnd
```

На відміну від інших варіантів CUBIC з максимальною затримкою, ця версія не потребує використання механізму пейсингу, оскільки це, по суті, алгоритм на основі втрат, який використовує лише сигнали на основі затримки, щоб оцінити, чи зараз мережа недостатньо використовується. Тому йому не потрібна більш точна інформація про затримку.

Як і стандартний алгоритм CUBIC з максимальною затримкою, цей алгоритм забезпечує дуже високу ефективність мережі. Крім того, цікавою характеристикою є те, що після завершення більш агресивної фази, заснованої на затримці, він перемикає свій режим роботи на NewReno TCP, що призводить до TCP-дружньої поведінки.

6.5.3 Max CUBIC із покращеною короткостроковою справедливістю

Це ще один підхід, який використовує основний алгоритм максимальної затримки. Його головною перевагою є короткострокове покращення справедливості протягом періоду, коли нові потоки (тобто потоки в режимі на основі втрат) входять і досліджують мережу.

При виборі параметра порогового значення затримки Max-delay CUBIC існує компроміс між покращеною короткостроковою справедливістю та довгостроковою агресивністю.

Якщо використовується більше значення, Max-delay CUBIC зменшить черги, таким чином зменшуючи зайнятість черги та затримку в мережі. Це досягається ціною зниження короткострокової справедливості. Якщо натомість використовується менше значення, максимальна затримка CUBIC має кращу короткострокову справедливість, але зберігає більші черги на маршрутизаторах (лістинг 6.4).

Цей варіант намагається зменшити короткострокову проблему справедливості, коли нові потоки надходять у мережу, водночас дозволяючи використовувати порогове значення, яке зберігає черги малими та покращує продуктивність міжмережевого трафіку, такого як VoIP та Веб-трафік.

Лістинг 6.4 – Псевдокод алгоритму Max CUBIC із покращеною короткостроковою справедливістю

```
Ініціалізація:
δprobe ←10 /* Час очікування зондування мережі в секундах */
probeChangeTimer ← 0 /* Таймер переходу стану зондування */
probeEndTimer ← 0 /* Таймер відключення зондування */

Для кожного ACK:
/* Зміна на орієнтованого на втрату, якщо максимальний RTT */
if rtt ≥ maxRTT then
  maxRTT ← rtt
  if probe ≠ on or rtt>prevMaxRTT then
    probe ← on
    probeChangeTimer←now+δprobe /* оцінити стан мережі за δprobe
секунд */
    prevMaxRTT ← rtt
  end if
end if

/* В режимі втрати, доки не закінчиться таймер зондування */
if probe = on then
  if now < probeChangeTimer then
    cwnd ← cwnd+1/cwnd /* NewReno increase rule */
  else
    probe ← freezing
    probeEndTimer ← now+δprobe
  end if
end if

/* Виконати фактичну оцінку стану мережі та перевірити, чи можна
перейти в повний режим на основі затримки */
if probe = freezing or probe = decreasing then
  if rtt≤maxRTT − treshold then
    probe ← off
    /*з цього моменту алгоритм CUBIC з максимальною затримкою */
    else if now > probeChangeTimer then
      if probe = freezing then
        probe ← decreasing
        probeChangeTimer ← now+δprobe
      else
        /* probe = decreasing */
        cwnd ← cwnd*decreaseFactor
        probeChangeTimer ← now + δprobe
      end if
  else
    /* залишити cwnd незмінним */
  end if
else
  /* дотримуватися алгоритму максимальної затримки CUBIC */
end if
```

По суті, цей підхід, хоча і ідентичний CUBIC з максимальною затримкою, використовує додатковий механізм, який може виявляти некооперативні (тобто засновані на втратах) потоки, що входять у мережу, і перемикатися в режим на основі втрат.

Коли зайнятість черги маршрутизатора максимальна, і, отже, максимальний RTT, він переключить свою роботу в режим на основі втрат, використовуючи правило збільшення вікна перевантаження NewReno. Під час режиму на основі втрат використовувати більш консервативне правило збільшення вікна перевантаження є кращим, ніж правило CUBIC, оскільки більш агресивна поведінка CUBIC скоротить періоди перевантаження.

Епоха перевантаження – це період між двома послідовними скороченнями вікон.

Цей підхід, як і основний алгоритм максимальної затримки, має дуже хорошу ефективність, покращує справедливість, а також демонструє покращену зручність роботи в мережі після початкової фази зондування RTT.

6.6 Алгоритм Min CUBIC з мінімальною затримкою

Поріг затримки використовується для керування динамікою вікна перевантаження. Використання порогу, на відміну від традиційного підходу збереження кількості пакетів у черзі на маршрутизаторі з вузьким місцем, дозволяє масштабувати ці алгоритми, незалежно від кількості потоків, присутніх у мережі. Крім того, для оцінки фактичного значення затримки передачі (базового RTT) використовується метод скидання пакетів в черзі, оскільки цей підхід вимагає її для коректної роботи.

На відміну від підходу максимальної затримки, цей метод не потребує перевірки максимального RTT. Це дозволяє алгоритму бути повністю заснованим на затримці та бути здатним досягти рівня втрати пакетів 0%. Основним недоліком цього підходу є невелике зниження ефективності каналу, спричинене механізмом дренування черги.

Алгоритм регулює своє вікно перевантаження відповідно до правил, описаних в алгоритмі CUBIC з мінімальною затримкою. Min CUBIC дотримується тих самих правил, що й Max CUBIC. Основна відмінність полягає в тому, що TCP CUBIC з мінімальною затримкою використовує мінімальний виміряний RTT (базовий RTT) для керування своєю роботою, а CUBIC з максимальною затримкою використовує натомість максимальний RTT.

На відміну від CUBIC з максимальною затримкою, алгоритм CUBIC з мінімальною затримкою використовує більш стандартний підхід, і, як наслідок, було менше проблем, які потрібно було вирішити. Однак, як і Max-delay CUBIC, Min-delay CUBIC має використовувати абсолютне порогове значення затримки в черзі. Відповідно, єдиним питанням, яке потребувало більшої уваги, був вибір техніки зливу черги. Синхронізуючи потоки, можна збільшити шанси на належне спорожнення черг, оскільки потоки зменшуватимуть і збільшуватимуть свої вікна перевантаження. По суті, механізм організації черги, який, залишаючись досить простим, досліджує синхронізацію потоку, щоб забезпечити належне спорожнення черг, навіть із великою кількістю конкуруючих потоків (лістинг 6.5).

Цей варіант алгоритму здатний підтримувати рівень втрати пакетів на рівні 0%, включаючи більш агресивну фазу повільного запуску, навіть з багатьма конкуруючими потоками. Алгоритм контролю перевантажень, який прагне уникнути втрати пакетів, має цікаві переваги, оскільки (надмірно) затримані та відкинуті пакети повинні бути повторно передані, без потреби збільшуючи навантаження на мережу. Крім того, він може бути значно більш масштабованим, ніж алгоритми контролю перевантажень на базі TCP Vegas, які зазвичай вимагають значно великих черг для підтримки зручності мережі з великою кількістю одночасних потоків. Крім того, черги завжди залишаються невеликими, що значно покращує продуктивність чутливого до затримки трафіку, такого як VoIP, інтерактивні віддалені термінальні сеанси та навіть веб-сеанси, оскільки на продуктивність впливає мережева затримка.

