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)

Моделі та методи оптимізації трафіку IP-мереж

(тема)

Виконав:
студент II курсу, групи КСМм-21-1
Харченко О.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)

2022 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 Харченку Олексію Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі та методи оптимізації трафіку IP-мереж

затверджена наказом по університету від " 07 " листопада 2022 р. № 1453 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2022р.

3. Вхідні дані до роботи 1) моделі та методи для керування мережевими інформаційними потоками; 2) сучасні вимоги до мережних показників; 3) перелік використаних програмних та апаратних засобів: ОС Windows 10, OpNet 14, NS-3.

4. Перелік питань, що потрібно опрацювати в роботі

1) аналіз сучасного стану проблеми

2) огляд технологій управління перевантаженням та чергами маршрутизаторів

3) моделі управління мережним трафіком

4) вибір програмних та апаратних засобів реалізації

5) проведення експериментальних досліджень

6) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Слайдів презентації –15 шт.

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	08.11.22–09.11.22	
	вирішення		
2	Огляд технологій управління перевантаженням	10.11.22–15.11.22	
3	Розробка моделі управління мережним	16.11.22 –23.11.22	
	трафіком		
4	Вибір програмних та апаратних засобів	24.11.22 –25.11.22	
	реалізації		
5	Тестування запропонованого метода	26.11.22–04.12.22	
6	Оформлення пояснювальної записки	05.12.22–-11.12.22	

Дата видачі завдання 07 листопада 2022 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 91 с., 26 рис., 1 табл., 1 дод., 16 джерел.

АДИТИВНЕ ЗБІЛЬШЕННЯ/МУЛЬТИПЛІКАТИВНЕ ЗМЕНШЕННЯ, АКТИВНЕ УПРАВЛІННЯ ЧЕРГОЮ, ПЕРЕВАНТАЖЕННЯ, ПРОТОКОЛ, СКИДАННЯ ПАКЕТІВ, ТАЙМ-АУТ, ТОПОЛОГІЯ.

Перевантаження є ключовою темою в комп'ютерних мережах, яка активно вивчається через її прямий вплив на продуктивність мережі. Одним із широко досліджених методів контролю перевантажень є випадкове раннє виявлення (RED). Щоб підтримувати продуктивність RED для отримання бажаних результатів, зазвичай налаштовують вхідні параметри, особливо максимальну ймовірність відкидання пакетів, до певних значень. Налаштування цього параметра на ці значення призводить до хороших, але упереджених результатів продуктивності. У кваліфікаційній роботі запропоновано RED_E алгоритм для вирішення цієї проблеми шляхом експоненціального відкидання пакетів, що надходять, без використання максимальної ймовірності відкидання пакетів.

ABSTRACT

Master’s thesis: 91 pages, 26 figures, 1 tables, 1 appendices, 16 sources.

ADDITIVE INCREASE/MULTIPLICATIVE DECREASE, ACTIVE QUEUE MANAGEMENT, OVERLOAD, PROTOCOL, PACKET DROP, TIMEOUT, TOPOLOGY.

Congestion is a key topic in computer networking that is heavily studied because of its direct impact on network performance. One widely researched congestion control method is random early detection (RED). In order to keep the performance of RED to obtain the desired results, it is common to tune the input parameters, especially the maximum packet drop probability, to certain values. Setting this parameter to these values produces good but biased performance results. In the qualification work, the RED_E algorithm is proposed to solve this problem by exponentially dropping incoming packets without using the maximum probability of dropping packets.

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)

AQM – активне управління чергою (англ., Active queuing mechanism)

ARED –адаптивне випадкове раннє виявлення (англ., Adaptive random early detection)

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

EWA – експоненціальне середньозважене (англ., Exponential weighted average)

FIFO – першим прибув, першим обслужений (англ., First In First Out)

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

RED – випадкове раннє виявлення (англ., Random Early Detection)

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

Низька затримка стала нескінченною вимогою якості обслуговування для сучасних програм. Хоча бітрейт Інтернету продовжує стрімко зростати в усьому світі, можна побачити таку ж тенденцію в напрямку затримки. Величезні затримки в Інтернеті впливають на життя майже кожного в усьому світі, і це вартість, пов’язана з просто перебуванням в мережі.

Вимога низької затримки не означає, що трафік споживає низьку пропускну здатність, потреба в низькій затримці стала важливою вимогою для всього жадібного трафіку, що працює через сучасний протокол керування передачею (TCP).

Розвиток сучасних додатків призвів до потреби в дуже низькій наскрізній затримці, де чим менша затримка, тим кращою та привабливішою є послуга для кінцевого користувача. Індустрія ігор і VoIP вже давно є одним із таких застосувань, але в сучасному світі потреби змінилася, і потреба в низькій затримці стала настільки ж важливою, якщо не більшою, ніж високий бітрейт буквально для всіх програм.

Сучасними додатками, яким задовольняє така потреба, може бути веб, обмін миттєвими повідомленнями, віртуальна/доповнена реальність, хмарні ігри, відеоконференції/потокове передавання, системи екстреної допомоги, віддалена допомога, дрони та багато, багато іншого.

Потреба, звичайно, лише верхівка айсберга, безсумнівно, потреба скоріш за все зростатиме й у майбутньому, а переваги меншої затримки залишатимуться ігнорованими, доки вони не почнуть бути частиною повсякденного життя.

Час проходження пакету туди й назад (RTT), який відчуває програма, складається в основному з незнижуваного базового RTT, який складається з великих фрагментів, таких як затримки передачі та поширення. Інша частина RTT складається з частин, які можуть бути значно скорочені. Однією з таких

частин є переважна затримка в чергах в маршрутизаторах і комутаторах на шляху між кінцевими хостами. Затримка в черзі зазвичай означає найбільш суттєву скорочувану частину RTT і була проблемою з самого початку Інтернету.

Схеми активного керування чергами (AQM) були запроваджені для вирішення величезної проблеми затримки в чергах і є окремим напрямком досліджень. Вузьке місце зроблено «розумним» для вирішення проблеми, тому воно або вилучає, або позначає пакети, які залишаються в черзі протягом більш тривалого часу. Тоді черга вузького місця залишається переважно короткою, а буфер, що залишився, поглинає сплески синхронізованого трафіку. В цьому випадку, AQM повністю контролює чергу та може вибрати порогове значення, достатньо високе, щоб підтримувати повне використання каналів зв’язку. Однак співіснування трафіку TCP і AQM, оптимізованого для підтримки неглибокої черги, погано поєднуються разом.

Проблема виникає в Інтернет-з’єднаннях, де потрібна низька швидкість передачі на RTT. Слід зазначити, що низька швидкість передачі не обов’язково означає низький бітрейт, натомість може означати низький базовий RTT. Імовірність виникнення проблеми в Інтернеті, як не дивно, вважається більш поширеною в сучасній топології мережі. Проблема наразі не існує в Інтернеті через пропорційно велику чергу на вузьке місце порівняно з наданим бітрейтом. Однак ця тенденція скоро закінчиться з розгортанням AQM неглибокої черги.

1 МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Алгоритми контролю заторів на маршрутизаторах є основними факторами успішної, ефективної та результативної роботи сучасних комп’ютерних мереж. Таким чином, метою цієї кваліфікаційної роботи є розробка нового алгоритму AQM, який має добре використання каналів зв’язку, має малий коефіцієнт втрат, потребує мало системних ресурсів та простий у конфігурації.

Виконання кваліфікаційної роботи передбачає:

- аналіз літературних джерел, присвячених проблемам збільшення пропускної здатності комп’ютерних мереж;
- проведення аналізу сучасних механізмів роботи протоколу TCP та алгоритмів AQM для боротьби з мережевими перевантаженнями;
- розробку метода управління чергою маршрутизатора відповідно до поточного мережевого стану;
- проведення аналізу результатів моделювання.

2 ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

Перевантаження є основною проблемою в сучасній мережі з комутацією пакетів. Коли кількість пакетів, трафік або навантаження в мережі перевищують пропускну здатність мережі, це призводить до перевантаження в мережі. Через перевантаження відбувається зниження пропускної здатності та збільшення затримки. Перевантаження погіршує продуктивність мережі, серйозно впливаючи на QoS.

Перевантаження погіршує якість обслуговування мережі, що призводить до затримки в черзі, втрати даних і блокування нового з'єднання. У перевантаженій мережі час відгуку сповільнюється зі зниженням пропускної здатності мережі. На рисунку 2.1 показано перевантаження в комп'ютерній мережі.

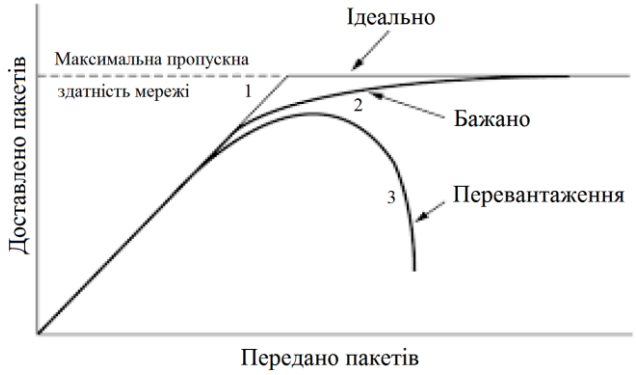

Рисунок 2.1 – Перевантаження в комп'ютерній мережі

Графік за номером 1 показує ідеальну поведінку комп'ютерної мережі під час збільшення навантаження, у той час, коли на виході доступна максимальна пропускна здатність. Графіки 2 відповідає випадку з наявністю механізмів управління мережевими перевантаженнями і наявності управління потоком. Графік 3 відповідає випадку відсутності механізмів управління перевантаженнями.

2.1 Протокол керування передачею (TCP)

Протокол керування передачею (TCP) – це стандартизований транспортний протокол, створений робочою групою Інтернету (IETF). TCP широко використовується через Інтернет-протокол (IP) і розвивався завдяки оновленням, наданим як запит на коментарі (RFC).

Реалізація TCP використовує змінний заголовок для зв’язку з іншою кінцевою точкою TCP, а дані програми додаються поверх цього заголовка. TCP, на відміну від альтернативного протоколу дейтаграм користувача (UDP), також має джерело, порт призначення та циклічну перевірку надмірності (CRC). Порт джерела та призначення забезпечують обмін даними між процесами, а CRC використовується для виявлення помилок передачі. У TCP використання CRC є обов’язковим як для відправника, так і для одержувача, а в UDP – необов’язковим.

UDP – це простий протокол, і якщо додаткам потрібні певні механізми, наявні в TCP, вони повинні бути реалізовані додатком або використовувати інші протоколи, які розширюють UDP.

Елементи мережі через Інтернет намагаються доставити пакети (наскрізний принцип), але немає жодних гарантій ні для доставки, ні для підтвердження, ні для передачі без помилок. TCP відновлює помилки шляхом впровадження повторної передачі втрачених сегментів, зміни порядку вхідних сегментів і використання циклічної перевірки надмірності (CRC). Це означає, що TCP забезпечує надійний і невпорядкований зв’язок між двома вузлами в мережі.

Втрата пакетів в мережах зазвичай відбувається через обмеження черги в мережевому елементі на шляху між відправником і одержувачем. Пакет також може бути пошкоджено під час передачі між двома посиланнями в мережі. Хоча пошкоджені пакети частіше трапляються під час бездротового зв’язку. Таким чином, TCP використовує номер підтвердження для щойно отриманих даних. TCP встановлює прапор ACK у заголовку TCP для

підтвердження даних. Цей прапорець використовується для вказівки того, що номер підтвердження в заголовку TCP є наступними очікуваними байтами в потоці. Підтвердження сповіщає відправника, що одержувач успішно отримав байти, надіслані перед номером підтвердження.

Сегменти можуть надходити в Інтернеті в неправильному порядку. Могла статися втрата сегмента, і наступний сегмент може надійти першим, створивши дірку в даних. Сегмент, отриманий у неправильному порядку, ще не може бути доставлений до відповідного додатку, оскільки мета TCP полягає в тому, щоб доставити дані в такому ж порядку, як їх надіслав відправник. Таким чином, TCP використовує порядкові номери в заголовку, щоб надавати дані в правильному порядку програмі-одержувачу. Крім того, приймаючі TCP використовують порядковий номер для вибору номера підтвердження [1].

Максимальний розмір сегмента (MSS) може бути змінений обома хостами під час ініціювання з’єднання, щоб замінити значення за замовчуванням; стандартним значенням є 536 байт. Значення MSS повідомляє хосту-відправнику, наскільки великі сегменти хост-одержувач готовий прийняти. Це правило обмежує лише кількість октетів/байтів, які надсилаються після змінних заголовків TCP (крім будь-якого використовуваного параметра TCP). Однак відправник не завжди може надіслати сегмент таким розміром через інші фактори, такі як MTU, виявлення MTU шляху або вікно потоку.

Таким чином, відправник може надсилати сегменти менші за MSS, відомі як максимальний розмір сегмента відправника (SMSS). Конфлікти виникають, оскільки обидва хости можуть мати різні вимоги до того, що відповідає повному сегменту. Одержувач використовує значення MSS для передачі підтвердження. Однак відправник може бути змушений надіслати менші сегменти, ніж MSS. Останній стандарт контролю перевантаження TCP не визначає, як одержувач повинен відповідати на такі пакети. Стандарт дозволяє одержувачу вільно вибирати відповідь.

Рисунок 2.2 – Заголовок TCP

2.2 Явне повідомлення про перевантаження (ECN)

Явне сповіщення про перевантаження (ECN) – це оновлення заголовка IP і TCP. У той момент, коли в елементі мережі закінчується буферний простір, відбувається відкидання пакетів для вхідних пакетів, і пакети продовжують відкидатися, доки попередні пакети не почнуть вилучатися з черги (відкидання хвоста FIFO).

З точки зору відправника, втрата пакетів є ознакою перевантаженості через час очікування. Недоліком використання лише втрати пакетів як ознаки перевантаження є величезна черга, яка накопичилася до того, як сталася подія втрати.

Черга стає величезною, оскільки відправлення пакетів йде занадто швидко протягом досить тривалого часу. ECN намагається вирішити сценарій величезної черги за допомогою явного сигналу перевантаження, але потрібна допомога від елементів мережі. Елемент мережі, який зараз є вузьким місцем на шляху, повинен дозволяти відправникам знати про перевантаження, щоб вони могли зменшити швидкість передачі, перш ніж черга буде переповнена. Елемент мережі, який активно намагається контролювати свою чергу, зазвичай відомий як Active Queue Management (AQM). AQM має залишатися без стану, щоб рішення було масштабованим.

IP-заголовок оновлюється двома бітами; ці біти використовуються, щоб визначити, чи підтримує кінцева точка ECN для цього конкретного пакету. Ці біти також використовуються AQM для сповіщення про перевантаження в черзі.

Хости встановлюють для цих бітів значення 2 (ECT(0)) або 1 (ECT(1)), щоб вказати, що хост підтримує ECN для цього пакета, рекомендованим підходом є використання ECT(0), щоб забезпечити його зворотну сумісність з попередньою версією ECN.

Крім того, AQM на шляху встановлює ці біти на значення 3 (виявлено перевантаження (CE)), коли пакети залишаються в буфері протягом більш тривалого періоду часу.

AQM встановлює ці біти, лише якщо хост підтримує використання ECN, іншими словами, значення цих бітів не повинно давати 0 (Not-ECT)). Причиною використання бітів для визначення того, чи підтримує хост ECN, є можливість поступового розгортання ECN; ця додаткова інформація дозволяє AQM прийняти правильне рішення щодо сумісних з ECN та інших хостів. Для хостів із підтримкою ECN позначки ECN достатньо, щоб сигналізувати про перевантаження, але лише втрата пакета дає такий самий ефект на інших хостах.

Таким чином, ця функція потрібна в мережах, де вона може співіснувати з ECN-Capable та іншими хостами.

Використання ECN у TCP є необов’язковим і вирішується між відправником і одержувачем, коли вони ініціюють з’єднання (сегменти з установленим SYN-бітом). Два біти використовуються в заголовку TCP для сигналізації ECN-Echo (ECE) і Congestion Window Reduced (CWR). Найперший сегмент, який ініціалізує з’єднання TCP, має біти ECE і CWR, які вказують, що відправник TCP підтримує ECN. Використання тут ECE-біту зобов’язує відправника відповідати на кодову точку CE, і так само відправник встановлює CWR-біт, щоб обіцяти зробити відповідне зниження швидкості передачі після отримання сегмента з установленим ECEbit.

Одержувач відповідає на перший сегмент сегментом SYN-ACK, де встановлено лише біт ECE, щоб повідомити відправнику, що він братиме участь як ECN-Capable хост. Хост не може змінити своє зобов'язання пізніше для цього TCP-з'єднання.

Одержувач встановлює біт ECN-Echo (ECE) на сегмент, щоб повідомити про перевантаження відправнику, і робить це шляхом перевірки IP-заголовка на наявність кодової точки CE.

Відправник передає наступний сегмент із встановленим бітом CWR, щоб підтвердити, що відправник отримав сегмент із встановленим бітом ECE, і таким чином підтвердити відповідне зниження швидкості передачі. Одержувач продовжує передачу будь-яких нових сегментів із встановленим бітом ECE, доки не буде отримано сегмент із відповідно встановленим бітом CWR.

Відправники тепер краще усвідомлюють, наскільки перевантажена мережа, і більше не спостерігають втрату пакетів через перевантаження. Відправник реагує на позначку ECN так само, як і на втрату пакета, а це зниження швидкості передачі вдвічі. Позначка ECN зазвичай запускається лише один раз у RTT, і це також те, як NewReno реагує на сигнали перевантаження.

2.3 Розширені або відкладені підтвердження

Розширені підтвердження використовуються реалізацією TCP, щоб зменшити кількість підтвердження, надісланих одержувачем. Розширене підтвердження означає, що приймаючий хост підтверджує кожен другий повний отриманий сегмент.

Використання розширеного підтвердження не є вимогою стандарту TCP, а лише рекомендацією для кращого використання мережі. Розширені підтвердження заощаджують пропускну спроможність і вартість обробки по всій мережі, оскільки на один RTT надходить менше контрольних пакетів.

Від одержувача вимагається тайм-аут розширеного підтвердження, щоб запобігти взаємоблокуванню в ситуаціях, коли відправник не може передати більше сегментів (рисунок 2.3 а).

Таким чином, стандарт TCP вимагає від одержувача не чекати більше 500 мс, перш ніж остаточно повернути підтвердження відправнику. Хоча більшість реалізацій запускає таймер не більше ніж на 40-200 мс. У цьому випадку отримувач передає підтвердження із затримкою.

Хоча одержувач усе ще повертає підтвердження негайно, як один сегмент даних, якщо обидва хости беруть участь у спілкуванні. Механізм відкладеного підтвердження є причиною, чому відправник намагається підтримувати вікно перевантаження щонайменше з двох цілих сегментів, або 2*байт SMSS, на RTT. Тоді відправник уникає відкладених підтверджень і, як правило, може очікувати надходження розтягнутих підтвердження в межах RTT (рисунок 2.3 б).

Механізм відкладеного підтвердження є дуже проблематичним, оскільки відправник має надіслати принаймні 2*SMSS байти на RTT, і не дозволяє малий RTT у мережі. Мережа з малим RTT може вимагати від відправника підтримувати вікно перевантаженості менше двох цілих сегментів на RTT. Відправник не може бути менш агресивним, оскільки одержувач у кінцевому підсумку затримає підтвердження в надії отримати більше байтів (рисунок 2.3 б), і відправник може ризикувати чекати дуже довго.

У той же час немає додаткової затримки для RTT, якби приймач не використовував розширені підтвердження і, таким чином, не затримував свої підтвердження (рисунок 2.4), але це призводить до великої кількості контрольних пакетів у мережі.

Тому бажано контролювати пакети, які споживають ресурси мережі, і розширювати відправника лише до одного сегмента на RTT у сценаріях, коли відправник продовжує відраховувати нові сегменти на основі годинника підтвердження.

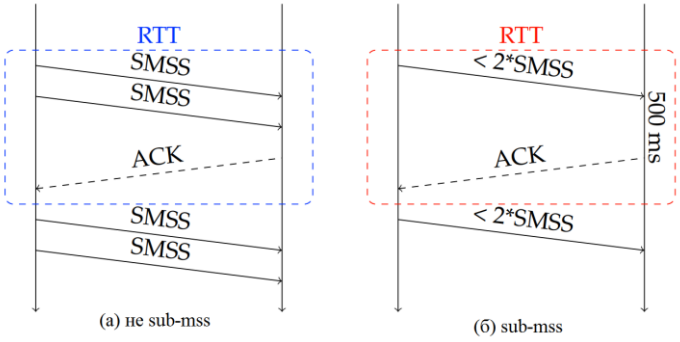

Рисунок 2.3 – Відкладене підтвердження: увімкнено

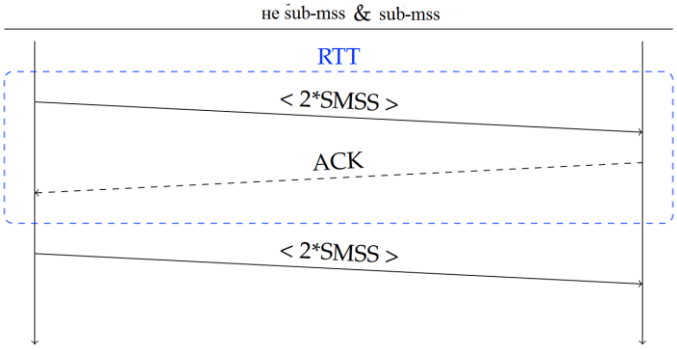

Рисунок 2.4 – Відкладене підтвердження: вимкнено

Тут припускається, що відправник ніколи не захоче надсилати сегменти, які містять менше байтів SMSS, якщо можлива передача повного сегмента. Передача меншого сегмента впливає на хорошу пропускну здатність з'днання, оскільки мережеві заголовки потребують більше витрат, ніж корисні дані на пакет.

Це означає, що відправник не повинен використовувати свій годинник підтвердження для відстеження нових пакетів у цьому середовищі, оскільки це призведе до жахливої продуктивності, оскільки RTT зменшиться в мережі. Прийнято називати таке середовище режимом sub-mss. Кажуть, що відправник перебуває в режимі sub-mss, якщо вікно перевантаження відправника опускається нижче коефіцієнта підтвердження розтягування, встановленого в одержувачі. Таким чином, TCP не працює належним чином

у режимі sub-mss і уникає режиму, зберігаючи вікно перевантаження на мінімальному рівні коефіцієнта підтвердження розтягування мережі. Порушення підтвердження відбувається, коли реалізація TCP намагається зберегти більш значний коефіцієнт розтягування підтвердження, ніж два. Як обговорювалося раніше, вищий коефіцієнт підтвердження розтягування є корисним для різних типів мережевих топологій.

Однак кажуть, що реалізація TCP порушила принцип розширеного підтвердження, якщо підтвердження не повертається після отримання двох сегментів. Проблема настільки поширена, що отримала свою назву і називається порушенням stretch ACK. Це загальновідома проблема, і вона була відкритою протягом дуже тривалого часу.

Незважаючи на те, що останній стандарт TCP для контролю перевантажень не забороняє реалізатору використовувати більш значний коефіцієнт ACK розтягування, стандарт чітко попереджає реалізатора про погіршення продуктивності.

Проблеми випливають із прямого факту потрапляння відправника в режим sub-mss. Відправник не може надіслати достатньо сегментів, щоб ініціювати підтвердження. Відправник не має іншого вибору, окрім як чекати, поки надійде відкладене підтвердження.

Відправник, чекаючи на надходження відкладеного підтвердження, припускає найгірше та ініціює відновлення втрати та отримує повторне виникнення тайм-аутів

Відправник повинен повторно перевірити мережу, і в результаті це призведе до низької продуктивності. Можна стверджувати, що в контрольованих середовищах, таких як центри обробки даних, легко налаштувати відправника, щоб він не потрапляв у режим sub-mss, але це лише переносить проблему на вищі бітрейти.

Проблема стає ще гіршою в не дуже контрольованих середовищах, оскільки для вибору відповідного коефіцієнта розтягування потрібне узгодження між кінцевими точками.

2.4 Алгоритм Негла

Алгоритм Нейгла Джона Нейгла став частиною стандарту TCP. Мета алгоритму Нейгла полягає в тому, щоб уникнути мережі з безліччю малих послідовних сегментів від відправника. Для кожного пакета додається додаткова інформація із заголовків стеку TCP/IP, які споживають пропускну здатність.

Тому краще надсилати цілі сегменти, ніж багато маленьких сегментів. Цей алгоритм має спільну мету, як використання розтягнутих підтверджень, тобто мати менше сегментів у транзиті на RTT, щоб заощадити обмежені ресурси мережі.

Алгоритм Нейгла надсилає сегмент завжди негайно, якщо немає попереднього надходження сегмента. Будь-які подальші невеликі сегменти не потрапляють у мережу, доки є деякі попередні сегменти в транзиті. Замість цього сегмент буферизується відправником і зберігається, доки весь сегмент не буде готовий до передачі. Алгоритм передає малий сегмент, що очікує на розгляд, щоразу, коли сегментів у транзиті немає. Це означає, що відправник може надіслати один невеликий сегмент для кожного RTT. Цей алгоритм намагається зберегти споживання смуги пропускання в мережі з припущенням, що програма або додаток продовжуватиме надсилати багато малих сегментів, напр. Telnet.

Алгоритм Нейгла має ненавмисний ефект, коли використовується разом із розтягнутими підтвердженнями. Дані програми можуть розділятися на два або більше сегментів, і алгоритм закінчує буферизацією останнього маленького сегмента. Використання як алгоритму Нейгла, так і розтягування підтвердження викликає м’яке взаємоблокування між двома хостами. Зрештою повертається відкладене підтвердження, яке у свою чергу дозволяє відправнику передати останній відсутній сегмент. Весь процес змушує програму чекати відповіді ще 40-500 мс, і це обмежує короткочасні потоки, які залежать від швидкої відповіді сервера (наприклад, веб-браузера).

Неглом було запропоновано зміну, що полягає у затримці останнього сегмента лише у тому випадку, якщо попередній сегмент також був невеликим. Тепер, коли дані програми розбиваються на сегменти, останній сегмент надсилається негайно, тоді як якщо програма надсилає декілька невеликих фрагментів даних у межах RTT, алгоритм має таку саму поведінку, як і раніше. Алгоритм Нейгла також можна вимкнути для кожного сокета за допомогою опції TCP_NODELAY у програмі.

2.3 Вибіркове підтвердження (SACK)

Вибіркове підтвердження (SACK) використовується на етапі відновлення стандартного TCP для ефективного відновлення після втрати. Відправник використовує додаткову інформацію, додану до підтвердження, щоб приймати розумні рішення, які сегменти повторно передавати на етапі відновлення.

Використання SACK дозволено, лише якщо обидва хости під час встановлення з’єднання дозволили його використання, використовуючи двобайтовий параметр у заголовку TCP. Приймач повідомляє про сегменти, які надійшли не в порядку.

Одержувач використовує параметр TCP у заголовку TCP, щоб повідомити про ці сегменти; кожен сегмент містить безперервний простір послідовності, приймач повідомляє про лівий і правий край кожного сегмента.

Лівий край – це порядковий номер першого байта в сегменті, а правий край – порядковий номер наступного байта після цього сегмента. Одержувач може повідомити до 4 таких записів (блоків) в одному пакеті; точна кількість залежить від використання інших параметрів TCP, оскільки простір для параметрів TCP обмежений. Відправник може з цієї інформації дізнатися, які сегменти були успішно отримані одержувачем з порушенням порядку, і знати, які сегменти повторно передати.

Duplicate-SACK (D-SACK) є розширенням SACK і може використовуватися для звітування про отримані дублікати сегментів (помилкова повторна передача). Кожен сегмент може містити один блок D-SACK. Коли блок D-SACK є підмножиною більшого безперервного простору послідовності, для звіту про цей масивний блок використовується наступний блок. Блоки повторного значення використовуються, як і раніше, для звітування про сегменти, які були отримані не в порядку.

2.5 Контроль перевантаження

Контроль перевантаження (Congestion Control, CC) дуже важливий для того, щоб відправники в мережі працювили разом, щоб ефективно використовувати обмежені ресурси. CC – це спосіб об’єднати набір споживачів для рівномірного розподілу обмежених ресурсів.

Найперший стандарт TCP містив опис керування потоком, але не мав жодного детального опису механізму контролю перевантаження. Одержувач повертав вікно потоку, знайдене в заголовку TCP, щоб контролювати швидкість передачі відправника.

Проте стандарт TCP визначив час очікування повторної передачі (Retransmission Timeout, RTO), який використовувався для повторної передачі втрачених сегментів. Сегмент буде скопійовано до черги повторної передачі як частину процесу передачі та вилучено, коли сегмент буде підтверджено.

Час очікування мав нижню та верхню межі та обчислювався на основі згладженого RTT із дисперсією затримки. Відправник повторно передає найперший непідтверджений сегмент після закінчення таймера для певного сегмента в черзі повторної передачі. Потім відправник повторно ініціює RTO. Реалізації TCP, засновані на цій простій логіці передачі, призводять до колапсу перевантаження в мережах. Колапс перевантаження відбувається, коли RTT раптово підвищується, напр. ініціюється велика передача, яка є

достатньо великою, щоб відправник повторно передав сегменти, оскільки таймер постійно закінчується (RTT>тайм-аут). Приблизний час очікування не оновлюється відправником достатньо швидко. Відправник продовжує повторно передавати сегменти та створює кілька копій тих самих сегментів у мережі. Ці пакети, швидше за все, будуть відкинуті або продовжуватимуть залишатися в черзі. Приймач отримує багато дубльованих сегментів і, зрештою, кілька нових сегментів. Пропускна здатність відправника залишається сильно зниженою, що призводить до значного зниження загальної пропускної здатності мережі. Кращий спосіб контролювати кожного відправника було запропоновано в роботі Ван Якобсена. Запропоноване рішення зберегло механізм керування потоком і надсилало сегменти за допомогою "годинника" підтвердження, щоб підтримувати фіксовану кількість сегментів у транзиті на RTT.

Годинник підтвердження було ініційовано за допомогою нового алгоритму повільного запуску. Розрахунок RTO було оновлено для використання експоненціального таймера відстрочки для сегментів у черзі повторної передачі, і відправник тепер повинен був зменшити швидкість передачі після виявлення втрати пакета. У документі також визначено алгоритм уникнення перевантаження для перевірки пропускної здатності в стабільному стані TCP. Ці алгоритми стають частиною наступного стандарту TCP, і всі реалізації TCP тепер повинні реалізовувати ці два алгоритми. Хоча найновіший стандарт керування перевантаженням TCP надає деталі щодо двох додаткових алгоритмів, які реалізація TCP повинна використовувати для відновлення після події втрати: швидка повторна передача та швидке відновлення.

Відправник використовує змінну стану порогового значення повільного запуску (ssthresh), щоб визначити, використовувати повільний запуск або алгоритм уникнення перевантаження на фазі адитивного збільшення TCP. Відправник використовує алгоритм повільного запуску, якщо відправник має менше сегментів у транзиті, ніж значення ssthresh. В іншому випадку

відправник використовує алгоритм уникнення перевантаження. Алгоритм уникнення перевантаження, як випливає з назви, є повільнішим і шукає додаткову ємність у більш консервативному підході. Значення ssthresh, таким чином, вирішує, як швидко відправник досліджує ємність. Механізм відновлення втрат TCP істотно уповільнює роботу відправника, зменшуючи значення ssthresh.

Відправник спочатку встановлює змінну стану ssthresh на максимально можливе значення (всі біти дорівнюють 1). Алгоритм повільного запуску зазвичай запускається, коли відкривається нове з'єднання для дослідження мережі з невідомим рівнем перевантаження. У деяких сценаріях значення ssthresh може бути встановлено нижче максимального значення, але це оптимізація, виконана реалізацією TCP.

Реалізація TCP повторно використовує різні значення, збережені з попереднього потоку. Мета полягає в тому, щоб запустити цей потік так, ніби це був попередній потік. Успіх цього залежить від поточного стану мережі. Стан мережі може бути не таким, як раніше. Тому цей пошук часто встановлюється як кеш, де запис є дійсним, якщо він досить свіжий. Хоча така оптимізація є хорошою та може забезпечувати кращу продуктивність, чи зазвичай вона вимикається під час порівняльного аналізу TCP.

2.6 Перевантажувальне вікно

Перевантажувальне вікно (Congestion Window, WB) визначає кількість байтів, яку відправник може видавати в мережу. Сукупне підтвердження зазвичай надходить після одного RTT і повідомляє відправнику, що одержувач отримав певну кількість байтів. Таке підтвердження також повідомляє відправнику, що ці байти більше не перебувають у мережі, і дозволяє відправнику продовжувати передачу нових байтів. Відправник використовує кумулятивне підтвердження, щоб підтримувати фіксовану кількість байтів на RTT ("годинник" підтвердження).

Перевантажувальне вікно можна візуалізувати як ковзне вікно (рисунок 2.5). Відправник для кожного RTT буде видавати W_B нових байтів (рисунок 2.5а), швидкість відправника (рівняння (2.1)). Швидкість передачі, таким чином, можна контролювати, регулюючи W_B.

$$x = WB/RTT \tag{2.1}$$

$$r = Ws/RTT \tag{2.2}$$

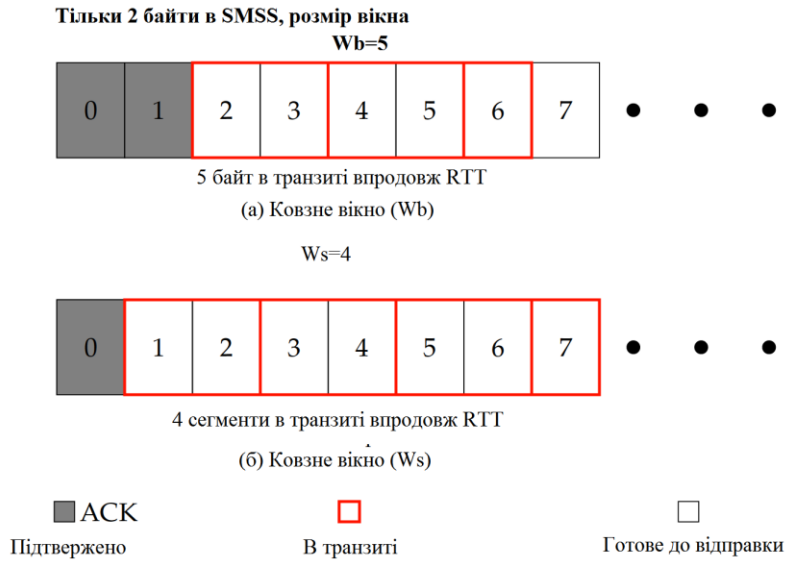

Рисунок 2.5 – Ковзне вікно

Крім того, відправник може використовувати концепцію цілих сегментів, де кожен сегмент містить до SMSS байтів (рисунок 2.5б). Тепер відправник надсилає Ws сегментів кожного RTT, швидкість пакетів відправника можна розрахувати за рівнянням (2.2). Швидкість передачі тепер можна контролювати, регулюючи Ws.