Лістинг 6.5 – Псевдокод алгоритму Min CUBIC з мінімальною затримкою

```
Ініціалізація:
δ ← 500 /* Період очікування, у мілісекундах */
reduce ← false /* Якщо істина, то зменшити cwnd */
threshold ← 100 /* Поріг затримки в мілісекундах */
Для кожного ACK:
baseRTT ← min(baseRTT, rtt)
β ← 0.8*baseRTTrtt/* β, розраховане для того, щоб очистити чергу
*/
if now > waitTime then
  if reduce then
    cwnd ← β*cwnd /* β – параметр зменшення */
    ssthresh ← cwnd
    waitTime = now + δ
   reduce ← false
  else if rtt – baseRTT > threshold then
    reduce ← true
    waitTime ← now + δ
  else
    /* збільшити вікно перевантаження за допомогою правила CUBIC
*/
  end if
else
  /* залишити cwnd без змін */
end if

При втраті пакету:
cwnd ← β*cwnd /* β – параметр зменшення */
ssthresh ← cwnd
```

Крім того, він демонструє покращену короткострокову справедливість порівняно з алгоритмом CUBIC із максимальною затримкою.

Нарешті, він показує суттєво покращену ефективність порівняно з TCP NewReno, хоча й має незначну втрату ефективності порівняно з оригінальним протоколом CUBIC.

7 РЕЗУЛЬТАТИ МОДЕЛЮВАННЯ

7.1 Топологія та параметри мережі

Використовується Network Simulator 2 (NS-2) для створення кількох мережевих сценаріїв. Моделювання мережі в NS-2 проводиться за допомогою сценаріїв OTcl, які потім інтерпретуються та виконуються NS-2. Використовується утиліта Gnuplot для побудови графіків результатів моделювання.

Використовуючи топологію мережі, показану на рисунку 7.1, можна визначити кілька параметрів, таких як пропускна здатність каналу, затримка передачі, прихід і вихід потоків в мережі, відповідно до сценарію моделювання. Деякі з цих параметрів також можна змінити під час моделювання, щоб імітувати динамічні умови мережі.

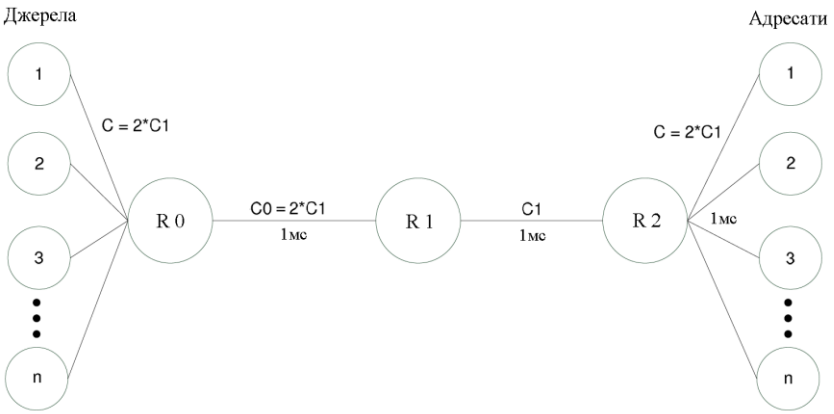

Рисунок 7.1 – Топологія експериментальної мережі

У визначеній топології мережі маршрутизатор 0 – це маршрутизатор, до якого підключаються всі вихідні вузли, маршрутизатор 1 – це маршрутизатор з вузьким місцем мережі, а маршрутизатор 2 – це маршрутизатор, до якого підключаються вузли-приймачі.

Під час моделювання кожен вихідний вузол передає дані відповідному вузлу адресату. C0 – пропускна здатність між маршрутизаторами 0 і маршрутизаторами 1, встановлюється на подвійне значення C1, яке є пропускною здатністю маршрутизатора вузького місця. Якщо не вказано інше, затримки встановлено в 1 мс. У мережевих сценаріях, де потрібні різнорідні затримки поширення, встановлюються значення затримки для кожного вихідного вузла.

Щоб оцінити продуктивність алгоритмів, вимірюються шість ключових показників продуктивності з інтервалом в одну секунду. Це пропускна спроможність, справедливість, ефективність каналу, швидкість скидання пакетів, зайнятість черги та час проходження в обидві сторони.

Спочатку оцінюється ефективність алгоритмів. Ефективність пов’язана з використанням каналу зв’язку, тому високошвидкісний варіант TCP повинен мати можливість використовувати доступну пропускну здатність мереж. Ефективність каналу вимірюється на маршрутизаторі з вузьким місцем (маршрутизатор 1), спостерігаючи за швидкістю виведення каналу. По-друге, вимірюється справедливість. Очікується, що потоки, які конкурують за пропускну здатність у зв’язку з вузьким місцем, зрештою прийдуть до справедливої частки пропускної здатності. Справедливість потоку оцінюється за різними сценаріями, наприклад, потоки з різними RTT. Справедливість вимірюється за індексом справедливості Джайна:

$$F(x)=\frac{\left(\sum x_i\right)^2}{n\left(\sum x_i^2\right)}, \qquad (7.1)$$

де x_i – виміряна швидкість передачі потоку і в інтервалі часу. Індекс справедливості обмежений між 0 і 1. Загальний справедливий розподіл (усі x_i рівні) має справедливість 1; оскільки розподіл смуги пропускання стає більш несправедливим, значення справедливості відповідно зменшується. Нарешті, дружність до мережі визначається шляхом перевірки рівня втрат пакетів,

затримки в черзі та зайнятості черги маршрутизатора. Зручність мережі є бажаною характеристикою алгоритму контролю перевантажень, оскільки вона зменшує навантаження на мережу та покращує продуктивність чутливих до затримок програм, таких як VoIP та інтерактивні сеанси. Швидкість відкидання пакетів обчислюється на вузькому маршрутизаторі за допомогою $L_{rate} = P_d/P_a$, де P_d – загальна кількість пакетів, відкинутих у вузькому маршрутизаторі, а P_a – загальна кількість пакетів, що надійшли на вузький маршрутизатор. Зайнятість черги обчислюється на маршрутизаторі з вузьким місцем шляхом спостереження за кількістю пакетів, які наразі буферизуються для передачі.

Щоб оцінити алгоритми, реалізувано алгоритми Max-delay CUBIC і Min-delay CUBIC в симуляторі NS-2. Крім того, пейсінг було реалізовано шляхом перенесення попереднього патча NS-2 [32] на останню версію NS-2. Реалізацію NS-2 CUBIC було взято з офіційної сторінки CUBIC [33]. Нарешті, реалізація NS-2 FAST TCP була взята з [34].