Ідея підходу на основі сегментів полягає в тому, що передавання сегментів, менших за SMSS байт, ніколи не є гарною ідеєю. Відправник завжди хоче надіслати цілий сегмент, якщо це можливо, щоб мінімізувати накладні витрати на мережеві заголовки. Реалізація TCP, яка реалізує

сегментний підхід, може легко забезпечити вікно перевантаження щонайменше одним цілим сегментом, щоб пакети були заповнені цілими сегментами. Можуть бути ситуації, коли відправник змушений передавати менші пакети, але відправник може вирішити це за допомогою алгоритму Нейгла.

Навпаки, реалізація TCP, яка використовує байтовий підхід, має підтримувати перевантажувальне вікно на рівні мінімум SMSS байтів і передавати лише цілі сегменти, щоб досягти тих же переваг. Таким чином, немає очевидних переваг використання підходу на основі байтів, а використання підходу на основі сегментів дозволяє більш просту реалізацію.

Перевантажувальне вікно зазвичай встановлюється на значення не менше двох сегментів, оскільки одержувач може використовувати відкладені підтвердження. Відправник намагається надіслати щонайменше два сегменти на RTT, щоб отримати негайне підтвердження від одержувача. Відправник використовує лише перевантажувальне вікно одного сегмента, коли RTO закінчується (вікно втрати). Стандарт TCP визначає початкове перевантажувальне вікно для нового з’єднання між двома та чотирма цілими сегментами. Модифікацією для підвищення загальної продуктивності був експериментальний стандарт, запропонований для початкового перевантажувального вікна з десяти цілих сегментів. Короткочасні з’єднання TCP виграють від цього, оскільки вони зазвичай не живуть достатньо довго, щоб підтримувати велике перевантажувальне вікно.

Відправник оновлює своє перевантажувальне вікно на основі зворотного зв’язку від одержувача. Сукупне підтвердження дозволяє відправнику збільшити або зменшити перевантажувальне вікно. Зростання у перевантажувальному вікні зазвичай відноситься до підтвердження в стабільному стані TCP. Тим часом, підтвердження на етапі відновлення втрат TCP відноситься до негативного підтвердження. Підтвердження має негативне значення, оскільки воно призводить до скорочення перевантажувального вікна, а не до зростання.

Схеми адитивного збільшення мультиплікативного зменшення (AIMD) широко використовуються відправниками для оновлення свого перевантажувального вікна. Відправник оновлює своє перевантажувальне вікно на основі поточного використовуваного алгоритму керування перевантаженням. Звичайною реакцією на відсутність перевантаження є збільшення перевантажувального вікна на один цілий сегмент на RTT. З іншого боку, звичайною реакцією на перевантаження є зменшення перевантажувального вікна до половини початкового розміру. Таким чином, відправник із великим перевантажувальним вікном має більший штраф, ніж відправник із меншим перевантажувальним вікном. Усі відправники, які використовують схеми AIMD, у кінцевому підсумку об’єднуються, щоб мати рівну частку доступної смуги пропускання, коли перевантажувальне вікно стає рівним.

Атака ACK Division – це атака, яка використовується проти реалізацій TCP, які збільшують своє перевантажувальне вікно рівно на один цілий сегмент за кожне отримане підтвердження.

Одержувач зловживає використанням часткових підтверджень і спонукає відправника використовувати набагато більше перевантажувальне вікно. Відправник видає більше байтів, ніж те, що залишає мережу. Зловмисний отримувач порушує годинник підтвердження відправника. Захист від такої атаки полягає у збільшенні перевантажувального вікна за допомогою відповідного підрахунку байтів (ABC), яке має збільшуватися лише на кількість байтів, підтверджених одержувачем, але не більше ніж на один цілий сегмент.

2.7 Перевантажувальне вікно та вікно потоку

Перевантажувальне вікно не слід плутати з вікном потоку в заголовку TCP (16 біт). Вікно потоку є обмеженням пам’яті одержувачів і оновлюється з кожним підтвердженням. Вікно потоку інформує відправника про обсяг

пам’яті, який має одержувач на даний момент. Відправник не може надіслати одержувачу більше байтів, ніж це обмеження, байти, які наразі перебувають у транзиті, також зараховуються до цього обмеження. Замість цього відправнику потрібно дочекатися підтвердження від одержувача з оновленим вікном потоку і лише потім передати більше байтів, якщо це дозволяє нове вікно потоку.

Перевантажувальне вікно не дозволяє відправнику надсилати за межі вікна потоку, оголошеного одержувачем. Відправник керує своєю передачею мінімальним вікном потоку, перевантажувальним вікном та розміром буфера відправки відправника.

2.8 Дружнє керування швидкістю TCP (TFRC)

TCP Friendly Rate Control (TFRC) може використовуватися замість віконного керування для контролю перевантаження. Одноадресний потік TFRC діє справедливо проти іншого потоку TCP, використовуючи схеми AIMD.

TFRC використовує рівняння пропускної здатності на основі сигналів перевантаження та вимірювань RTT. Для цих вимірювань відправник використовує зворотний зв’язок від одержувача. На відміну від керування вікном, відправник TFRC має більш стабільну пропускну здатність (невеликі коливання), але він повільніше адаптується до змін у мережі. TFRC добре підходить для програм, які передають потокове відео.

Однак, TFRC не є гарним вибором, якщо програма хоче досягти максимально високої пропускної здатності. Відправник TFRC зазвичай використовує фіксований розмір сегмента та змінює швидкість надсилання пакетів за секунду, щоб контролювати швидкість передачі. Відправник має використовувати TFRC, лише якщо необхідна безперебійна пропускна здатність, оскільки загальний алгоритм контролю перевантажень із використанням AIMD зменшує швидкість вдвічі.

2.9 Повільний старт

Відправник починає повільне дослідження мережі за допомогою алгоритму повільного запуску та ініціює «годинник» підтвердження. Алгоритм повільного запуску збільшує вікно перевантаження на максимальну кількість SMSS байтів або на один повний сегмент під час кожного вхідного підтвердження. Хоча рекомендовано лише збільшити кількість байтів, нещодавно підтверджених одержувачем, N байтів, для захисту від атаки з розділенням ACK:.

$$W \mathrel{+}= \min(N, SMSS) \tag{2.3}$$

Таким чином, алгоритм повільного запуску має експоненціальне зростання перевантажувального вікна для кожного RTT, подвоюючи перевантажувальне вікно кожного RTT (рисунок 2.6). Алгоритм повільного запуску розвивається дуже швидко з метою пошуку доступної смуги пропускання якомога швидше, і він схильний до «перевищення» доступної смуги пропускання. Така поведінка може призвести до черги та вибухової поведінки у вузькому місці.

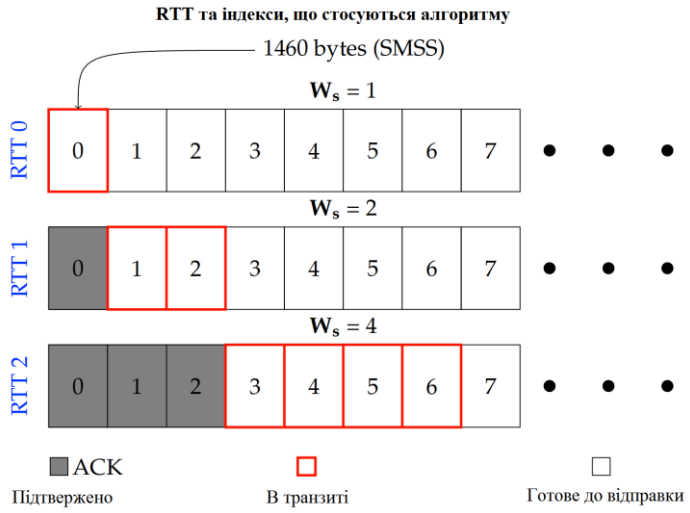

Рисунок 2.6 – Ковзне вікно повільного старту (Ws)

2.10 Уникнення перевантаження

Відправник перемикається на алгоритм уникнення перевантаження, коли перевантажувальне вікно перевищує значення ssthresh. Алгоритм уникнення перевантажень є більш консервативним, ніж алгоритм повільного старту.

Алгоритм збільшує перевантажувальне вікно на один повний сегмент для кожного RTT (рисунок 2.7). Рівняння (2.3) з повільного старту все ще можна використовувати для оновлення перевантажувального вікна. Єдина відмінність полягає в тому, що рівняння застосовується лише один раз у RTT, тоді як в алгоритмі повільного запуску воно застосовується для кожного підтвердження, отриманого в RTT.

Це означає, що алгоритм уникнення перевантаження має лінійне зростання перевантажувального вікна для кожного RTT, тоді як алгоритм повільного запуску мав експоненціальне зростання перевантажувального вікна для кожного RTT.

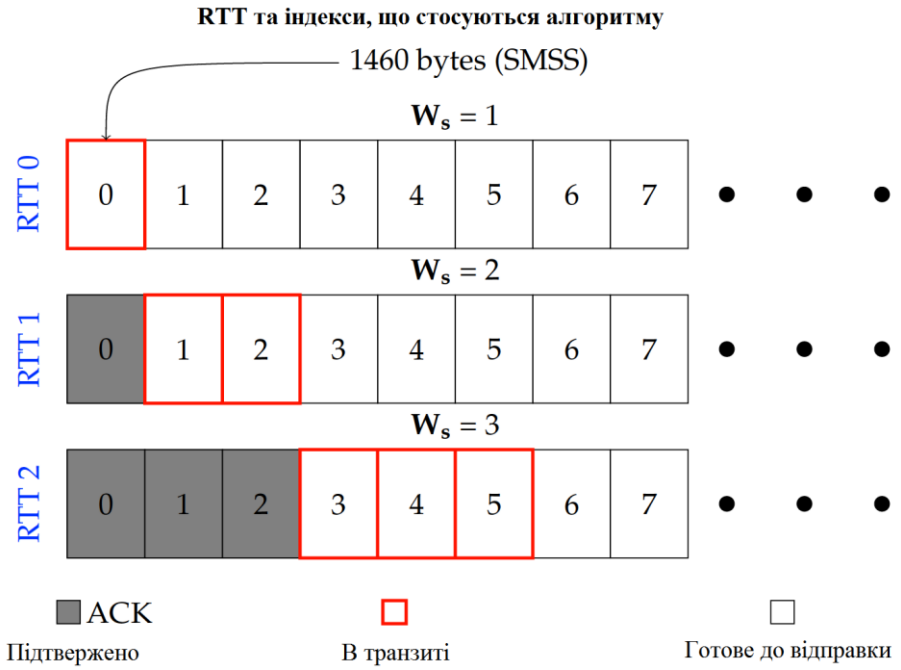

Рисунок 2.7 – Ковзне вікно для уникнення заторів (Ws)

Відправник все ще може оновлювати перевантажувальне вікно для кожного отриманого підтвердження, оскільки для кожного RTT є байти WB у транзиті:

$$WB \mathrel{+}= SMSS * SMSS / WB \tag{2.4}$$

Однак останній стандарт контролю заторів рекомендує оновлювати перевантажувальне вікно під час уникнення заторів за допомогою додаткової змінної стану. Кількість нещодавно підтверджених байтів збільшує змінну стану, а перевантажувальне вікно спочатку збільшується на один повний сегмент, оскільки змінна стану є такою ж важливою, як і перевантажувальне вікно. Потім змінна стану повертається до 0 для наступного RTT. Цей метод гарантує, що вікно перевантаження має лінійне зростання для кожного RTT у ситуаціях, коли отримувач використовує відкладені підтвердження.

Подібним чином, якщо відправник використовує концепцію цілих сегментів, перевантажувальне вікно все одно оновлюватися після кожного підтвердження, оскільки є Ws сегментів у транзиті. Відправник оновлює перевантажувальне вікно на 1/Ws для кожного отриманого підтвердження. Відправник накопичує змінну стану кожного разу, коли одержувач підтверджує отримання цілого сегмента. Відправник збільшує перевантажувальне вікно на один сегмент, як тільки змінна стану підтвердить одне вікно сегментів у транзиті.

Алгоритм запобігання перевантаженню зберігає вже отриману пропускну здатність і намагається вимагати додаткової пропускної здатності, доки не буде отримано сигнал перевантаження. Відправник зменшує швидкість передачі на етапі відновлення втрати.

Фаза уникнення перевантажень повторюється після завершення фази відновлення втрат. Природа алгоритму уникнення перевантаження робить алгоритм досить неефективним для мереж із низькою пропускною спроможністю (низьким BDP (Bandwidth Delay Product)). Перевантажувальне

вікно збільшується лише на один повний сегмент у RTT незалежно від того, наскільки великим є шлях BDP. У таких ситуаціях процес збільшення перевантажувального вікна від зменшеного вікна назад до обмеження мережі є повільним і вимагає великої кількості RTT. Алгоритм уникнення перевантаження також не підходить для шляху з низьким рівнем BDP, оскільки додавання одного цілого сегмента може бути більшим, ніж пропонує шлях з низьким рівнем BDP. Іншими словами, алгоритм уникнення перевантажень погано масштабується ні для низьких, ні для великих шляхів BDP у мережі.

2.11 Фаза відновлення

Фаза відновлення ініціюється відправником для відновлення втрачених сегментів. Втрачений сегмент – це сегмент, який мережа вважає за втрачений. TCP використовує різні підходи для виявлення втрати сегмента з різною ефективністю.

Втрата завжди врешті-решт усувається тайм-аутом повторної передачі (RTO), але це досить неефективно. Відправник використовує RTO як запасний механізм для повторного тестування мережі, коли всі інші інтелектуальні механізми вийшли з ладу.

Реалізація TCP зазвичай реалізує алгоритми швидкої повторної передачі та швидкого відновлення для покращеної фази відновлення. NewReno – це оптимізація алгоритму швидкого відновлення, яка забезпечує ефективну фазу відновлення навіть із частковими підтвердженнями. Відправник, який використовує NewReno, повинен оновити змінну стану ssthresh до половини потоку даних, щойно виявить втрату сегмента. Це оновлення, по суті, є фазою мультиплікативного зменшення в схемі AIMD і потрібне, щоб змусити відправників об’єднуватися у вузькому місці. Відправник зменшує ssthresh лише один раз для певного сегмента у вікні. Однак відправник повинен ще більше зменшити своє значення ssthresh, якщо

подія втрати сталася після повторної передачі. Змінна стану ssthresh також має бути принаймні двома повними сегментами для взаємодії з відкладеними підтвердженнями. Відправник використовує нове значення ssthresh, лише якщо воно вище нижньої межі:

$$ssthresh=max(inflight/2,2*SMSS) \quad (2.5)$$

Відправник повинен обмежити передачу нових сегментів на етапі відновлення втрат, доки залишиться менше незавершених сегментів, ніж нове значення ssthresh. Відправник має дочекатися завершення фази відновлення і лише тоді шукати додаткову ємність за допомогою алгоритму уникнення перевантажень. Механізм відновлення також спрацьовує щоразу, коли мітка ECN повертається від одержувача. Звичайний TCP розглядає позначку ECN як такий самий рівень перевантаження, що й подія втрати. Це означає, що відправник реагує такою ж відповіддю та зменшує швидкість передачі вдвічі.

Вибіркове підтвердження (SACK) дозволяє відправнику дуже ефективно відновлюватися після втрати за допомогою додаткової інформації, яку одержувач додає до підтвердження. Duplicate-SACK (D-SACK) можна використовувати з SACK для виявлення помилкових повторних передач. Використання інформації вибіркового підтвердження (SACK) є рекомендованим способом відновлення після втрати, і його слід використовувати, коли це можливо.

2.12 Час очікування повторної передачі (RTO)

Тайм-аут повторної передачі (RTO) використовується як останній засіб, якщо підтвердження не повертається від отримувача протягом заданого періоду. Використання тайм-ауту повторної передачі (RTO) було оновлено та стало обов’язковим. Відправник має перезапустити зондування мережі,

враховуючи, що стан мережі змінився, тобто оновити ssthresh і встановити перевантажувальне вікно на 1 (вікно втрати). Відправник повторно запускає алгоритм повільного запуску та перемикається на алгоритм уникнення перевантаження, коли перевантажувальне вікно досягає нового значення ssthresh.

Тайм-аут повторної передачі (RTO) визначає тривалість, яку відправник має чекати перед повторною передачею першого непідтвердженого сегмента. Час очікування встановлюється за допомогою згладженого часу проходження туди й назад (SRTT), який обчислюється за допомогою вимірювань RTT. Відправник повинен оновлювати тайм-аут кожного RTT (виконує принаймні одне вимірювання RTT на RTT). Повторно передані сегменти можна використовувати для вимірювання RTT, лише якщо вони містили параметр мітки часу, інакше відправник не зможе відрізнити останній сегмент від раніше надісланих сегментів (підтвердження може бути з сегмента, який вважався втраченим).

Час очікування спочатку встановлюється на одну секунду, якщо відправник не має вимірювання RTT. Відправнику не дозволяється ініціювати RTO тривалістю менше однієї секунди. Відправник повинен використати тайм-аут у 3 секунди, якщо перший сегмент було втрачено (SYN). Тайм-аут зростає експоненціально кожного разу, коли відправник повторно передає за допомогою RTO.

Тайм-аут запускається знову після отримання нового вимірювання RTT (успішна доставка). RTO працює як алгоритм "відступу", штраф зростає за кожен раз, коли відправник не отримує підтвердження від одержувача протягом тайм-ауту.

RTO допомагає уникнути збою перевантаження, оскільки час очікування зростає експоненціально. Відправник змушений чекати довше. Зрештою відправник отримує підтвердження, коли час очікування стає більш значним, ніж RTT. Нове вимірювання RTT використовується для повторного запуску RTO.

2.13 Швидка повторна передача

Індикатор високої втрати – це коли сегмент надходить, утворюючи дірку в просторі порядкового номера. Одержувач отримує сегмент не в порядку та ще не може переслати сегмент до програми, оскільки частина попередніх даних у потоці відсутня. Таким чином, надходження сегментів, що не відповідають порядку, є важливою інформацією для відправника. Відправнику потрібна ця інформація для ефективного відновлення після втрати.

Одержувач використовує дублікати підтвердження, щоб сповістити відправника про подію, що не відповідає порядку. Подвійне підтвердження підтверджує найперший відсутній сегмент, а не вхідний сегмент, оскільки одержувач не має права підтверджувати за межами наступного очікуваного байта в потоці. У цьому випадку одержувач згенерував дублікат підтвердження замість звичайного підтвердження.

Одержувач не повинен затримувати таке підтвердження, щоб відправник міг розпочати етап відновлення втрати якомога швидше. Також важливими є підтвердження того, що частини дірки усунуті, оскільки відправнику потрібен швидкий зворотний зв'язок на етапі відновлення. Таким чином, приймач повинен негайно підтвердити будь-який сегмент за межами дірки.

Хоча нерідкі випадки, коли сегмент прибуває раніше, ніж наступний очікуваний сегмент з кількох причин (перевпорядкування, втрата, дублювання).

Відправник не ініціює відновлення втрати одразу після першого дублікату підтвердження, сподіваючись на помилковий результат. Відправник чекає, доки не буде отримано принаймні три послідовні повторювані підтвердження для того самого сегмента, перш ніж ініціювати алгоритм швидкої повторної передачі. Цей алгоритм є першою фазою механізму відновлення втрат із повторюваними підтвердженнями.

Відправник не повинен оновлювати своє перевантажувальне вікно до того, як надійде третій дублікат підтвердження. Хоча відправник може надіслати додатковий сегмент, коли надходить дублікат підтвердження, оскільки дублікат підтвердження зазвичай означає, що в мережі на один сегмент менше.

Нарешті, коли надійшли три послідовні дублікати підтвердження відправник може з високою ймовірністю вважати сегмент втраченим. Відправник зменшує швидкість передачі, зменшуючи змінну стану ssthresh.

Відправник тепер повторно передає відсутній сегмент і оновлює своє перевантажувальне вікно до нового значення ssthresh+3. Додавання константи 3 стосується всіх сегментів, які залишили мережу перед входом у швидку повторну передачу. Потім відправник переходить до наступного етапу фази відновлення, відомого як алгоритм швидкого відновлення.

2.14 Швидке відновлення

Відправник ініціює алгоритм Fast Recovery після того, як Fast Retransmit повторно передасть втрачений сегмент. Відправник тимчасово збільшує перевантажувальне вікно під час фази швидкого відновлення на один цілий сегмент для будь-якого додаткового отриманого дубліката підтвердження.

Відправник може надіслати цілий сегмент нових даних, якщо це дозволено новим перевантажувальним вікном та вікном потоку.

Наступне підтвердження, яке дає нові дані, має підтверджувати до останнього сегмента, надісланого у відповідь на повторне підтвердження у швидкій повторній передачі. Відправник повинен був усунути діру в послідовності сегментів, тому повинен встановити перевантажувальне вікно назад на ssthresh перед виходом із фази швидкого відновлення. Уникнення перевантажень потім відновлюється для повторного дослідження доступної ємності.

2.15 NewReno

Алгоритм Fast Retransmit і Fast Recovery часто називають Reno. NewReno – це оптимізація алгоритму Fast Recovery у Reno.

Відправник припускає, що алгоритм швидкого відновлення усунув дірку в даних, коли надходить підтвердження отримання нових даних. Відправник очікує, що підтвердження буде підтверджено до останнього сегмента, надісланого перед початком відновлення втрати. Однак це не завжди так, оскільки відправник міг втратити кілька сегментів або мережа могла змінити порядок сегментів. У такій ситуації відправник отримує лише часткове підтвердження.

Це часткове підтвердження підтверджує деякі, але не всі дані, надіслані перед входом у швидку повторну передачу. Відправник виходить із швидкого відновлення після отримання часткового підтвердження, вважаючи, що діру виправлено, але натомість ризикує чекати, доки закінчиться RTO.

Відправник може відновити втрату кількох сегментів, перейшовши знову до швидкої повторної передачі (отримано три додаткові дублікати підтвердження).

Однак відправник не працює належним чином, оскільки повторне проходження Fast Retransmit змушує відправника вдруге зменшити швидкість передачі. Таким чином, алгоритм Fast Recovery забезпечує жахливу продуктивність в умовах, коли отримані часткові підтвердження.

NewReno не оголошує фазу відновлення завершеною, доки отримувач не отримає всі відсутні дані до початку фази відновлення втрати. NewReno використовує додаткову змінну стану «recover» для відстеження початку фази відновлення.

Оскільки відправник отримує часткове підтвердження, яке не підтверджує аж до змінної стану "recover", передається перший непідтверджений сегмент. Потім відправник оновлює своє

перевантажувальне вікно, щоб відобразити додаткову кількість підтверджених байтів, і надсилає інший сегмент, що містить нові дані, якщо це дозволено новим перевантажувальним вікнм. Цей підхід гарантує, що відправник має ssthresh кількість байтів у транзиті після завершення фази відновлення.

Нарешті, відправник зменшує перевантажувальне вікно, як і раніше, і повторно перевіряє мережу за допомогою алгоритму уникнення перевантаження.

2.16 Відновлення за допомогою інформації SACK

Одержувач сповіщає відправника про сегменти, які були отримані не по порядку, використовуючи повторювані підтвердження з інформацією SACK. Ця інформація може бути використана відправником, щоб дізнатися про кілька втрачених сегментів в одному RTT, де на стандартній фазі відновлення TCP відправник дізнається лише про один втрачений сегмент для кожного RTT. Однак відправник не повинен бути більш агресивним, тобто повторно передавати більше сегментів, ніж це дозволено на стандартній фазі відновлення TCP.

Одержувач повинен вказати останній сегмент, який ініціював повторне підтвердження в першому блоці інформації SACK. Відправник отримує своєчасну інформацію від одержувача на етапі відновлення. Таким чином, відправник може знати, який сегмент викликав повторне підтвердження, і що сегмент має бути в межах сукупного номера підтвердження та інформації SACK першого блоку.

Приймач заповнює решту блоків SACK інформацією про останні отримані ізольовані сегменти. Це означає, що відправник отримає перший блок, повторений три рази в трьох послідовних повторних підтвердженнях. Тому інформація SACK для кожного блоку повторюється до трьох разів. Таким чином, SACK є надійним, коли мережа втрачає деякі підтвердження,

що містять інформацію SACK. Однак з D-SACK справа йде трохи інакше, оскільки він також приймає перший блок підтвердження. Таким чином, приймач не повторює інформацію про помилкову повторну передачу.

Відправник все ще повинен залежати від RTO як резервного механізму. У момент закінчення терміну дії RTO відправник повинен ігнорувати будь-яку попередню надану інформацію SACK. Потім відправник повинен повторно перевірити мережу та вважати будь-які сегменти, що перевищують кумулятивне підтвердження, втраченими.

2.17 Активне керування чергою (AQM)

Активне керування чергою (AQM) намагається усунути високу затримку в черзі, яка спостерігається в мережах передачі даних через величезну чергу. AQM вилучає пакет із черги, оскільки черга починає виходити з-під контролю. AQM може позначати пакети ECN, якщо обидва хости підтримують ECN і якщо хост дозволив його використання в поточному пакеті з відповідною кодовою точкою. Існують схеми AQM з різними підходами та складністю, і це сфера, яка активно досліджується. AQM дозволяє часткове розгортання на елементах мережі, розташованих в Інтернеті.

AQM зазвичай видає сигнал перевантаження, коли пакет стоїть у черзі протягом більш тривалого періоду. Відправник дійсно знижує швидкість передачі, як тільки виявляє сигнал перевантаження. Це означає, що AQM може запропонувати, як швидко повинен рухатися кожен відправник, щоб досягти бажаної довжини черги. Основною метою використання AQM є швидкість введення, яка відповідає швидкості виведення з мінімальною чергою. Ідея полягає в тому, щоб решта черги була доступною для пакетів. Іншою метою AQM є зниження ймовірності глобальної синхронізації між потоками. AQM не хоче сигналізувати всім відправникам одночасно, щоб запобігти синхронізованій пилкоподібній поведінці TCP у черзі.

Перевантажувальні вікна відправників збігаються і зазвичай стають однаковими значеннями. Небажаний сценарій, коли всі відправники йдуть однаково швидко, оскільки тоді це стає періодом з невеликою чергою та в інші періоди, коли черги немає взагалі. AQM зазвичай використовує випадкову ймовірність, щоб сигналізувати про пакет після того, як черга досягла певного порогу, щоб десинхронізувати потоки.

Розгортання AQM, здається, виправляє деяку затримку в черзі, спричинену поведінкою TCP, яка шукає пропускну здатність, але це викликає іншу проблему: більшість реалізацій TCP не надсилають менше двох повнорозмірних сегментів на RTT. Відправник намагається надсилати принаймні два повнорозмірних сегменти в RTT, щоб уникнути механізму затримки підтвердження, який є в одержувача. Відправник навмисно ігнорує будь-які рекомендації AQM щодо утримання вікна перевантаження вище рівня двох сегментів. Це викликає проблему, оскільки AQM намагається форсувати невелике RTT.

3 КОНТРОЛЬ ПЕРЕВАНТАЖЕНЬ ТА AQM

Перевантаження стало важливою проблемою в мережах з комутацією пакетів. Коли трафік або навантаження в мережі зростає за межі пропускної здатності мережі, це призводить до перевантаження комп’ютерної мережі. Перевантаження сильно впливає на пропускну здатність. Коли в комп'ютерній мережі виникає перевантаження, пропускна здатність значно зменшиться, а затримка збільшиться.

AQM – це механізм на основі маршрутизатора для раннього виявлення перевантажень у комп’ютерній мережі. Основна ідея AQM полягає в тому, щоб завчасно відчути й виявити перевантаження, а також повідомити відправника про необхідність зменшити швидкість надсилання, таким чином зменшуючи кількість пакетів, що надсилаються в мережі, і контролювати перевантаження. Існує кілька алгоритмів AQM, які контролюють перевантаження.

3.1 Контроль перевантаження

Контроль перевантаження – це механізм, який запобігає перевантаженню до того, як воно виникне, або усуває перевантаження після того, як воно сталося в мережі.

Контроль перевантажень можна розділити на дві широкі категорії. Контроль перевантажень у відкритому контурі (запобігання). Контроль перевантажень у відкритому контурі використовується для запобігання перевантаженню до того, як воно виникне в мережі.

Контроль перевантажень замкнутого контуру. У цьому механізмі політики застосовуються для усунення перевантажень мережі після їх виникнення.

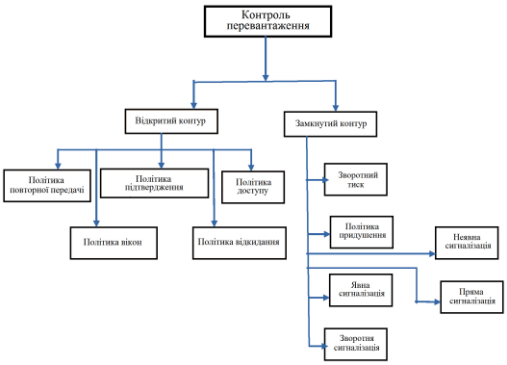

Рисунок 3.1 – Методи контролю перевантажень

Політика повторної передачі – коли маршрутизатор відкидає пакет, його має повторно передати відправник. Повторна передача збільшує перевантаження в мережі. Але хороша політика повторної передачі може запобігти виникненню перевантаження в мережі. Для ефективної політики повторної передачі таймери розроблені таким чином, щоб оптимізувати ефективність і запобігати перевантаженням у мережі.

Політика вікон – вибір відповідного типу вікна може допомогти зменшити затори. Наприклад, під час перевантаження, якщо ви вибрати вікно GO-Back-N, кількість пакетів збільшиться, оскільки вікно GO-Back-N знову надсилає всі пакети з порядкового номера втраченого пакета. Подібним чином вікно вибіркового повторення повторно передаватиме лише втрачені або скинуті пакети, таким чином зменшуючи перевантаження. Отже, вікно вибіркового повтору краще порівняно з вікном Go-Back-N.

Політика підтвердження – політика підтвердження покладається на стороні одержувача. Використовується кілька політик:

- одержувач не підтверджує кожен пакет, який він отримує, тому сповільнює відправника, таким чином зменшуючи перевантаження;
- одержувач надсилає підтвердження, лише якщо він має спеціальний пакет, який потрібно надіслати, або минув таймер повторної передачі;
- одержувач підтверджує N пакетів за раз, і не кожен пакет, таким чином зменшуючи трафік у мережі.

Політика відкидання – хороша політика відкидання може допомогти запобігти перевантаженню мережі, відкидаючи менш чутливі пакети, не впливаючи на цілісність передачі. Наприклад, під час передачі аудіо- та відеофайлів політика відхилення може відкидати менш чутливі пакети, не впливаючи на якість звуку чи зображення, таким чином запобігаючи або зменшуючи перевантаження комп'ютерної мережі.

Політика доступу – протягом періоду перевантаження політика доступу встановлює віртуальне з'єднання лише для високопріоритетних потоків залежно від доступності ресурсів, інакше маршрутизатор просто відмовляє у встановленні з'єднання віртуального каналу.

Зворотний тиск – під час зворотного тиску перевантажений вузол перестає отримувати дані від найближчого вищестоящого вузла, а безпосередньо вищестоящий вузол, у свою чергу, відхиляє дані від свого наступного вищестоящого безпосереднього вузла. Таким чином у зворотному тиску перевантажений вузол відхиляє всі пакети від свого безпосереднього вузла, таким чином контроль над перевантаженням поширюється в протилежному напрямку до джерела.

Політика придушення – у політиці choke-пакетів окремий пакет надсилається від перевантаженого вузла безпосередньо до джерела, інформуючи джерело про перевантаження, щоб джерело, у свою чергу, могло зменшити швидкість надсилання, що, як наслідок, зменшує перевантаження в мережі.

Неявна сигналізація – у неявній сигналізації перевантажений вузол або приймач не надсилає жодних сповіщень про перевантаження безпосередньо відправнику. Відправник сам здогадується про виникнення затору, спостерігаючи наступні симптоми:

- немає підтвердження для надісланих пакетів;
- велика затримка отримання підтвердження від одержувача.

У таких випадках відправник припускає, що мережа перевантажена, і знижує швидкість надсилання.

Явна сигналізація – у явній сигналізації вузол, який є перевантаженим, явно сигналізує джерелу або одержувачу про сповіщення про перевантаження.

Явна сигналізація відрізняється від choke-пакетів, оскільки в choke-пакеті відправнику надсилається окремий пакет для сповіщення про перевантаження, але у випадку явної сигналізації сигнал сповіщення про перевантаження включається в сам пакет, який переносить дані.

Зворотна сигналізація – у зворотній сигналізації біт встановлюється для сповіщення про перевантаження, яке рухається в протилежному напрямку перевантаження до джерела. Цей біт інформує джерело про перевантаження в мережі, щоб джерело могло зменшити швидкість надсилання.

Пряма сигналізація – у прямій сигналізації біт встановлюється для сповіщення про перевантаження в пакеті, що рухається в напрямку перевантаження до одержувача. Цей біт інформує приймача про перевантаження в мережі, щоб приймач міг затримати підтвердження джерела.

3.2 Активне керування чергою

Активне керування чергою (AQM) має на меті виявити перевантаження в мережі до того, як воно стане серйозним через переповнення черги маршрутизатора. Це означає, що маршрутизатор намагається зменшити швидкість надсилання джерел трафіку, відкидаючи або позначаючи пакети.

Існує два підходи до вказівки перевантаження: пакети можуть бути відкинуті та пакети можуть бути позначені.

Перша стратегія вимагає взаємодії кінцевих точок, а друга генерує додаткові витрати через повторне надсилання. Кінцеві точки повинні реагувати на відмічені пакети, коли вони були відкинуті, і зменшувати пропускну здатність. Завдяки цьому можна досягти такого ж покращення використання пропускної здатності, але без додаткових накладних витрат.

Крім того, деякі механізми AQM спрямовані на зменшення пропускної здатності жадібних потоків шляхом відкидання їхніх пакетів на вищих швидкостях.