Таблиця 7.1 – Параметри моделювання

Параметри		Події
Джерел	1-50	t = 0 с старт 1 джерела
C0 (Мбіт/с)	500-2000	t = 300-3600 с час симуляції
C1 (Мбіт/с)	250-1000	
RTT (мс)	25, 160	
Довжина черги	100% BDP	
MSS	1460 Байт	

7.2 Ефективність передачі одного потоку

Досить популярний тест ефективності алгоритму контролю перевантажень полягає в спостереженні за тим, скільки даних можна передати одним потоком у каналі зв’язку зі значною затримкою пропускної

здатності. Дотримуючись цієї ідеї, щоб оцінити ефективність алгоритмів, моделюється мережевий канал 1 Гбіт/с із значенням затримки поширення в обидва кінці 160 мс, пропускною здатністю вузького місця в 1 Гбіт/с и часом симуляції в 30 хвилин.

Оцінюються результати ефективності каналу для алгоритмів Max CUBIC і Min CUBIC та FAST TCP відносно базового алгоритму TCP CUBIC. Крім того, ми також перевіряється підваріант CUBIC з максимальною затримкою з NewReno (Max CUBIC+NR), щоб побачити, чи зможе він досягти своєї основної мети: дуже добре використання каналу.

На рисунку 7.2 показано, що CUBIC і Max CUBIC з максимальною затримкою мають чудову ефективність з'єднання, повністю використовуючи вузьке місце. Min CUBIC з мінімальною затримкою, як і очікувалося, має нижчу продуктивність.

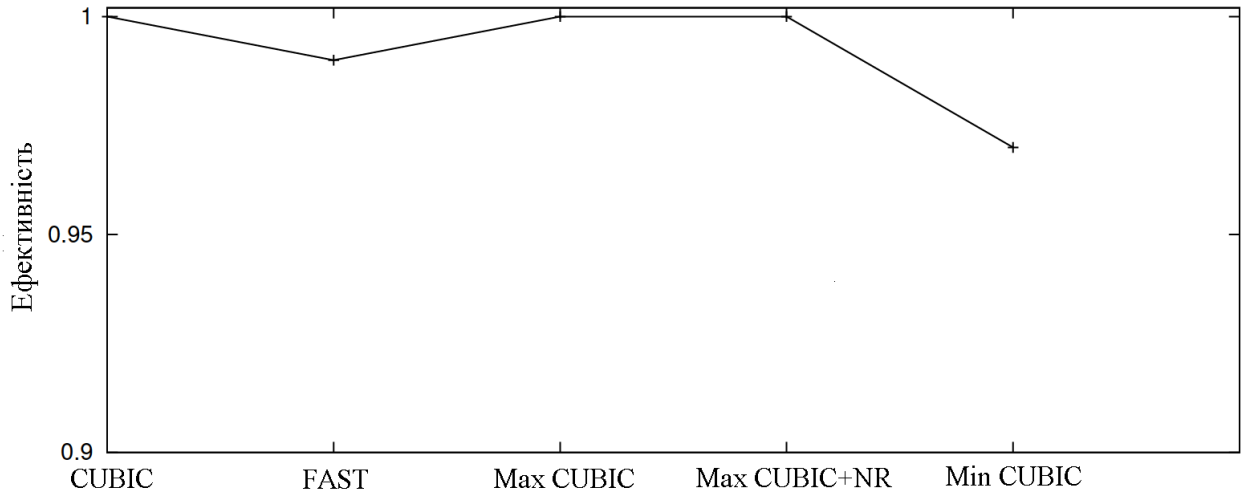

Рисунок 7.2 – Ефективність передачі одного потоку

CUBIC і CUBIC з максимальною затримкою з NewReno змогли передати найбільшу кількість даних протягом 30 хвилин моделювання. Це пояснюється тим, що в обох алгоритмах не використовується механізм пейсингу, і тому їхня поведінка повільного старту є більш агресивною, ніж поведінка повільного старту варіанта, який використовує механізм пейсингу.

7.3 Справедливість RTT

Два конкуруючих потоки зі значною різницею RTT мають спільний канал 250 Мбіт/с. Потоки 1 і 2 мають 160 мс і 25 мс затримки двостороннього поширення відповідно.

Обидва потоки використовують алгоритм CUBIC з максимальною затримкою з увімкненим механізмом виявлення виснаження черги, щоб пом’якшити короткострокову несправедливість. Черга маршрутизатора з вузьким місцем встановлена на 100% BDP. При t = 0 с у моделюванні потік 1 починає передачу на повній швидкості. При t = 60 с потік 2 входить в мережу і також починає передачу на повній швидкості. Загальний час моделювання 10 хвилин. Параметри моделювання: пропускна здатність вузького місця 250 Мбіт/с.

На рисунку 7.3 показано результати моделювання з вимкненим механізмом покращення справедливості RTT.

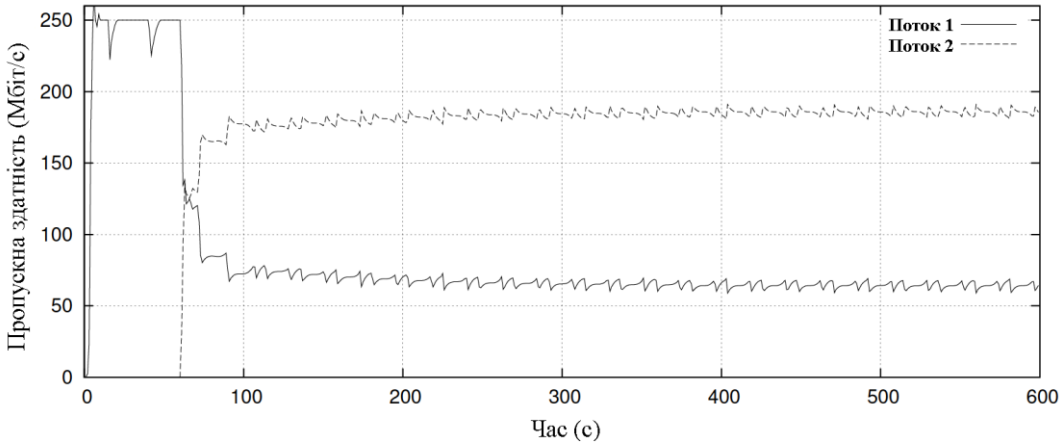

Рисунок 7.3 – Потоки з вимкненим механізмом справедливості RTT

На рисунку 7.4 показано результати з увімкненим механізмом покращення справедливості RTT. Вікно перевантаження потоку 2 штрафується через його коротший RTT. Це призводить до покращення справедливості RTT, як показано на рисунку 7.4.

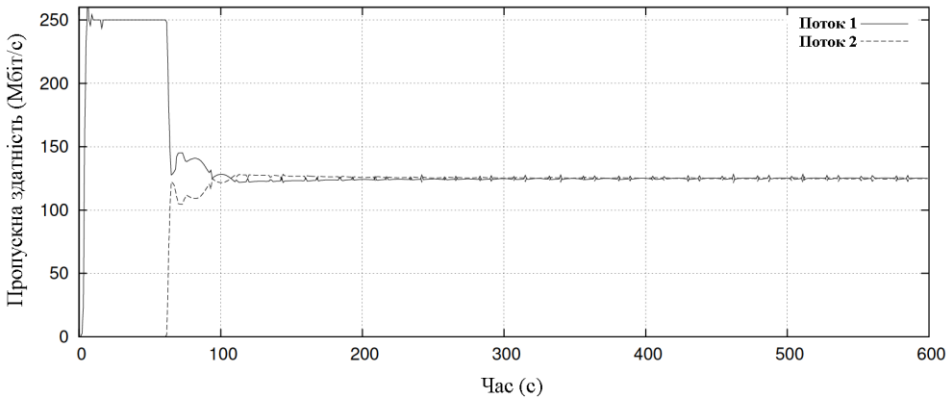

Рисунок 7.4 – Потоки з увімкненим механізмом справедливості RTT

7.4 Короткострокова поведінка

Моделюється поступове входження п'яти потоків у мережу. Мета цього експерименту полягає в тому, щоб спостерігати за тим, як кілька алгоритмів реагують протягом періоду нестабільності мережі, і як після цього перехідного періоду вони врешті-решт наближуються до справедливого розподілу частки доступної пропускної здатності. Використані параметри для цього моделювання:

- 5 потоків стартують через 10 секонд кожний;
- пропускна здатність вузького місця 250 Мбіт;
- затримка 25 і 160 мс.

На рисунку 7.5 показана продуктивність, досягнута п'ятьма одночасними потоками TCP CUBIC.

Потрібно приблизно 2 хвилини, починаючи з моменту входу потоку 5, за t = 40 с, щоб усі п’ять потоків зійшлися до розумно справедливого значення пропускної здатності.

Протягом цього часу потік 1 отримує несправедливу перевагу пропускної здатності над потоками, які входять пізніше в мережу. Ближче до кінця симуляції потоки показують деякі проблеми з підтриманням справедливого розподілу пропускної здатності.

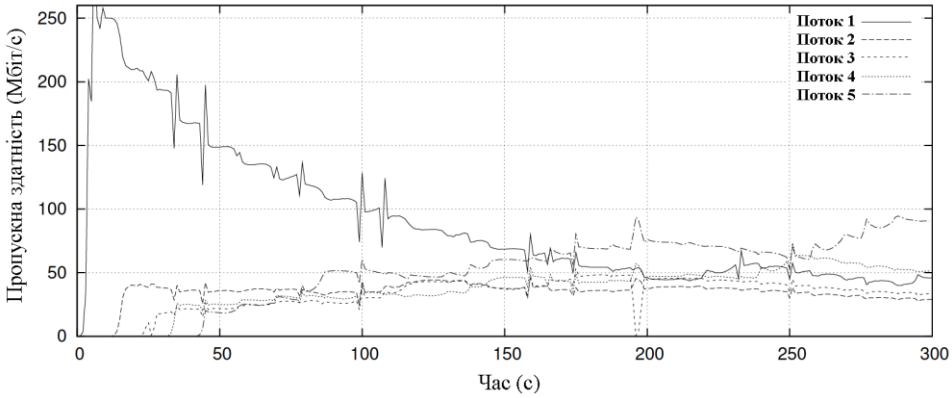

Рисунок 7.5 – Розподіл пропускної здатності для 5 потоків з алгоритмом CUBIC

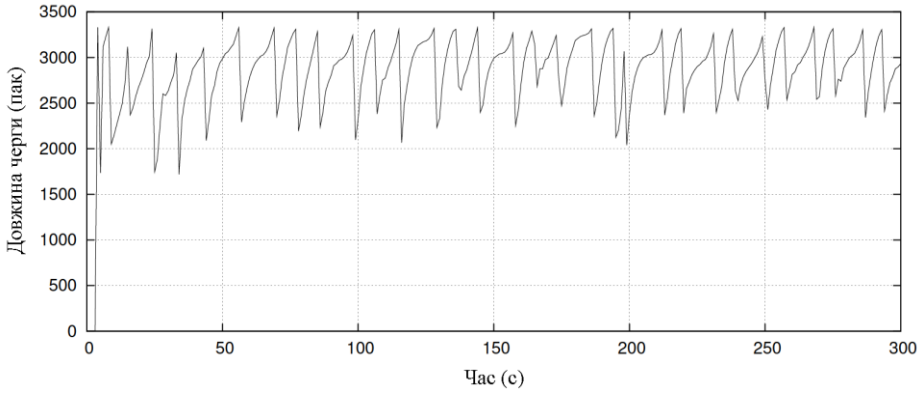

Рисунок 7.6 – Довжина черги вузького місця з 5 потоками та алгоритмом CUBIC

Max CUBIC з максимальною затримкою RTT може мати різну короткочасну поведінку залежно від вибраного порогового значення затримки. Проводиться тестування з максимальною затримкою CUBIC з порогом затримки 100 мс і 10 мс. Крім того, тестується Max CUBIC з максимальною затримкою RTT із короткостроковим покращенням справедливості, використовуючи поріг затримки в 100 мс, щоб оцінити, чи можливо зберегти більше порогове значення, одночасно пом’якшуючи короткострокову несправедливість, спричинену новими потоками, що надходять у мережу.

Для Max CUBIC з максимальною затримкою RTT з порогом 100 мс на рисунку 7.7 чітко видно, що нові потоки набувають смуги пропускання за рахунок старих потоків. Це протилежно тому, що відбувається з CUBIC (рисунок 7.5), де перший потік повільно звільняє пропускну здатність для нових потоків. Поведінка Max CUBIC з максимальною затримкою є очікуваною, оскільки це, новим потокам надається тимчасовий підвищений пріоритет.