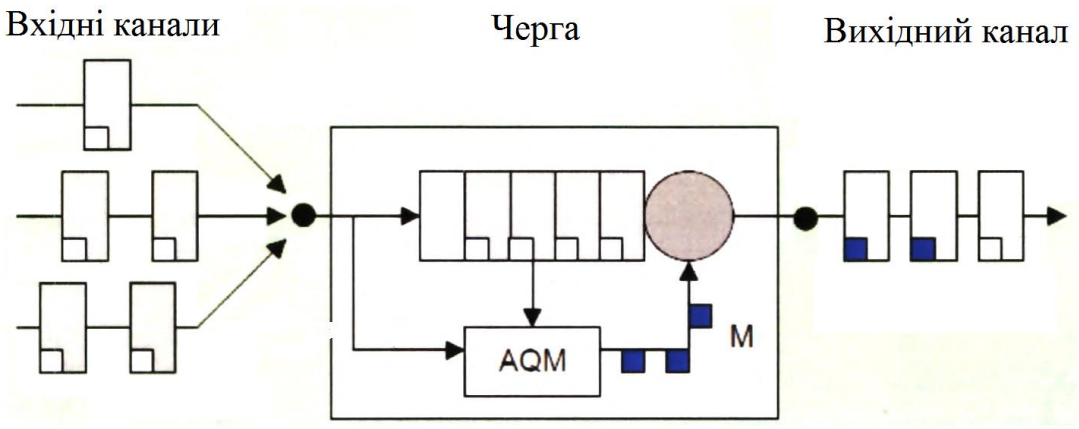

Рисунок 3.2 – Активне керування чергою

Активне керування чергою (AQM) відноситься до методів керування трафіком на маршрутизаторі, які виявляють і повідомляють джерела трафіку про неминуче перевантаження мережі, щоб запобігти вихідному буферу через потік і контролювати затримку в черзі. Повідомляючи про перевантаження мережі, кооперативні джерела трафіку, такі як TCP, знижують швидкість передачі, щоб взяти участь у контролі перевантаження. У випадку, якщо джерела трафіку не можуть добровільно врегулювати перевантаження мережі, AQM можуть використовувати методи керування буфером, щоб придушити трафік до цільового рівня трафіку та досягти мети QoS.

Загалом завдання AQM можна розділити на такі: монітор перевантаження, який виявляє та оцінює перевантаження, контролер пропускної здатності, який керує використанням вихідної смуги пропускання, контролер перевантаження, який обчислює та застосовує ймовірність сповіщення про перевантаження (CNP) до вхідного трафіку. і контролер черги, який керує використанням буфера та плануванням пакетів.

Першим завданням AQM є моніторинг, виявлення та оцінка перевантаження. Ця оцінка використовується контролером пропускної здатності для рішень щодо керування смугою пропускання або для обчислень ймовірності сповіщення про перевантаження (CNP) у контролері перевантаження.

AQM може мати контролер пропускної здатності, який керує використанням вихідної пропускної здатності. Контролери пропускної здатності можна класифікувати на основі характеру послуг, які вони надають, і цілей QoS. Контролер пропускної здатності може забезпечувати пріоритетну переадресацію або службу диференціації втрат.

Пріоритетне пересилання – це механізм захисту на основі класу пріоритетів, у якому під час перевантаження пакети з класу нижчого пріоритету відкидаються перед видаленням пакетів, що належать до класів з вищим пріоритетом

Служба диференціації втрат також є механізмом захисту на основі класів, у якому після перевантаження попередньо визначена частка трафіку відкидається з кожного класу.

Вхідний трафік, який проходить через контролер пропускної здатності, пересилається до контролера перевантаження. Робота контролера перевантаження полягає в тому, щоб запобігати або контролювати перевантаження мережі, сповіщаючи джерела трафіку про загрозливе перевантаження раніше, щоб джерела трафіку, що реагують на перевантаження, такі як TCP, могли зменшити швидкість передачі. Хоча явний двійковий метод сповіщення про перевантаження називається явним сповіщенням про перевантаження (ECN), історично використовувався неявний механізм відкидання вхідних пакетів.

З цієї причини іноді важко відрізнити контролери перевантаження від контролерів пропускної здатності, оскільки відкидання пакетів у результаті керування пропускною здатністю також діє як неявне сповіщення про перевантаження.

Останнім компонентом AQM є контролер черги. Контролер черги керує передачею пакетів, що пересилаються контролером перевантаження або контролером пропускної здатності. Як правило, механізми AQM зберігають лише одну чергу пакетів.

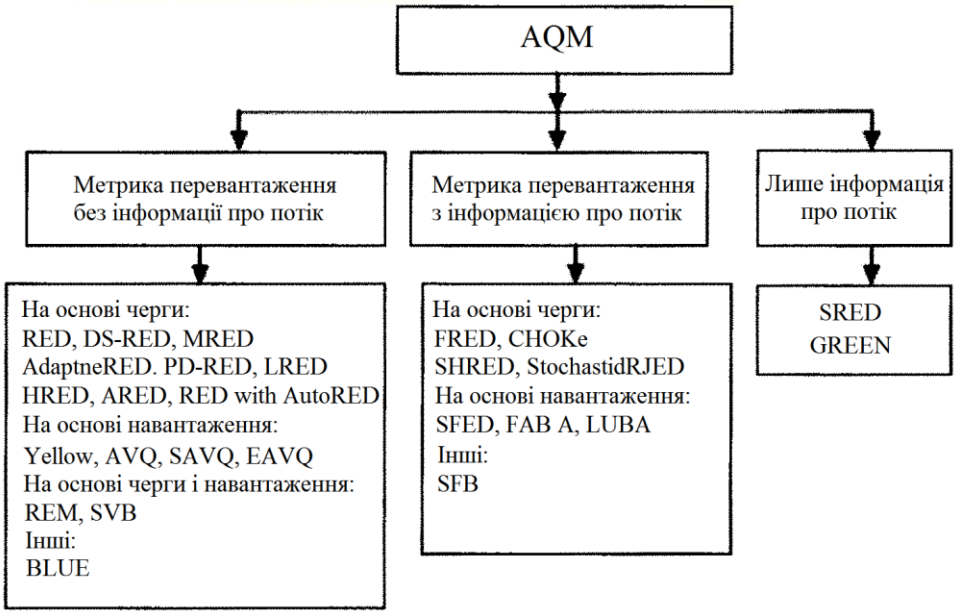

Рисунок 3.3 – Класифікація алгоритмів AQM

Однак механізм може призначати чергу пакетів для кожного вхідного потоку та виконувати планування каналів (хоча можна стверджувати, що цей механізм не є AQM). Крім того, AQM може призначати чергу пакетів для кожного класу трафіку. Щоб охопити ці можливості, таксономія AQM включає кількість черг пакетів для категоризації контролера черги.

4 АЛГОРИТМИ AQM

AQM – це механізм на основі маршрутизатора для раннього виявлення перевантажень у комп'ютерній мережі. Основна ідея AQM полягає в тому, щоб завчасно відчути й виявити перевантаження, а також повідомити відправника про необхідність зменшити швидкість надсилання, таким чином зменшуючи кількість пакетів, що надсилаються в мережі, і контролювати перевантаження. Існує кілька алгоритмів AQM, які контролюють перевантаження.

4.1 Випадкове раннє виявлення (RED)

RED – це алгоритм AQM, який також відомий як випадкове раннє виявлення, який забезпечує механізм для уникнення перевантаження. Традиційний алгоритм відкидання хвоста черги відкидає пакети, якщо буфер заповнений. Алгоритм Drop tail несправедливо розподіляє буферний простір між потоком трафіку. Алгоритм DropTail також може призвести до глобальної синхронізації. Цю проблему можна вирішити в TCP RED.

Лістинг 4.1 – Псевдокод алгоритму RED

```
Крок 1: Begin
Крок 2: Розрахувати середній розмір черги AvgQueSize
Крок 3: if (AvgQueSize>MaxQueSize) скинути пакет
Крок 4: else if (AvgQueSize ==0) порожня черга приймає всі вхідні пакети
Крок 5: else if (AvgQueSize=minqueuethreshold) обчислити ймовірність скидання pa скинути пакет з ймовірністю pa
Крок 6: інакше переслати пакет.
Крок 7: End
```

TCP RED відстежує розмір черги. RED приймає рішення про відкидання пакета: якщо черга порожня, усі пакети приймаються, коли черга заповнюється, ймовірність відкидання пакета також зростає, коли черга заповнюється, усі вхідні пакети відкидаються.

4.2 Стохастична справедлива черга (SFQ)

SFQ – стохастична справедлива черга (SFQ) – це алгоритм AQM, який використовує хешування та циклічний алгоритм. В алгоритмі SFQ потік трафіку ідентифікується чотирма параметрами: адреса джерела, адреса призначення, порт джерела та порт призначення. Ці параметри використовуються алгоритмом хешування SFQ для класифікації пакетів на 1024 підпотоки.

Пропускна здатність рівномірно розподіляється між усіма підпотоками за допомогою циклічного алгоритму. Алгоритм Round Robin розподіляє байти трафіку на кожному раунді, таким чином справедливо розподіляючи доступну пропускну здатність між усіма підпотоками. Черга SFQ містить 1024 підпотоки та 128 пакетів. (Stochastic Fair Queueing).

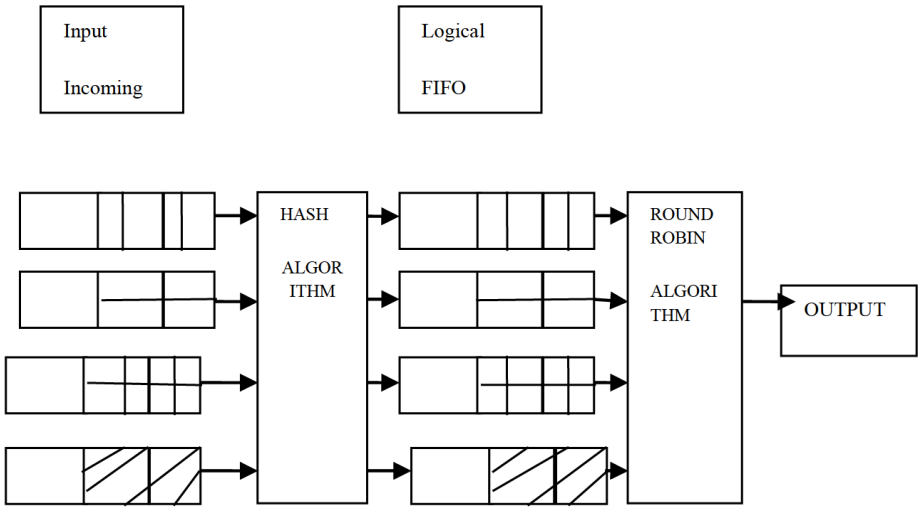

Рисунок 4.1 – Робота алгоритму SFQ

4.3 Випадкове експоненціальне маркування (REM)

Випадкове експоненціальне маркування – це активний алгоритм керування чергою, який відокремлює показники перевантаженості від показників продуктивності, таких як затримка, коефіцієнт доставки пакетів, пропускна здатність, втрата пакетів тощо, і стабілізує показник продуктивності навколо цілі незалежно від кількості користувачів.

REM має дві важливі ключові особливості:

- висока швидкість очищення буферу;
- сумарну вартість.

Алгоритм REM стабілізує швидкість введення із загальною ємністю та довжиною черги з малим цільовим значенням незалежно від кількості користувачів, які спільно використовують канал. Черга виведення REM використовує змінну під назвою «ціна», яка використовується для оцінки ймовірності маркування. Значення цінової змінної оновлюється періодично або асинхронно у двох випадках:

- коли є різниця між вхідною швидкістю та пропускною здатністю зв’язку;
- коли є різниця між довжиною черги та цільовим значенням.

Зважена сума додатна, якщо вхідна швидкість перевищує пропускну здатність зв’язку, інакше зважена сума є від’ємною. Коли зважена сума цих невідповідностей (тобто різниця між швидкістю введення та пропускною спроможністю каналу зв’язку та різницею між довжиною черги та цільовим показником) додатна, значення змінної ціни збільшується, інакше значення змінної ціни зменшується.

Випадок 1: коли кількість користувачів збільшується, швидкість введення зростає, отже, зважена сума є додатною, ціна збільшується, а також зростає ймовірність маркування. У такому випадку відправнику надсилається потужний сигнал перевантаження, щоб зменшити швидкість надсилання.

Випадок 2: коли вхідна швидкість набагато нижча, ніж пропускна

здатність зв’язку, зважена сума буде від’ємною, ціна та ймовірність маркування зменшуються, що підвищує вихідну швидкість, доки невідповідності не зведуться до нуля.

Таким чином, REM чітко контролює значення ціни. Значення ціни оновлюється для довжини черги l в період часу t за такою формулою:

$$pl(t+1)= [\ pl(t) + \gamma(\alpha1(bl(t)- bl^*)+xl(t)-cl(t))], \quad (4.1)$$

де:

$\gamma>0$ – мала константа;

a1>0 – мала константа;

bl(t) – це зайнятість буфера в черзі l за період t;

bl*(t)>=0 цільова довжина черги;

xl(t) – вхідна швидкість;

cl(t) – доступна пропускна здатність черги l за період t;

xl(t)-cl(t) – невідповідність швидкостей;

bl(t)-bl* - це невідповідність черги.

α може бути встановлено кожною чергою залежно від використання пропускної здатності та затримки в черзі. γ контролює чутливість REM для контролю стану мережі.

Коли зважена сума та розбіжності черги додатні, які зважені на α, значення ціни збільшується, інакше значення ціни зменшується.

Ціна може бути стабілізована, коли зважена сума дорівнює нулю, тобто $\alpha1(bl- bl^*)+(xl-cl)=0$, це може статися лише тоді, коли швидкість введення дорівнює пропускній здатності каналу (xl=cl), а довжина черги дорівнює цільовій (bl=bl*) Коли цільова довжина черги b* не дорівнює нулю, швидкість невідповідності xl(t)-cl(t) можна обійти, щоб оновити ціну xl(t)-cl(t), яка зростає, коли довжина черги та буфер не порожній. Отже, апроксимуючи цей член зміною відставання bl(t+1)-bl(t), він стає:

$$pl(t+1)=[pl(t)+\gamma(bl(t+1)-(1-\alpha l)bl(t)-\alpha lb^*)]. \quad (4.2)$$

Отже, видно, що алгоритм REM підвищує ціну, тоді як довжина черги стабілізується навколо цільового bl* незалежно від збільшення кількості користувачів.

Сума цін: Сума цін – це сума всіх цін на канал на шляху від джерела до місця призначення для оцінки ймовірності наскрізного маркування.

Припустимо, що пакет проходить по посиланнях l=1,2,3,------l, які мають ціну pl(t) у період часу t, тоді ймовірність маркування ml(t) у черзі l у період часу t обчислюється як:

$$ml(t)=1-Ф-pl(t) \qquad (4.3)$$

Імовірність наскрізного маркування є високою, коли перевантаження на шляху є великим. Коли ймовірність маркування посилання ml(t) мала, отже, ціни на канал pl(t) малі.

4.4 Стабілізований RED (SRED)

SRED називається стабілізованим алгоритмом RED AQM, який є похідним від алгоритму RED AQM шляхом додавання деяких функцій. Мета алгоритму SRED полягає в тому, щоб визначити потік, який займає більше пропускної здатності, і розподілити справедливу частку пропускної здатності, не виконуючи великих обчислень. Для досягнення цього алгоритм SRED використовує список Zombie, який є невеликим списком нещодавно відвіданих активних потоків з додатковою інформацією для кожного потоку. у списку «кількість» і позначки часу.

Список зомбі спочатку порожній, коли надходить новий пакет, його ідентифікатор пакета (адреса джерела, адреса призначення) додається до списку.

Лічильник встановлюється на нуль, а мітка часу встановлюється на час надходження пакета.

Після заповнення списку зомбі алгоритм SRED порівнює пакет, що надійшов із випадково вибраним зомбі у списку зомбі. Після цього порівняння виконується одна з двох дій:

- щоразу, коли пакет, що надійшов збігається з пакетом у списку зомбі, це є хітом, кількість змінних збільшується на одиницю, а часові позначки встановлюються на час останнього надходження пакета.
- кожного разу, коли новий пакет, що надійшов, не збігається з випадково вибраним пакетом (зомбі) у списку зомбі, це не є збігом або промахом, тоді список зомбі замінюється або записується новим пакетом, що надійшов. Лічильник встановлюється на нуль, а мітка часу встановлюється на час надходження в буфер з імовірністю p.

SRED оцінює p(t) для частоти попадань t-го пакета в буфер.

$$Hit(t)=\begin{cases}0 & \text{if no hit}\\ 1 & \text{if hit}\end{cases}. \tag{4.4}$$

$$p(t)=(1-\alpha)p(t-1)+\alpha\_hit(t), \tag{4.5}$$

де $0<\alpha<1$.

SRED оцінює p(t)-1 для ефективної кількості активних потоків. Припустімо, що є багато потоків з номерами 1,2,3.....n. Припустимо, що кожного разу, коли пакет надходить, він належить до того самого потоку з імовірністю. Тому ймовірність того, що кожен пакет, що надійшов спричинить збіг, дорівнює:

$$P\{Hit(t)=1\}. \tag{4.6}$$

Для зменшення накладних витрат SRED оновлює p(t):

$$0<=p(t)<=1/256.$$

SRED обчислює ймовірність скидання пакету за наступною формулою. Нехай ємність буфера становить B байт. Визначемо функцію Psred(q) наступним чином:

$$Psred(q)=\begin{cases} Pmax & \text{if } 1/3\ B <= q <= B \\ 1/4 * Pmax & \text{if } 1/6\ B <= q <= 1/3\ B \\ 0 & \text{if } 0 <= q <= 1/6B \end{cases},$$

де Pmax вибрано як 0,15;

q – загальна кількість байтів у буфері;

B – ємність буфера в байтах.