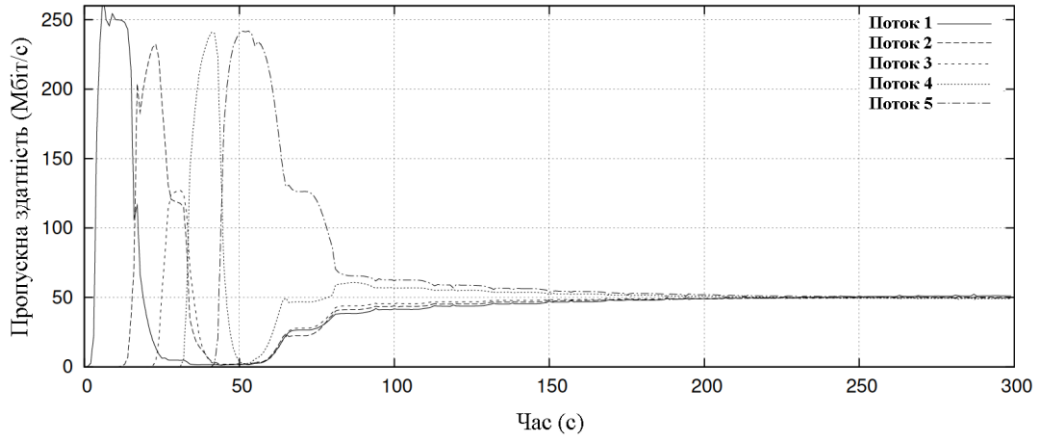

Рисунок 7.7 – Розподіл пропускної здатності для 5 потоків з алгоритмом Max CUBIC (поріг 100 мс)

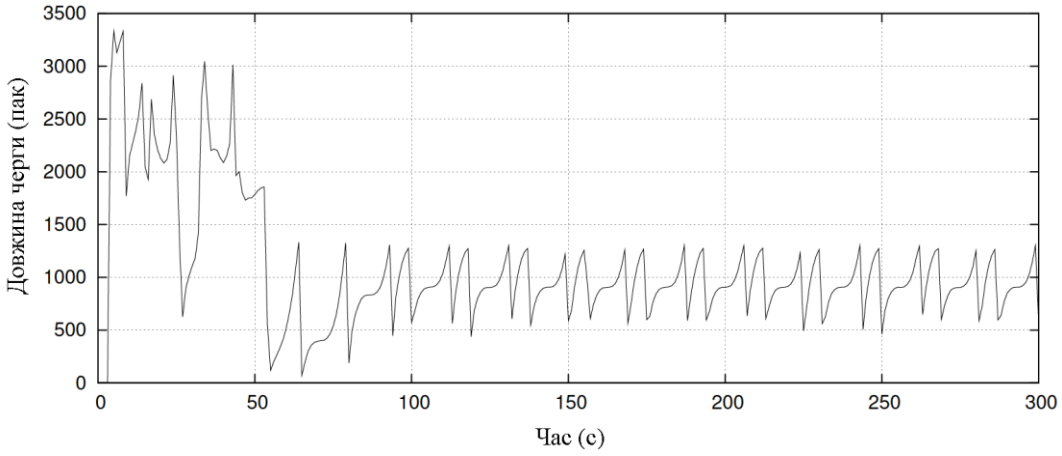

Рисунок 7.8 – Довжина черги вузького місця з 5 потоками та алгоритмом Max CUBIC (поріг 100 мс)

На рисунку 7.9 показано пропускну здатність для п'яти потоків Max CUBIC з максимальною затримкою з використанням порогового значення затримки 10 мс. Можна спостерігати, що зменшення значення порогу затримки до 10 мс призводить до іншої динаміки короткострокової справедливості, коли потоки збігаються швидше до справедливого розподілу частки пропускної здатності каналу. Причина полягає в тому, що при використанні порогу затримки в 10 мс потоки на основі затримки реагують пізніше на сигнали затримки в черзі.

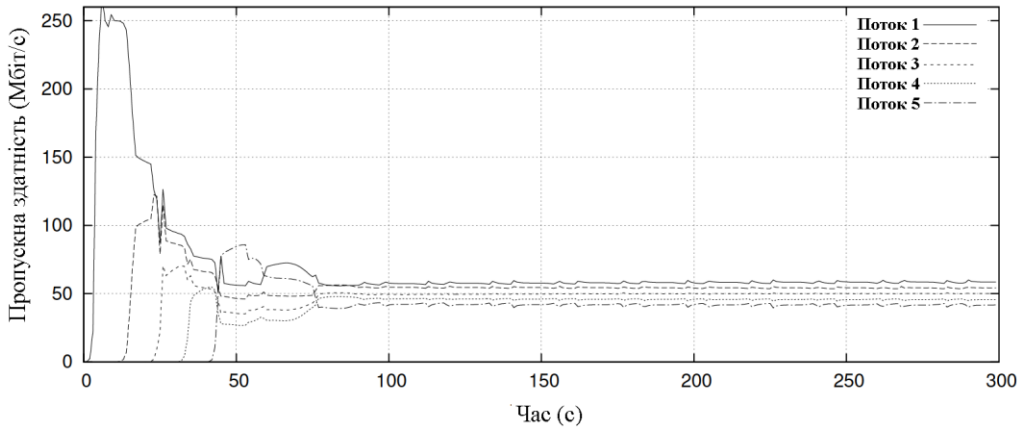

Рисунок 7.9 – Розподіл пропускної здатності для 5 потоків з алгоритмом Max CUBIC (поріг 10 мс)

Отримані результати вказують на те, що менший поріг затримки призводить до короткочасного покращення справедливості, коли старіші потоки на основі затримки конкурують за пропускну здатність з новими потоками на основі втрат. Крім того, ми зазначаємо, що досягнута довгострокова продуктивність справедливості дещо поступається довгостроковій продуктивності справедливості, досягнутої, коли порогове значення затримки встановлено на 100 мс.

На рисунку 7.10 показано, що, оскільки порогове значення затримки менше, Max CUBIC зменшує свою перевантаженість ближче до ліміту черги, що призводить до більшого заповнення черги. Ці результати вказують на те,

що налаштування порогового значення затримки значно впливає на короткострокову динаміку справедливості, а також на довгострокову зручність мережі, наприклад, на зайнятість черги.

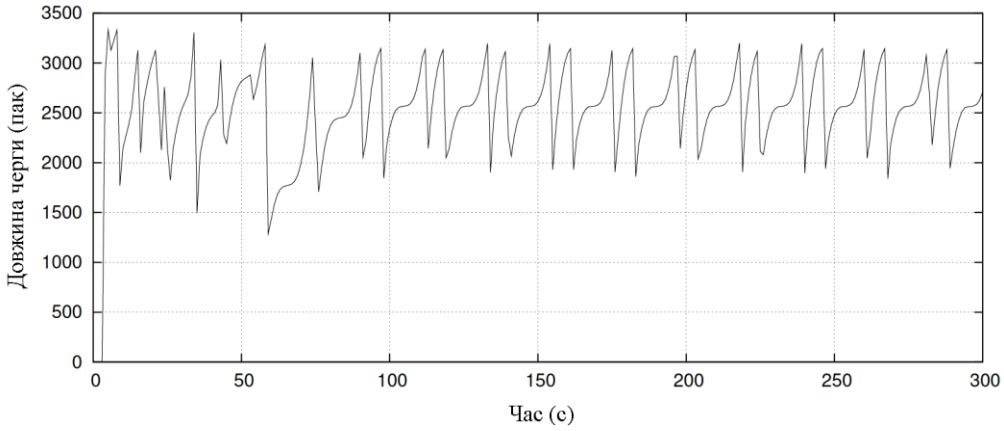

Рисунок 7.10 – Довжина черги вузького місця з 5 потоками та алгоритмом Max CUBIC (поріг 10 мс)

Max CUBIC із покращенням короткострокової справедливості намагається увімкнути використання вищого порогового значення затримки в Max-delay CUBIC, зберігаючи при цьому сумісність із потоками на основі втрат і покращуючи короткострокову справедливість. На рисунку 7.11 можна бачити, що старі потоки не відразу зменшують свою пропускну здатність у присутності нових потоків, незважаючи на те, що вони мають порогове значення затримки 100 мс. Це призводить до кращої короткострокової справедливості у порівнянні з короткостроковою справедливістю Max CUBIC з таким самим пороговим значенням.

Що стосується зайнятості черги, рисунок 7.12 показує, що приблизно при t = 50 с потоки починають координувати свою поведінку, щоб знизити час зворотного проходження нижче порогового значення затримки в 100 мс. Спочатку потоки заморожують своє вікно перевантаження, а потім повільно його зменшують, що призводить до нахилу черги з 50 до 60 с. Після цього періоду вмикається основний алгоритм CUBIC з максимальною затримкою.

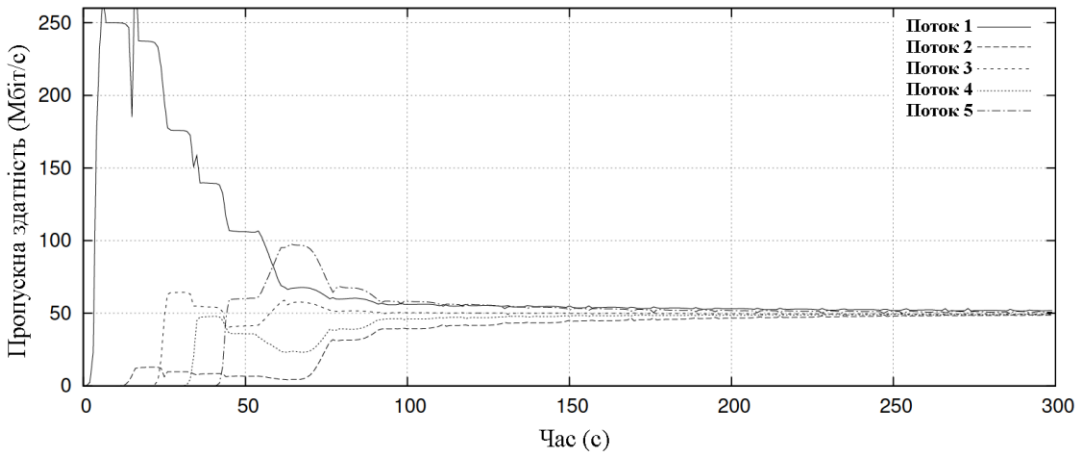

Рисунок 7.11 – Розподіл пропускної здатності для 5 потоків з Max CUBIC з короткочасним покращенням справедливості (поріг 100 мс)

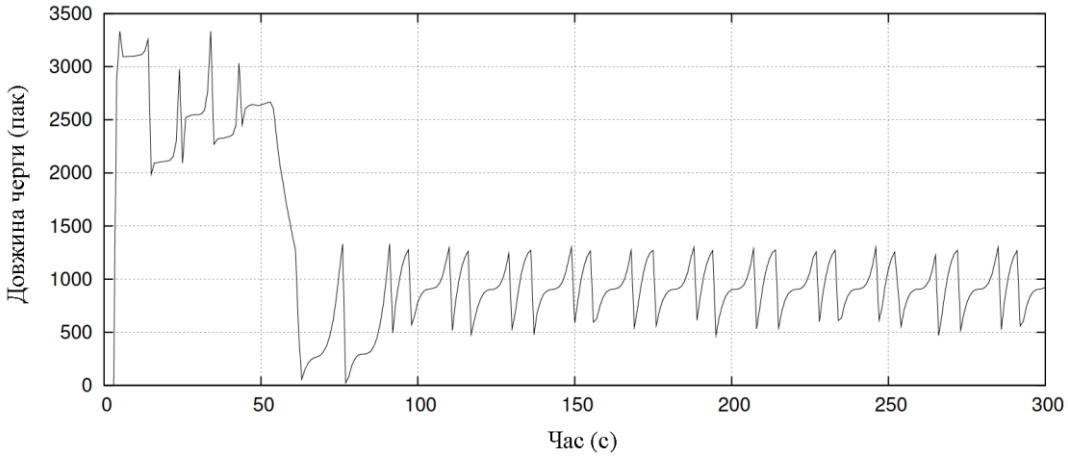

Рисунок 7.12 – Довжина черги вузького місця для 5 потоків з Max CUBIC з короткочасним покращенням справедливості (поріг 100 мс)

На рисунку 7.13 показано, що збіжність пропускної здатності Min CUBIC з мінімальною затримкою значно покращилася порівняно з CUBIC.

Крім того, рисунок 7.13 також показує, що його механізм повільного старту є менш агресивним, ніж повільний старт, який використовується CUBIC, і варіантами CUBIC з максимальною затримкою. Це пов’язано з використанням обмеженого повільного старту, який використовується для уникнення втрати пакетів.

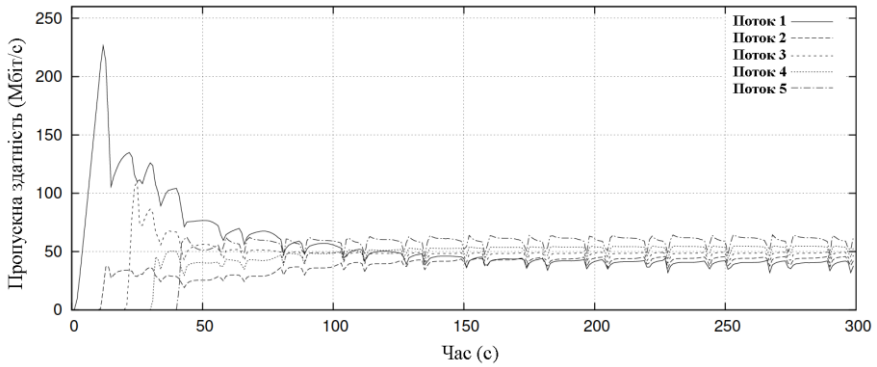

Рисунок 7.13 – Розподіл пропускної здатності для 5 потоків з алгоритмом Min CUBIC

Таким чином, на рисунку 7.14 показано, що цей варіант жертвує продуктивністю заради ефективності через механізм спорожнення черги. На рисунку видно періоди нульового заповнення черги.

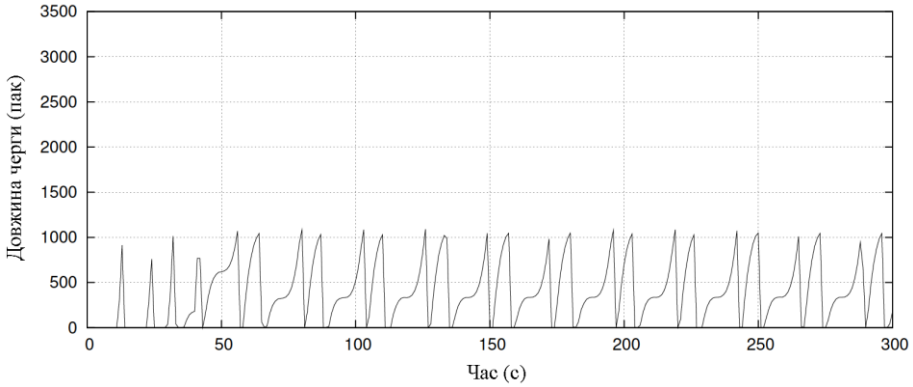

Рисунок 7.14 – Довжина черги вузького місця з 5 потоками та алгоритмом Min CUBIC

7.5 Динамічний сценарій

У цьому сценарії спостерігається поведінка алгоритмів CUBIC, максимальної затримки CUBIC і мінімальної затримки CUBIC. Щоб досягти цього, моделюється низка подій, які змінюють стан мережі, а саме, поява і

зникнення потоків, а також зміна пропускної здатності маршрутизатора вузького місця. Основна мета – оцінити, як різні алгоритми адаптуються до мінливих умов мережі.