Отже, SRED обчислює ймовірність падіння за допомогою наступного рівняння для простого RED:

$$P(zap)= Psed(q) * \min (1, 1/256*P(t)2) \quad (4.7)$$

У цілому, ймовірність скидання пакету в SRED розраховується як:

$$p(zap)= psred(q) * \min(1,1/256*p(t)2) * (1+(Hit(t)/P(t))) \quad (4.8)$$

4.5 Flow RED (FRED)

FRED – це випадкове раннє виявлення на основі потоку, модифікована версія RED. Він використовує облік кожного активного потоку для прийняття рішення про відмову від різних облікових записів активного потоку та для прийняття рішення відкидання для різних активних потоків у черзі залежно від використання пропускної здатності. FRED відстежує кожен потік і використання пропускної здатності кожного потоку всередині черги, тому вартість FRED не залежить від кількості потоків, але пропорційна розміру буфера.

FRED було розроблено як альтернативу алгоритму RED для підтримки високого рівня справедливості. До FRED входять додаткові параметри:

- min q – представляє мінімальну кількість пакетів, які кожному потоку і дозволено буферизувати в черзі;
- max q – представляє максимальну кількість пакетів, які кожному потоку і дозволено буферизувати в черзі;
- avgcq – глобальна змінна, яка оцінює пакети потоку, які будуть буферизовані в черзі (потік і має менше пакетів для постановки в чергу в буфері, якому avgcq надає перевагу, ніж потокам, чія кількість пакетів для постановки в чергу більша, ніж avgcq);
- qlen(i) – змінна підтримує кількість буферизованих пакетів для кожен потоку;
- strike(i) – для кожного потоку і strike(i) є підрахунком кількості разів, коли потік не зміг відповісти на сповіщення про перевантаження (якщо значення strike(i) для потоку збільшується, FRED штрафує такий потік);
- nactive – RED оцінює номер активного потоку служби очищення за допомогою змінної nactive FRED..

4.6 BLUE AQM

BLUE – це алгоритм AQM, у якому керування чергою здійснюється на основі використання каналу та кількості відкинутих пакетів. BLUE підтримує змінну pm, щоб оцінити ймовірність маркування або для позначення пакета, або для скидання пакета. Коли черга заповнюється, пакети починаються скидатися.

Коли черга заповнюється, пакет починає скидатися, а pm збільшується на коефіцієнт δ1. Якщо черга порожня, pm зменшується на коефіцієнт δ2. Значення δ1 встановлюється таким чином, щоб δ1 > δ2

Blue використовує ще один параметр під назвою freeze_time, який визначає інтервал часу між двома послідовними оновленнями freeze_time.

Вхід: пакети до BLUE черги.

Процес: перевірка, чи заповнена черга, якщо правда, видалити пакети та збільшити pm на δ1, інакше зменшиіти pm на δ2.

Вихід: успішно вилучити пакет із черги до місця призначення.

Лістинг 4.2 – Псевдокод алгоритму BLUE

```
Крок 1- BEGIN
Крок 2- У разі втрати пакета або (Qlen>L) event
if((now_last_update)>freeze_time) //перевірка
pm=pm+51
last_update=now
Крок 3 - у разі простою посилання
if(now_lastupdate)>freeze_time) //перевірка
pm=pm-52
last_update=now
Крок 4- END
```

4.7 Stochastic Fair Blue (SFB)

SFB – це ще один алгоритм AQM, який захищає потоки TCP від агресивного потоку за допомогою алгоритму BLUE AQM.

Алгоритм SFB визначає та обмежує швидкість потоку, що не відповідає, а механізм, який використовується для ідентифікації цього потоку, який не відповідає, такий самий, як механізм обліку, який використовується в алгоритмі BLUE.

SFB підтримує N*L облікових комірок, де L – це кількість рівнів, а N – кількість комірок на кожному рівні.

SFB також підтримує L незалежних хеш-функцій, кожну з яких пов’язано з одним рівнем комірки обліку. SFB підтримує змінну під назвою pm, яка відстежує ймовірність позначення/скидання в кожній комірці. Коли надходить новий пакет, він відображається в одній з N комірок на кожному з

l рівнів. Коли кількість пакетів, відображених у комірці, перевищує певне порогове значення pm, збільшується. Якщо кількість пакетів падає до нуля, pm зменшується.

Вхід: пакети до N * L облікових комірок.

Процес: вхідний пакет відображається в одній з N облікових комірок на рівні L. Перевіряє, чи кількість пакетів, зіставлених на облікові комірки, перевищує розмір комірки, pm для комірки збільшується, а пакет відкидається, інакше pm для комірки зменшується.

Вихід: успішно вилучити пакет із черги до місця призначення.

Лістинг 4.2 – Псевдокод алгоритму SFB

```
Крок 1 : Begin
Крок 2:Ініціалізація B[l][n]: L*N масиви комірок (L levels, N
bins per level)
Крок 3: Виклик enqueue функції
Крок 4: Обчислити значення хеш-функції h0,h1
Крок 5: оновити комірки на кожному рівні
Крок 6: for i=0 to L-1
Крок 7: перевірити if(B[i][hi]*pm+=delta
        Скинути пакет
Крок 8: else if(B[i][hi]*qlen==0)
        B[i][hi]pm==delta
Pmin=min(B[0][h0].pm.... B[L][hL]*pm)
Крок 9: if(pmin==1)
        ratelimit()
Крок 10: else
         mark/drop with probability pmin
Крок 11: End
```

SFB статистично мультиплексує буфер до бункерів, але потребує перенастроювання з великою кількістю невідповідаючих потоків.

5 УДОСКОНАЛЕНИЙ АЛГОРИТМ ДЛЯ СИСТЕМИ AQM

Існуючі AQM були модифіковані та покращені, але вони все ще страждають від великої втрати пакетів під час пікового періоду перевантаження. Щоб зменшити цю високу втрату пакетів, IETF розглядає можливість розгортання явного сповіщення про перевантаження разом із AQM.

5.1 Тенденції розвитку AQM

З швидким розвитком комп'ютерного обладнання та передачі даних якість обслуговування (QoS) у комп'ютерних мережах стала важливою проблемою для кінцевих користувачів [1]. QoS можна визначити як продуктивність мережі для різних послуг, які шукає користувач під час проходження даних через цю мережу [1]. QoS має різні рівні, які надаються користувачам на основі вимог додатків, що використовуються, постачальників мережевих послуг або угоди про рівень обслуговування (SLA) між користувачами [2]. Найкраща служба – це одна із послуг, яка використовується в Інтернеті для доставки пакетів і не розрізняє пакети, створені різними класами послуг [3]. QoS можна оцінити за різними показниками, такими як пропускна здатність, втрата пакетів, тремтіння та затримка.

У сучасних комунікаційних і комп'ютерних мережах розроблено різні мережеві програми, такі як передача голосу через IP (VoIP), відеоконференції, живе відео, електронна пошта, передача файлів. Ці програми потребують різного QoS.

Наприклад, VoIP, відеоконференції та відео в прямому ефірі вимагають високої пропускної здатності, низької затримки та тремтіння. Крім того, мережеві програми мають низьку чутливість до втрати пакетів. З іншого

боку, програми електронної пошти та передачі файлів мають високу чутливість до вимог втрати пакетів і низьку чутливість до пропускної здатності, затримки та тремтіння. Підвищення продуктивності мережі можна виконати на основі отримання QoS для цієї мережі.

RED є одним із ключових методів AQM, які були запропоновані для підвищення продуктивності класичного методу скидання пакетів. Тим не менш, незважаючи на перевагу RED над методом скидання хвоста черги, прогрес у різноманітності обслуговування трафіку виявив наступні проблеми продуктивності, пов’язані з продуктивністю RED [2].

- зазвичай ключовий показник перевантаження (середня довжина черги (aql)), який використовує RED, змінюється відповідно до рівня перевантаження (наприклад, коли aql близький до мінімального порогу, виникає невелике перевантаження, тоді як, якщо aql близько максимального порогу, виникає сильний затор, а це може спричинити переповнення буфера маршрутизатора та, отже, відкидання пакетів, що надходять);
- іншою проблемою є налаштування певних параметрів RED на конкретні значення, тобто максимальне значення ймовірності відкидання пакетів (D_{max}), щоб забезпечити задовільну продуктивність, але це призводить до упереджених результатів;
- часто aql залежить від кількості TCP-з'єднань (коли кількість TCP-з’єднань збільшується, aql також збільшується і може перевищити максимальне порогове значення; отже, кожен надійшовший пакет відкидається.

Далі розглядається проблема, описана вище, яка полягає в налаштуванні D_{max} на певне значення, щоб гарантувати високу, але упереджену якість обслуговування.

Бажано мінімізувати цю проблему, запропонувавши новий метод, який пом’якшує залежність від D_{max} і замість цього використовує нову експоненціальну міру під назвою D_{init}, яка базується на середній довжині черги (aql). Зокрема, коли aql знаходиться між мінімальним і максимальним

пороговими значеннями, запропонований метод контролю перевантажень експоненціально відкидає прибулі пакети. Це призведе до більш реалістичних результатів продуктивності, а не упереджених.

Запропонований метод (RED_E) реалізовано за допомогою моделювання та підходу черг з дискретним часом. Підхід дискретних черг використовується для моделювання надходження та відправлення пакетів у кожній одиниці часу, яка називається слотом. Переваги запропонованого алгоритму RED_E такі:

- розрахунок Dinit йде без використання параметра Dmax, що зменшує суб'єктивність користувачів під час налаштування параметрів RED;
- покращені показники продуктивності mql і D у порівнянні з RED і одним із методів AQM на основі RED, таким як NLRED, особливо у випадках, коли значення ймовірності надходження пакету є дуже високим.

5.2 Деталізований огляд RED та NLRED

5.2.1 RED і нелінійний NLRED

RED відноситься до методів AQM, які були запропоновані для виявлення та контролю заторів [7]. RED покладається на певні параметри для розрахунку свого значення D_p. Ці параметри: мінімальний поріг, максимальний поріг, D_{max} і qw.

Для кожного пакета, що надійшов до буфера маршрутизатора, RED обчислює значення aql. Коли значення aql менше мінімального порогу, перевантаження не відбувається, і, таким чином, жодні пакети не можуть бути відкинуті.

Однак, якщо значення aql менше максимального порогу та дорівнює мінімальному порогу або перевищує його, це вказує на наявність перевантаження, і буфер маршрутизатора ймовірно відкидає пакети, що надходять. Нарешті, коли значення aql дорівнює або перевищує максимальне

порогове значення, виникає сильна перевантаженість, і кожен пакет, що надходить, буде відкинуто, щоб впоратися з цим перевантаженням. Це досягається або скиданням пакетів, що надійшли, у буфер маршрутизатора, або позначенням їх за допомогою явного повідомлення про перевантаження (ECN).

Параметри алгоритму RED пояснюються нижче:

- current_time – поточний час;
- idle_time – початок простою в буфері маршрутизатора RED;
- n – кількість пакетів, надісланих до буфера маршрутизатора RED протягом інтервалу простою;
- C – кількість пакетів, які надійшли до буфера маршрутизатора RED, але не скинуто з моменту скидання останнього пакета;
- D_p – ймовірність миттєвого відкидання пакетів;
- D_{init} – початкова ймовірність відкидання пакета;
- q_instantaneous – миттєва довжина черги;
- qw – вага черги;
- D_{max} – максимальне значення D_p;
- q(time) – лінійна функція для часу.

Однією з головних проблем RED є те, що налаштування параметрів не може гарантувати стабільну продуктивність за змінних навантажень трафіку. Це можна пояснити лінійної функцією ймовірності скидання пакетів, яка часто є агресивною, коли навантаження трафіку невелике, і не агресивною, коли трафік дуже великий, що може призвести до того, що значення середньої довжини черги досягне значення максимального порогу.

Щоб подолати цю проблему, [14] запропонував Nonlinear RED (NLRED), який використовує нелінійну функцію відкидання пакетів, подібну до запропонованої в [4].

У випадках, коли aql знаходиться між мінімальним порогом і максимальним порогом, тоді NLRED використовує квадратичну функцію для обчислення ймовірності відкидання пакета показано у формулі (5.1).

$$D_{init} = D'_{max} \cdot \left(\frac{\text{aql-min threshold)}}{\text{max threshold- min threshold}} \right)^2, \quad (5.1)$$

де D'_{max} – максимальне значення ймовірності відкидання пакету. Якщо використовується те саме значення для D_{max} в RED і D'_{max} в NLRED, тоді NLRED буде м'якішим, ніж RED для всіх навантажень трафіку, оскільки значення ймовірності скидання пакетів NLRED менше, ніж значення RED. Якщо загальна ймовірність відкидання пакетів RED і NLRED однакова, тоді значення D'_{max} буде встановлено таким чином:

$$D'_{max} = 1{,}5 \cdot D_{max}. \quad (5.2)$$

Функції ймовірності скидання пакетів методів RED і NLRED показані на малюнку 1.

5.2.2 Інші методи AQM

Щоб усунути перший недолік RED, aql змінюється відповідно до рівня перевантаження, тому адаптивна методика RED (ARED) була вперше запропонована в [8].

ARED мав на меті стабілізувати значення aql на певному рівні між мінімальним і максимальним пороговими значеннями. Це може запобігти відкиданню великої кількості пакетів.

Техніка Gentle RED (GRED) [9] була запропонована для зменшення кількості пакетів, що відкидаються. Це було досягнуто шляхом запобігання відкиданню кожного пакета, що надходить, коли значення aql дорівнює позиції максимального порогу, як у RED; натомість GRED відкидає прибулі пакети ймовірнісно на основі значень у діапазоні від D_{max} до 1,0. Алгоритм динамічного випадкового раннього скидання (DRED) [5] був

запропонованим для роботи з залежністю aql від кількості з’єднань TCP; Технологія BLUE [6] була розроблена, щоб запропонувати кращі вимоги до мережі щодо швидкості втрат пакетів і розміру черги, ніж вимоги RED.

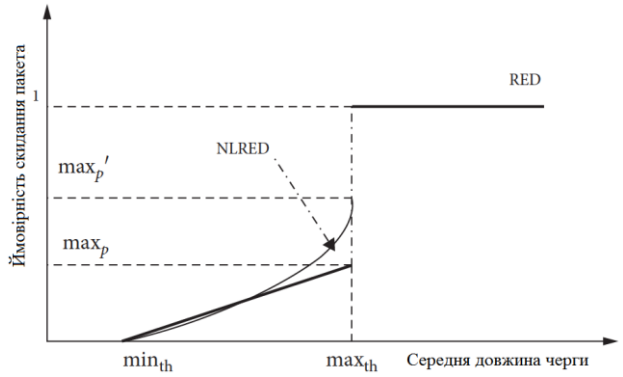

Рисунок 5.1 – Функції ймовірності скидання пакетів алгоритмів RED і NLRED

Ефективна техніка RED (ERED) була запропонована для зниження рівня втрати пакетів простим і масштабованим способом. Було внесено кілька змін у функцію скидання пакетів RED, а решта параметрів RED залишаються незмінними. Зміни вдосконалили функцію відкидання пакетів, використовуючи параметри середнього розміру черги та миттєвого розміру черги.

Псевдокод техніки RED показаний в лістингу 5.1. Лістинг 5.1 показує, що RED використовує кілька факторів для обчислення aql і D_p.

Алгоритм AQM на основі RED був запропонований [15]. Цей покращений адаптивний алгоритм RED використовує нелінійну згладжену функцію для коефіцієнта втрати пакетів через використання нечіткого розподілу. В алгоритмі ARED збільшення швидкості втрати пакетів поблизу максимального порогового значення є швидким, тоді як воно повільне поблизу мінімального порогового значення. Крім того, максимальне значення ймовірності відкидання пакетів (D_{max}) ARED зміняє розмір середньої черги та цільове значення для адаптації зміни умов мережі.

Лістинг 5.1 – Псевдокод алгоритму RED

```
Етап ініціалізації: C=-1, aql=0.0
Для кожного пакета, що надійшов, у буфер маршрутизатора RED
Обчислити aql для цього пакета в буфері маршрутизатора RED
Перевірити статус черги в буфері маршрутизатора, порожня чи ні:
  If queue==Null    // якщо черга в буфері RED порожня тоді:
    n=q(current_time-idle_time)
    aql = aql*(1-qw)^n
  Else
    aql=aql*(1-qw)+qw*q_instantaneous
Визначити стан перевантаження в буфері маршрутизатора RED:
  if aql<min_threshold then
    Обчислити значення Dp для прибулого пакета наступним чином:
    Dp=0.0
    C=-1
  Else if aql≥min_threshold && aql<max_threshold then
    C=C+1
    Dinit=(Dmax*(aql-min_threshold))/(max_threshold-min_threshold)
    Dp=Dinit/(1-C*Dinit)
    Позначити/відкинути прибулий пакет імовірнісно Dp
    через виникнення перевантаження:
    C=0
  Else
    Позначати/відкидати кожен пакет, що надійшов, Dp=1,0 через
    виникнення сильного перевантаження:
    C=0
Коли буфер маршрутизатора RED стає порожнім:
  idle_time=current_time
```

Псевдокод методу NLRED наведено в лістингу 5.2. Лістинг 5.2 показує, що буфер маршрутизатора NLRED працює як буфер маршрутизатора RED, коли значення aql менше мінімального порогу або більше, або дорівнює максимальному порогу.