На рисунку 7.15 показано, що потокам CUBIC важко досягти стабільної швидкості пропускання, що призводить до погіршення продуктивності справедливості.

Відбуваються події перевантаження, коли черга маршрутизатора переповнюється, але це впливає лише на підмножину потоків. Без пейсингу пакети мають тенденцію групуватися в чергах маршрутизаторів. Таким чином, коли черга переповнюється, скинуті пакети можуть належати лише до невеликої кількості потоків у каналі.

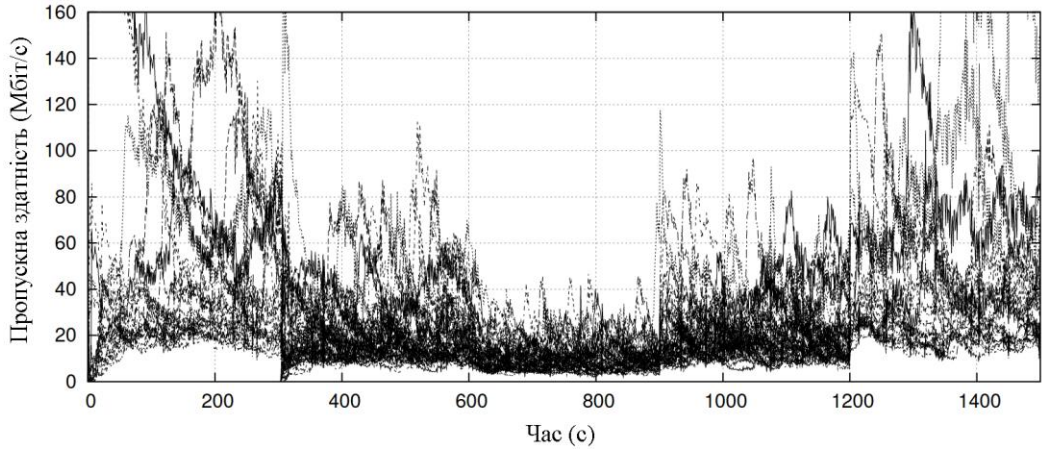

Рисунок 7.15 – Розподіл пропускної здатності для 50 потоків з алгоритмом CUBIC

Результати для Max CUBIC з використанням максимальної затримки показано на рисунку 7.16. Спочатку деякі потоки набувають більшої пропускної здатності порівняно з іншими потоками. Це тому, що Max CUBIC використовує звичайний механізм повільного старту. Як згадувалося раніше, повільний запуск призводить до того, що потоки з коротшими RTT будуть набагато агресивнішими, ніж потоки з довшими RTT. Тим не менш, після цього початкового перехідного періоду можна побачити, що потоки зрештою

стабілізуються навколо нормальної пропускної здатності. Така ж поведінка знову спостерігається при t = 300 с, коли в мережу надходять додаткові 25 потоків. Загалом ми бачимо, що потоки мають значно менше коливань порівняно з потоками CUBIC.

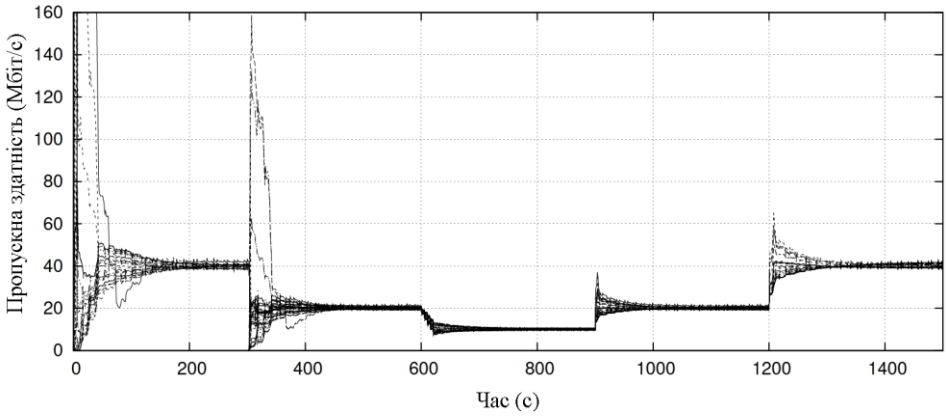

Рисунок 7.16 – Розподіл пропускної здатності для 50 потоків з алгоритмом Max CUBIC

Результати для Min CUBIC з використанням мінімальної затримки показано на рисунку 7.17. Потоки Min CUBIC з мінімальною затримкою мають значно більш стабільну траєкторію пропускної здатності, ніж CUBIC.

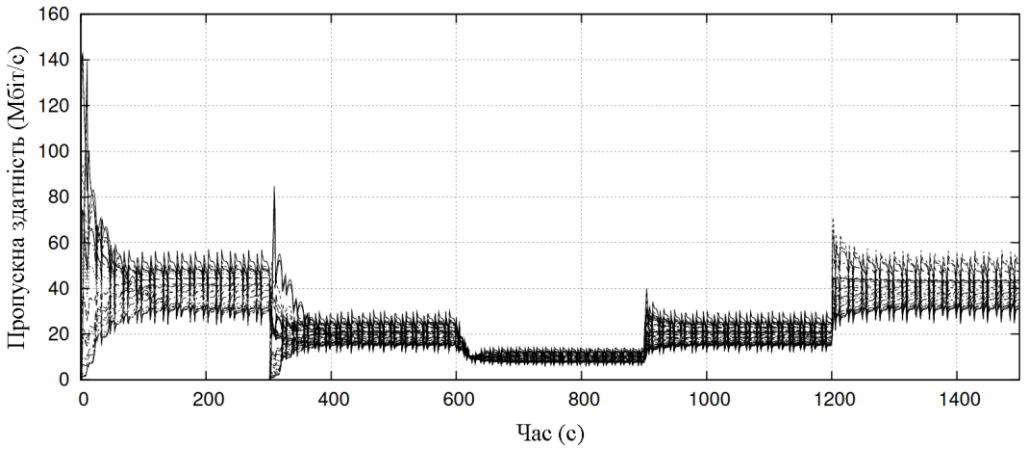

Рисунок 7.16 – Розподіл пропускної здатності для 50 потоків з алгоритмом Min CUBIC

Його довгострокові показники справедливості, однак, поступаються показникам справедливості Max-delay CUBIC (рисунок 7.16). Використовуючи обмежений повільний запуск, усі потоки збільшують своє вікно перевантаження повільніше, ніж зі звичайним експоненційним повільним запуском. Крім того, завжди залишаючись у режимі на основі затримки, потоки можуть залишити (обмежений) повільний запуск перед тим, як переповнити черги маршрутизатора, і почати сходитися раніше.