Лістинг 5.1 – Псевдокод алгоритму NLRED

```
Для кожного пакету, що надійшов
розрахувати середню довжину черги (aql):
  if aql≤M                                    //M в порозі
    Жоден пакет не буде скинутий
  else if M<aql<max_threshold
    Обчисліти імовірність скидання пакетів за допомогою
    квадратичної функції
    Відкинути пакет, щонадійшов із обчисленою ймовірністю
  else
    Скинути прибулий пакет
```

Метод, подібний до автоматів навчання, для уникнення випадків перевантаження в мережах під назвою LALRED був запропоновано в минулому. Мета LALRED полягала в тому, щоб збільшити значення середнього розміру черги, що використовується для контролю перевантаження, і таким чином зменшити загальну втрату пакетів у черзі.

Аналітична модель для черг RED із використанням змішаних типів трафіку, таких як TCP і протокол дейтаграм користувача (UDP), була представлена науковцями раніше.

Були отримані стаціонарні вирази хорошої пропускної здатності для кожного потоку та середньої затримки в черзі. Аналітичну структуру було розширено, щоб включити клас черг RED, який пропонує різні послуги для потоків з кількома класами.

Було запропоновано адаптивне керування чергою з методом випадкового відкидання, яке використовує інформацію про середню довжину черги та швидкість її зміни. Цей метод отримав назву AQMRD, і він використовує швидкість зміни довжини черги як додатковий параметр для виявлення перевантаження. Цей метод дозволяє уникнути того, що середня довжина черги часто перевищує максимальне порогове значення, швидко реагуючи на перевантаження, таким чином уникаючи переповнення буфера.

RED чутливий до своїх параметрів і трафіку, тому, коли навантаження трафіку низьке, пропускна здатність використовується недостатньо. Однак, коли навантаження на трафік велике, це призводить до великої затримки.

5.3 Запропонований алгоритм RED_E

Як згадувалося раніше, RED має недоліки, які сприяють погіршенню його продуктивності і проблема полягає в попередньому налаштуванні параметрів RED, щоб забезпечити досягнення доброї продуктивності без необхідності налаштування параметрів на певне значення, зокрема, D_{max}. Тому було розроблено метод RED_E, який використовує aql як міру перевантаження. Проте він відрізняється способом відкидання пакетів щоразу, коли пакет надходить до буфера маршрутизатора, особливо коли значення aql дорівнює або перевищує мінімальний поріг і менше максимального порогу. У цьому сценарії класичний RED відкидає пакети ймовірнісно, використовуючи наступні рівняння:

$$D_{init} = D_{max} \cdot \frac{\text{aql-min threshold}}{\text{max threshold- min threshold}}, \tag{5.3}$$

$$D_p = \frac{D_{init}}{(1\text{-}C \cdot D_{init})}. \tag{5.4}$$

У рівняннях (5.3) і (5.4) D_{init} є початковою ймовірністю відкидання пакета, а в рівнянні (5.4) C являє собою кількість пакетів, які надійшли до буфера маршрутизатора і не були відкинуті з моменту відкидання останнього пакета. З іншого боку, RED_E, як показано в лістингу 5.3, не використовує параметр D_{max} в обчисленні D_{init}, який використовується в RED і багатьох його наступниках. Замість обчислення D_{init} як рівняння (5.3), RED_E використовує власний розрахований D_{init}, (рівняння 5.5).

$$D_{init} = \frac{(e^{aql} - e^{min\ threshold})}{(e^{max\ threshold} - e^{min\ threshold})}, \quad (5.5)$$

У рівнянні (5.5) можна побачити, що RED_E більше не використовує параметр D_{max}, який зазвичай встановлюється на попередній стадії, як у RED, щоб гарантувати задовільну продуктивність.

RED_E експоненціально відкидає пакети, що надходять, використовуючи рівняння (5.4) і (5.5). На рисунках 5.2 і 5.3 показано механізм D_{init} і механізм aql для RED і RED_E відповідно.

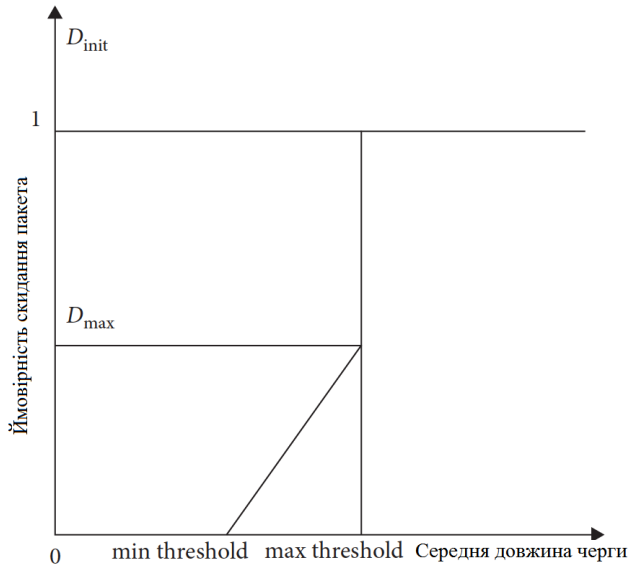

Рисунок 5.2 – D_{max} и aql в алгоритмі RED

Запропонована методика експоненціально збільшує значення Dinit від 0,0 до 1,0, коли значення aql збільшується від мінімального порогового значення до максимального порогового значення.

Псевдокод RED_E, наведений в лістингу 5.3 має й інші цілі, окрім усунення зміщених результатів шляхом попереднього встановлення параметрів, наприклад, забезпечення більш задовільної продуктивності в сценаріях сильного перевантаження.

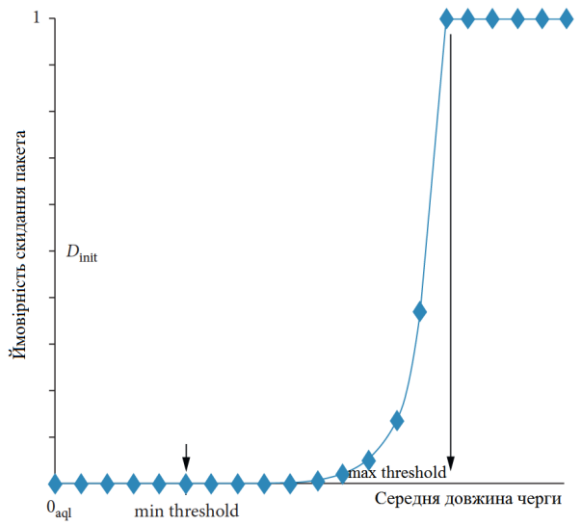

Рисунок 5.3 – D_{init} и aql в алгоритмі RED_E

5.4 Результати моделювання

5.4.1 Налаштування симуляції

Методи RED, NLRED і RED_E моделюються з використанням системи єдиного вузла черги. Один пакет може надходити та/або відправлятися в одиницю часу, що називається слотом, який використовується в черзі дискретного часу.

Реалізації RED, NLRED і запропонованих методів проводяться на основі черг дискретного часу в середовищі Java. Час між надходженнями пакетів і час обслуговування геометрично розподілені із середніми значеннями 1/α і 1/β відповідно, де α – ймовірність надходження пакету в слот, а β – ймовірність відправлення пакета зі слота.

Процес надходження, який використовується в обох методах, є процесом Бернуллі.

На рисунку 5.4 кінцева ємність одного вузла черги для RED, NLRED або RED_E становить K - пакетів. Перший прийшов, перший обслуговується (FCFS) – це дисципліна черги, яка використовується в RED, NLRED або RED_E.

Лістинг 5.3 – Псевдокод алгоритму RED_E

```
Етап ініціалізації: C=-1, aql=0.0
Для кожного пакета, що надійшов, у буфер маршрутизатора RED_E
Обчислити aql для цього пакета в буфері маршрутизатора RED_E
Перевірити статус черги в буфері маршрутизатора, порожня чи ні:
  If queue==Null    //якщо черга в буфері RED_E порожня тоді:
    n=q(current_time-idle_time)
    aql = aql*(1-qw)^n
  Else
    aql=aql*(1-qw)+qw*q_instantaneous
Визначити стан перевантаження в буфері маршрутизатора RED_E:
  if aql<min_threshold then
    Обчислити значення Dp для прибулого пакета наступним чином:
    Dp=0.0
    C=-1
  Else if aql≥min_threshold && aql<max_threshold then
    C=C+1
    Dinit=(e^aql-e^min_threshold))/(e^max_threshold-e^min_threshold)
    Dp=Dinit/(1-C*Dinit)
    Позначити/відкинути прибулий пакет імовірнісно Dp:
    C=0
  Else
    Позначати/відкидати кожен пакет, що надійшов, Dp=1,0 через
    виникнення сильного перевантаження:
    C=0
Коли буфер маршрутизатора RED стає порожнім:
  idle_time=current_time
```

5.4.2 Налаштування параметрів

Параметри RED, NLRED і RED_E представлені в таблиці 5.1. У таблиці 5.1 значення α (ймовірність надходження пакету) змінюються між 0,18 і 0,93, причому половина цих значень (тобто 0,18, 0,33 і 0,48) є меншими

за β (ймовірність відправлення пакета зі слота), а решта (тобто 0,63, 0,78 і 0,93) є більшими за β. Ці значення α і β були встановлені для перевірки результатів вимірювання продуктивності, коли α<β і α>β (сценарії перевантаження). K встановлено на 20 пакетів для перевірки перевантаження з малими розмірами буфера. Максимальний поріг утричі перевищує мінімальний поріг. Для qw і D_{max} у таблиці встановлено значення 0.1. Dmax′ встановлюється на значення, обчислене за допомогою рівняння (5.2). Задане значення кількості слотів для того, щоб гарантувати, що воно досягне сталого стану.

Таблиця 5.1 – Початкові параметри

Параметр	Значення
α	0.18-0.93
β	0.5
K	20
min threshold	3
max threshold	9
qw	0.002
D_{max}	0.1
Кількість слотів	2,000,000

У цій симуляції використовуються спеціальні параметри, такі як ймовірність надходження пакетів і ймовірність відправлення пакетів у чергах з дискретним часом. Крім того, ці особливі речі можна застосовувати в середовищах моделювання, таких як Java, і з цієї причини Java була обрана як середовище моделювання методів RED, NLRED і RED_E.

Всі джерела трафіку під’днані до маршрутизаторів дуплексними зв’язками 10,0 Мб та затримкою 10 мс. Симплексні зв’язки створюються в двох різних напрямках. Також, поміж двома вузлами R1 та R2 формуються два окремих симплексних канали, із пропускною здатністю 10 Мб. Таких установок цілком достатньо, щоб сформувати в мережі вузьке місце.

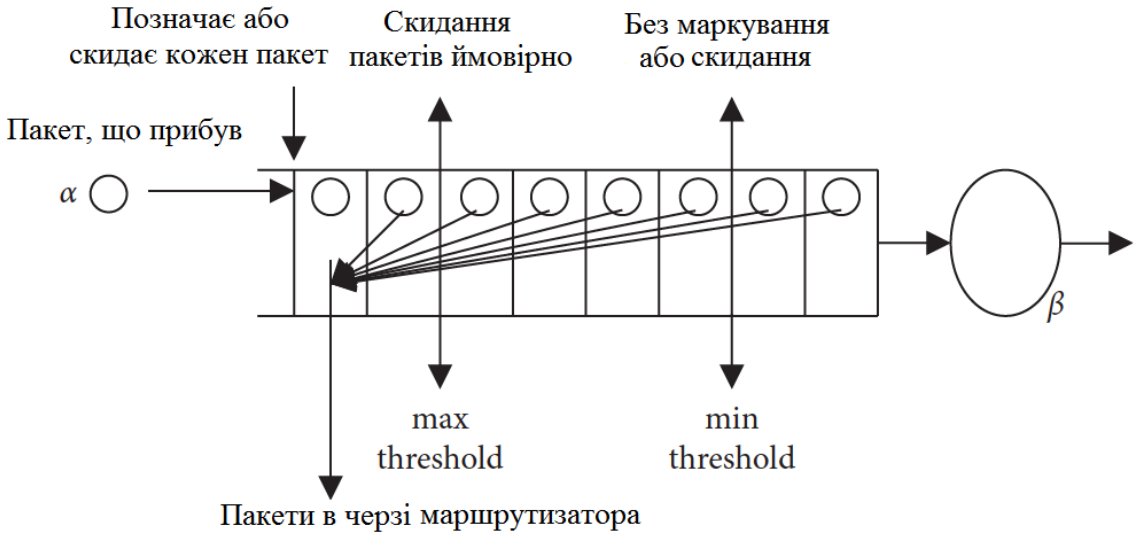

Рисунок 5.4 – Окремий буфер маршрутизатора для RED, NLRED або запропонованого RED_E

На зв’язку між маршрутизаторами задається пропускна здатність 10,0 Мб з симплексним зв’язком та затримкою 10 мс. Від маршрутизаторів до інших вузлів мережі використовується смуга пропускання 10 Мб з дуплексним зв’язком.

5.4.3 Аналіз результатів

Перед генеруванням результатів вимірювання ефективності проводиться період розминки. Коли система досягає стійкого стану, отримуються результати вимірювання продуктивності.

Кожен результат вимірювання продуктивності представляє середнє арифметичне десяти прогонів часу для кожного значення α. Для кожного запуску початкове значення змінюється за допомогою генератора випадкових чисел, щоб видалити будь-які упереджені результати. Рішення про те, який метод дає кращі результати, приймається лише за допомогою встановлених значень α.

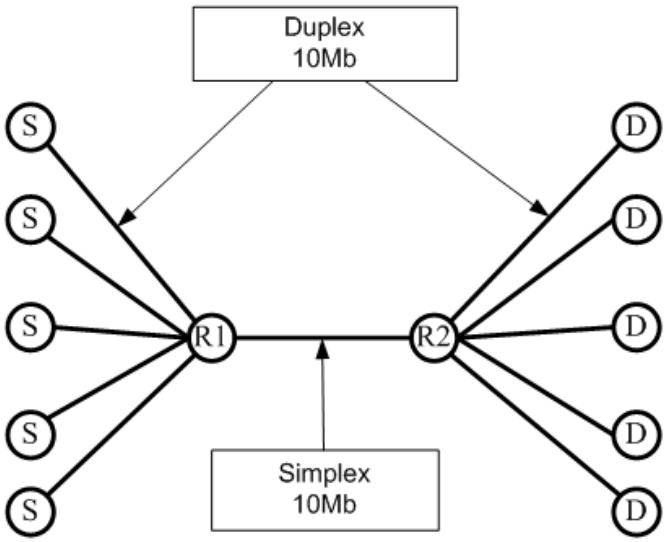

Рисунок 5.5 – Мережева модель

Алгоритми RED, NLRED і RED_E порівнюються з такими параметрами продуктивності: mql, T, D, PL і DP. Позначки абревіатури:

- mql позначає середню довжину черги;

- T – пропускна здатність, яка представляє кількість пакетів, які успішно пройшли через чергу вузол за кожну одиницю часу;

- D – середня затримка в черзі для пакетів;

- P_L – ймовірність втрати пакетів через переповнення буфера;

- D_P – ймовірність скидання пакетів до заповнення буфера маршрутизатора.

Це порівняння має на меті оцінити продуктивність RED_E у різних ситуаціях перевантаження та без використання параметра D_{max}.

На рисунках 5.6 – 5.10 показано результати вимірювання продуктивності (mql, T, D, PL і Dp) у порівнянні з α для RED, NLRED і RED_E.

Використовується стовпчаста діаграма, а не точкова діаграма, оскільки результати вимірювання ефективності порівнюваних методів дещо відрізняються на основі α, і ці відмінності видно чіткіше за допомогою типу стовпчастої діаграми.

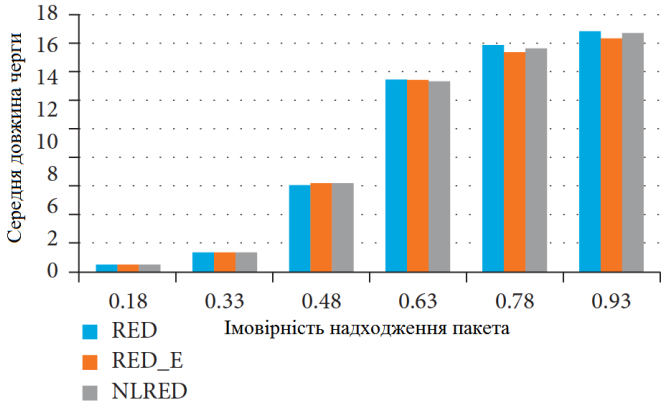

Рисунок 5.6 – Залежність середньої довжини черги mql від ймовірності надходження пакета α

Результати оцінки продуктивності виконуються на основі змінних значень α. Після аналізу рисунків 5.6 і 5.7 RED, NLRED і запропонованого RED_E дають подібні результати D і D, коли α менше або дорівнює 0,33. Іншими словами, RED і RED_E дають схожі результати mql і D у ситуаціях відсутності перевантаження в буфері маршрутизатора вузла черги. Це пов’язано з тим, що RED і RED_E відкидають однакову кількість пакетів до того, як буфер маршрутизатора заповниться (рисунок 5.8), тобто нуль, і втрачає однакову кількість пакетів через переповнення (рисунок 5.9), тобто, нуль.