Тести оцінювали продуктивність різних алгоритмів, головним чином за показниками ефективності, справедливості, зайнятості черги. Результати показують, що в змодельованих сценаріях Max CUBIC з використанням максимальної затримки і Min CUBIC з використанням мінімальної затримки можуть значно підвищити продуктивність початкового алгоритму CUBIC.

ВИСНОВКИ

Загальновідомо, що алгоритми управління перевантаженням TCP погано працюють у мережах наступного покоління. Джерела TCP адаптують своє вікно перевантаження, використовуючи мережеві сигнали, наприклад, втрати пакетів і зміни затримки. Однак поява мереж наступного покоління показала обмеження консервативного підходу до контролю перевантажень TCP, створюючи кілька проблем, таких як зниження ефективності з'єднання в гетерогенних мережах.

У ході виконання кваліфікаційної роботи представлені нові наскрізні методи управління перевантаженням в протоколі TCP, які можуть значно покращити популярний високошвидкісний наскрізний алгоритм на основі втрат.

Під час проведення імітаційного моделювання оцінювалася продуктивність різних алгоритмів, головним чином за показниками ефективності, справедливості, зайнятості черги і масштабованості. Отримані результати показують, що в змодельованих мережевих сценаріях алгоритми Max CUBIC з використанням максимальної затримки і Min CUBIC з використанням мінімальної затримки можуть значно підвищити продуктивність широко застосованого алгоритму CUBIC.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lv Guanqiao, Liu Hanbing, Deng Xiaohong. Design of Network Congestion Control Algorithm Based on TCP Protocol. Software Guide, 2014,13(01):56-59

2. Li Yinghua, Cui Jiarong. Research Review of TCP Congestion Control Algorithm Based on Packet Loss and Network Measurement. DigitalCommunicationWorld, 2022(04):9-11

3. Wang Guodong, Ren Yongmao. Li Jun. Performance evaluation of TCP congestion control algorithms in fast long distance network[J]. Journal on Communications,2014,35(4) : 81-90

4. Lal Pratap Verma and Indradeep Verma and Mahesh Kumar. An Adaptive Congestion Control Algorithm[J]. MMC_A, 2019, 92(1) : 30-36

5. Vinton G. Cerf and Robert E. Icahn. A protocol for packet network intercommunication. SIGCOMM Comput. Commun. Rev., 35(2):71–82, April 2005.

6. V. Cerf, Y. Dalal, and C. Sunshine. Specification of Internet Transmission Control Program. RFC 675, December 1974.

7. M. Bateman and S. Bhatti. Tcp testing: How well does ns2 match reality? In Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on, pages 276–284, april 2010.

8. K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard), September 2001. Updated by RFCs 4301, 6040.

9. A. Kuzmanovic, A. Mondal, S. Floyd, and K. Ramakrishnan. Adding Explicit Congestion Notification (ECN) Capability to TCP’s SYN/ACK Packets. RFC 5562 (Experimental), June 2009.

10. J. Postel. Transmission Control Protocol. IETF RFC 793, September 1981.

11. V. Jacobson. Congestion Avoidance and Control. In proc. ACM SIGCOMM, pages 314–329, Stanford, CA, August 1988.

12. S. Molnár, B. Sonkoly, and T. A. Trinh. A Comprehensive TCP Fairness Analysis in High Speed Networks. Computer Communications, 32(13-14):1460–1484, August 2009.

13. S. Floyd and K. Fall. Promoting the Use of End-to-End Congestion Control in the Internet. IEEE/ACM Transactions on Networking, 7:458–472, 1999.

14. S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. IEEE/ACM Transactions on Networking, 1(4):397–413, August 1993.

15. K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Noti- fication (ECN) to IP. IETF RFC 3168, September 2001.

16. A. Falk and D. Katabi. Specification for the Explicit Control Protocol (XCP). IETF Internet Draft, November 2006.

17. M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. IETF RFC 2581, April 1999.

18. S. Floyd and T. Henderson. The NewReno Modification to TCP's Fast Recovery Al- gorithm. IETF RFC 2582, April 1999.

19. M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. IETF RFC 2582, October 1996.

20. S. Floyd. HighSpeed TCP for Large Congestion Windows. IETF RFC 3649, December 2003.

21. T. Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks. Computer Communication Review, 33(2):83–91, April 2003.

22. D. Leith and R. Shorten. H-TCP Protocol for High-Speed Long Distance Networks. In proc. International Workshop on Protocols for Fast Long-Distance Networks, Argonne, Illinois, USA, February 2004.

23. C. Casetti, M. Gerla, S. Mascolo, M. Sansadidi, and R. Wang. TCP Westwood: End- to-End Congestion Control for Wired/Wireless Networks. Wireless Networks Journal, 8:467–479, 2002.

24. R. Wang, K. Yamada, M. Sanadidi, and M. Gerla. TCP With Sender-Side Intelligence to Handle Dynamic, Large, Leaky Pipes. IEEE Journal on Selected Areas in Commu- nications, 23(2):235–248, February 2005.

25. L. Xu, K. Harfoush, and I. Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks. In INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, volume 4, pages 2514–2524 vol.4, March 2004.

26. S. Ha, I. Rhee, and L. Xu. CUBIC: a New TCP-Friendly High-Speed TCP Variant. SIGOPS Operating Systems Review, 42(5):64–74, 2008.

27. D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. IEEE/ACM Transactions on Networking, 14(6):1246–1259, 2006.

28. S. Liu, T. Başar, and R. Srikant. TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks. In proc. International Conference on Per- formance Evaluation Methodologies and Tools (VALUETOOLS), Pisa, Italy, October 2006.

29. K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High- speed and Long Distance Networks. In proc. IEEE INFOCOM, Barcelona, Spain, April 2006.

30. X. Wu, M.C. Chan, A.L. Ananda, and C. Ganjihal. Sync-TCP: A New Approach to High Speed Congestion Control. In Network Protocols, 2009. ICNP 2009. 17th IEEE International Conference on, pages 181–192, October 2009.

31. S. Floyd. Limited Slow-Start for TCP with Large Congestion Windows. EETF RFC 3742, March 2004.

32. A TCP pacing implementation for NS-2. http://netlab.caltech.edu/projects/ ns2tcplinux/ns2pacing/index.html.

33. BIC and CUBIC Web page. http://netsrv.csc.ncsu.edu/twiki/bin/view/Main/ BIC.html.

34. FAST TCP simulator module for NS-2 Web page. http://www.cubinlab.ee.unimelb. edu.au/ns2fasttcp/.

35. Сокирко М.А., Чернов Д.Д., Порошенко А.І. Управління мережевим трафіком//Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління. Чотирнадцята міжнародна науково-технічна конференція. Баку – Харків – Жиліна. 2024 р. Том 1. С. 118.

36. Козін М.В., Сокирко М.А., Янковський О.А. Методи адаптації протоколу TCP до поточного мережевого стану//Збірник тез доповідей дванадцятої міжнародної науково-технічної конференції «Проблеми інформатизації». Баку, Харків, Бельсько-Бяла. 2024. С. 72.