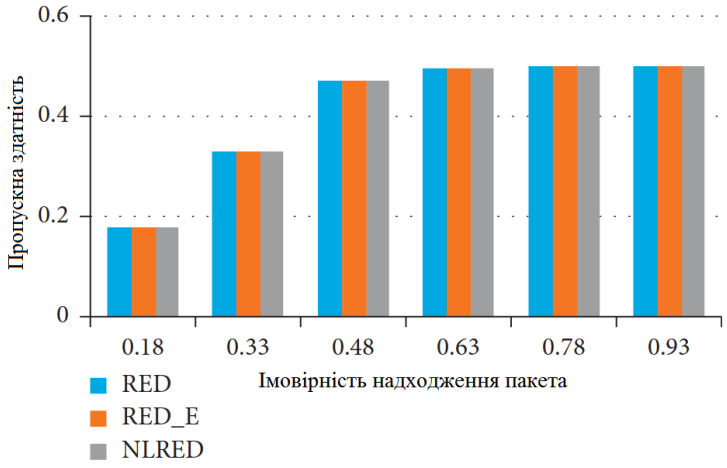

Рисунок 5.7 – Залежність пропускної здатності T від α

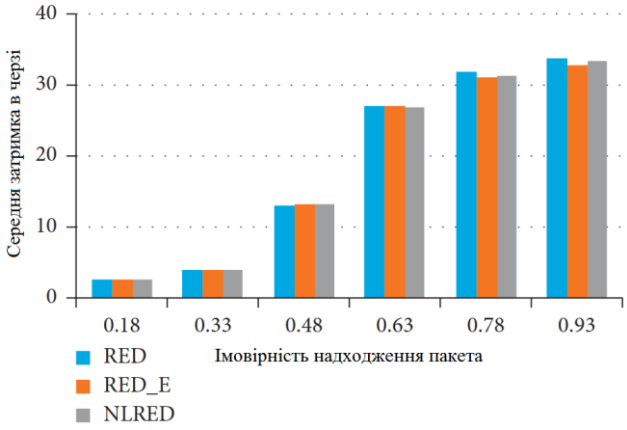

Рисунок 5.8 – Залежність середньої затримки в черзі від ймовірності надходження пакета α

Коли α більше за 0,33 і менше або дорівнює 0,48, RED забезпечує трохи нижчі результати mql і D, ніж NLRED і RED_E, оскільки RED відкидає трохи більше пакетів, ніж NLRED і RED_E, перш ніж буфер заповниться. RED_E і NLRED дали схожі результати щодо mql і D тому що RED_E і NLRED відкидають подібні пакети до того, як буфер заповниться. Також, RED, NLRED і RED_E втрачають порівнянну кількість пакетів через переповнення (PL). Коли α більше за 0,48 і менше за 0,63, NLRED забезпечує дещо менші результати mql і D серед порівнюваних методів через згадану раніше причину. RED і RED_E мають аналогічні результати mql і D.

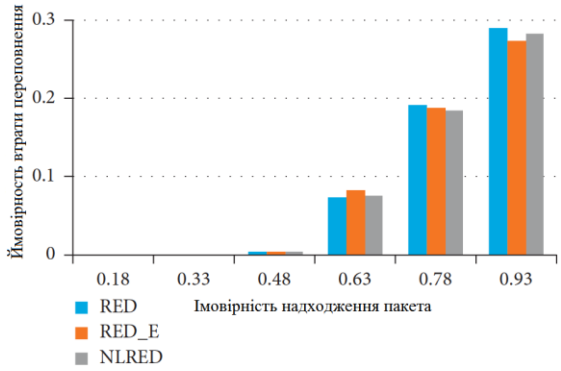

Рисунок 5.9 – Залежність ймовірності втрати переповнення від ймовірності надходження пакета α

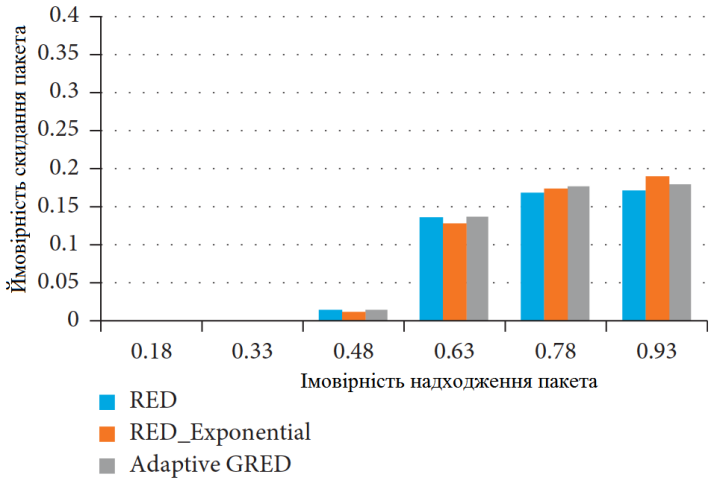

Рисунок 5.10 – Залежність ймовірності скидання пакетів до заповнення буфера маршрутизатора від ймовірності надходження пакета α

RED і NLRED втрачають менше пакетів, ніж RED_E тому, що буфери маршрутизатора RED і NLRED переповнюються у кілька разів менше ніж у RED_E. Крім того, у RED_E менше відкинутих пакетів, ніж RED і NLRED, і RED і NLRED відкидають пакети до того, як їх буфери заповняться.

Коли α перевищує 0,63, наприклад α=0,78, RED_E досягає найзадовільнішої вимірюваної продуктивності серед порівнюваних методів з точки зору результатів mql і D, оскільки RED_E підтримує меншу середню довжину черги. Крім того, NLRED певною мірою дав кращі результати для mql і D, ніж RED через згадану вище причину. RED забезпечує вищий результат P_L, ніж NLRED і RED_E, оскільки буфер RED переповнюється частіше, ніж NLRED і RED_E. NLRED має знижений результат P_L, ніж RED_E. Крім того, RED відкидає менше пакетів, ніж два інші методи, а RED_E має трохи кращий результат D_p, ніж NLRED.

За наявності сильного перевантаження, такого як α>0,78, наприклад α=0,93, RED_E досягає нижчих результатів mql, D і P_L, ніж RED або NLRED, через те, що RED_E відкидає більшу кількість пакетів (D_p), ніж RED або NLRED. Крім того, NLRED дає кращі результати mql, D і P_L, ніж RED, оскільки NLRED відкидає більше пакетів, ніж RED.

Були проведені подальші тести моделювання між RED, NLRED і RED_E на основі різних значень мінімального порогу для оцінки їх впливу на результати продуктивності.

α було встановлено на 0,78, оскільки це значення може спричинити сильний затор, і нам потрібно було оцінити ефективність мінімального порогового параметра для порівнюваних методів за наявності сильного затору. Мінімальне порогове значення було встановлено на різні значення в діапазоні від 3 до максимального порогового значення 8, щоб спостерігати за ефективністю кожного мінімального порогового значення на результати вимірювання ефективності. Результати вимірювання продуктивності RED, NLRED і RED_E порівняно з мінімальними пороговими значеннями наведені на рисунках 5.11-5.15.

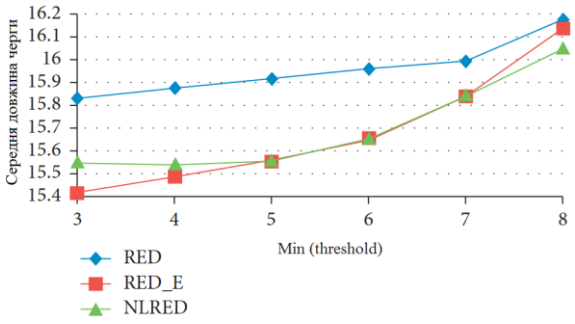

Рисунок 5.11 – Залежність mql від мінімального порогу

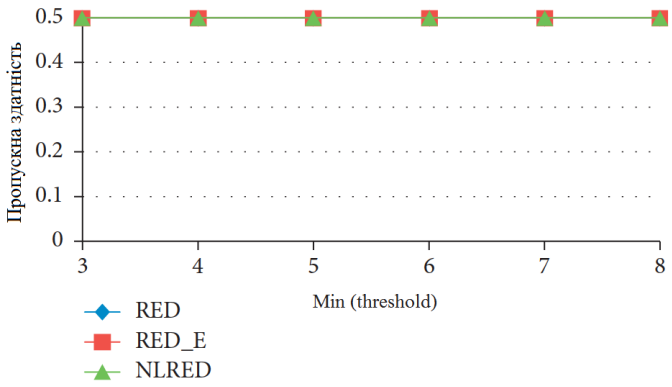

Рисунок 5.12 – Залежність пропускної здатності від мінімального порогу

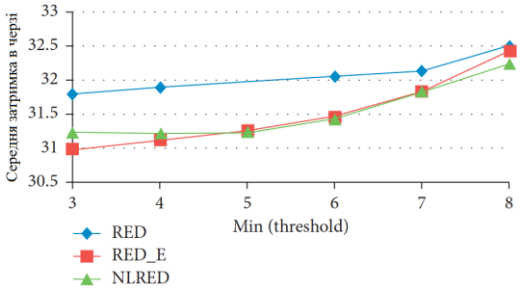

Рисунок 5.13 – Залежність середньої затримки від мінімального порогу

З рисунків 11, 13 і 14 видно, що RED пропонує вищі результати mql, D і P_L, ніж NLRED і RED_E для всіх мінімальних порогових значень. Це пояснюється тим, що буфер маршрутизатора RED скидає менше пакетів, ніж буфер NLRED і RED_E (рисунок 15). Крім того, RED_E забезпечує менші результати mql і D, ніж NLRED, коли мінімальне порогове значення дорівнює 3 або 4, і ці значення представляють найдальші значення від максимального порогу.

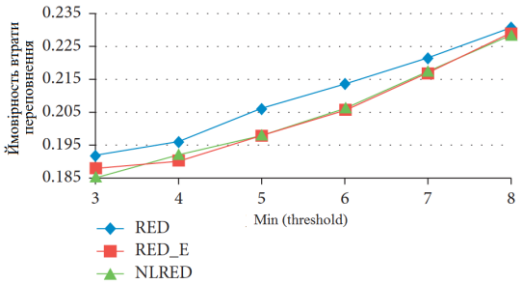

Рисунок 5.14 – Залежність ймовірності втрати пакетів через переповнення буфера від мінімального порогу

Якщо значення мінімального порогового значення встановлено на 8 (найближче значення до максимального порогового значення), тоді NLRED генерує нижчі результати mql і D, ніж результати RED_E; це пов’язано з тим, що довжина середньої черги NLRED менша, ніж RED_E. Результати продуктивності NLRED і RED_E щодо mql і D порівняно подібні, якщо

мінімальне порогове значення встановлено на 5, 6 і 7 (на півдорозі між мінімальним порогом і максимальним порогом). NLRED створює менший P_L, ніж RED_E, коли мінімальне порогове значення встановлено на найдальше значення від максимального порогу, тоді як отримує менший P_L, ніж NLRED, коли мінімальний поріг заданий як 4.

З рисунка 15 видно, що RED відкидає менше пакетів, ніж NLRED або RED_E, оскільки і NLRED, і RED_E пропонують менші результати mql, ніж RED. RED_E отримує менший D_P, ніж NLRED, коли мінімальне порогове значення надається значенню, яке є найбільшим значенням від максимального порогового значення. Однак NLRED досягає меншого D_P, ніж RED_E, коли мінімальне порогове значення встановлено на 4.

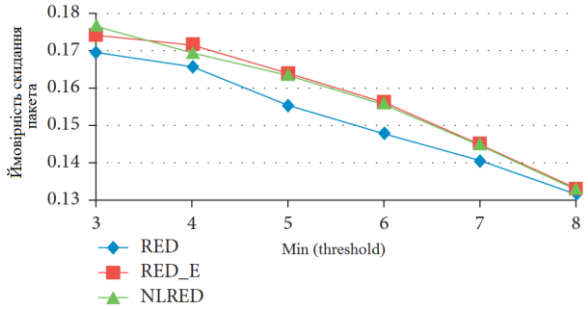

Рисунок 5.15 – Залежність ймовірність скидання пакетів D_L від мінімального порогу

Для інших мінімальних порогових значень (5, 6, 7 і 8) і NLRED, і RED_E генерують подібні результати P_L і D_P, оскільки вони втрачають і відкидають однакову кількість пакетів. Крім того, результати продуктивності T для RED, NLRED і RED_E подібні незалежно від мінімального порогового значення та стабілізовані на значенні β, і на них не впливає мінімальний пороговий параметр. Нарешті, результати D_P для RED, NLRED і RED_E зменшується, якщо значення мінімального порогу збільшується. Можна зробити висновок, що на результати показників ефективності RED, NLRED і RED_E впливає мінімальний пороговий параметр, за винятком результатів T.

ВИСНОВКИ

В кваліфікаційній роботі було проаналізовано роботу різних алгоритмів AQM контролю перевантаження з недоліками та їхніми перевагами. Після аналізу та порівняння кількох алгоритмів AQM було виявлено, що жоден алгоритм не може вирішити всі проблеми зменшення перевантаження в мережі. Запропоновано метод запобігання перевантаження в чергах маршрутизаторів, який відрізняється від алгоритму RED та його наступників.

Імітаційне моделювання було проведено для визначення переваг і недоліків RED_E. Результати моделювання були отримані при використанні різних значень ймовірності надходження пакетів для вимірювання продуктивності RED, NLRED і RED_E в різних ситуаціях (без перевантажень, незначне перевантаження та сильне перевантаження).

Проведене імітаційне моделювання показало, що застосування експоненціального процесу відкидання пакетів з перевантаженої черги зводить до мінімуму залежність від попередньо налаштованих параметрів алгоритму RED та сприяє зростанню пропускної здатності мереж.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. J. Wang, L. Guan, L. B. Lim et al., “QoS enhancements and performance analysis for delay sensitive applications,” Journal of Computer and System Sciences, vol. 77, no. 4, pp. 665–676, 2011.

2. M. Welzl, Network Congestion Control: Managing Internet Traffic, p. 282, Wiley, Hoboken, NJ, USA, 2005.

3. D. D. Clark and W. Wenjia Fang, “Explicit allocation of best- effort packet delivery service,” IEEE/ACM Transactions on Networking, vol. 6, no. 4, pp. 362–373, 1998.

4. S. Athuraliya, S. H. Low, V. H. Li, and Q. Qinghe Yin, “REM: active queue management,” IEEE Network, vol. 15, no. 3, pp. 48–53, 2001.

5. J. Aweya, M. Ouellette, and D. Y. Montuno, “A control TheoRED_Eic approach to active queue management,” Computer Network, vol. 36, no. 2-3, pp. 203–235, 2001.

6. W. Feng, D. Kandlur, D. Saha, and K. G. Shin, “Blue: a new class of active queue management algorithms,” Technical Report, UM CSE-TR-387-99, University of Michigan, Ann Arbor, MI, USA, 1999.

7. S. Floyd and V. Jacobson, “Random early detection gateways for congestion avoidance,” IEEE/ACM Transactions on Net-working, vol. 1, no. 4, pp. 397–413, August 1993.

8. S. Floyd, G. Ramakrishnan, and S. Shenker, “Adaptive RED: an algorithm for increasing the robustness of RED’s active queue management,” Technical Report, ICSI, New Delhi, India, 2001.

9. S. Floyd, “Recommendations on using the gentle variant of RED,” 2000, http://www.aciri.org/floyd/red/gentle.html.

10. B. Abbasov and S. Korukoglu, “Effective RED: an algorithm to improve RED’s performance by reducing packet loss rate,” Journal of Network and Computer Applications, vol. 32, no. 3, pp. 703–709, 2009.

11. W. Chen, Y. Li, and S. H. Yang, “An average queue weight parameterization in a network supporting TCP flows with RED,” in Proceedings of the 2007 IEEE International Con- ference on TuesM02 Networking, Sensing and Control, pp. 590–595, London, UK, April 2007.

12. W. Chen and S. H. Yang, “The mechanism of adapting RED parameters to TCP traffic,” Proc. of Computer Communica-tions, Elsevier, vol. 32, no. 13-14, pp. 1525–1530, 2009.

13. J. Hong, C. Joo, and S. Bahk, “Active queue management algorithm considering queue and load states,” in Proceedings of the 13th International Conference on Computer Commu-nications and Networks, pp. 140–145, Chicago, IL, USA, October 2004.

14. K. Okokpujie, C. Emmanuel, O. Shobayo, E. Noma-Osaghae, and I. Okokpujie, “Comparative analysis of the performance of various active queue management techniques to varying wireless network conditions,” International Journal of Elec- trical and Computer Engineering (IJECE), vol. 9, no. 1, p. 359-368, 2019.

15. H. Abdel-Jaber, M. Woodward, F. Thabtah, and A. Abu-Ali, “Performance evaluation for DRED discrete-time queueing network analytical model,” Journal of Network and Computer Applications, vol. 31, no. 4, pp. 750–770, 2008.

16. В.В. Філіппов., О.А. Харченко, С.О. Партика, О.А. Янковський, “Черги у маршрутизаторах та aqm алгоритми”, Десята міжнародна науково-технічна конференція «Проблеми інформатизації». – Черкаси-Баку-Бельсько-Бяла-Харків. 24 – 25 листопада 2022 р. – С. 37.