Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

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

Кафедра електронних обчислювальних машин
(повна назва)

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Програмно-алгоритмічні методи оптимізації трафіку комп'ютерних мереж

(тема)

Виконав:
студент II курсу, групи СПзм-18-2
Архирєєв Р.С.
(прізвище, ініціали)

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

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

Освітня програма
Системне програмування
(повна назва освітньої програми)

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

Допускається до захисту

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

2020 р.

Харківський національний університет радіоелектроніки

Факультет навчально-науковий центр заочної форми навчання

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

Освітня програма Системне програмування
(повна назва)

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

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Архирєєву Руслану Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Програмно-алгоритмічні методи оптимізації трафіку комп'ютерних мереж

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

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

3. Вхідні дані до роботи 1) моделі та сучасні методи управління потоками інформації
2) стек протоколів TCP/IP
3) алгоритми AQM
4) програмні засоби: Windows 10, OpNet 14.5, NS-2

4. Перелік питань, що потрібно опрацювати в роботі
1) аналіз стану проблеми
2) огляд технологій управління інформаційними потоками комп'ютерних мереж
2) вибір та обґрунтування методики та засобів дослідження
3) вибір інструментальних засобів для моделювання
4) проведення експериментальних досліджень
5) висновки

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (слайдів) Плакати –20 арк. ф. А4

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Огляд протоколів корпоративних комп'ютер-	31.03.20-10.04.20	
	них мереж		
2	Огляд технологій управління інформаційними	11.04.20-16.04.20	
	потоками		
3	Вибір та обґрунтування методики дослідження	17.04.20-21.04.20	
4	Розробка моделей та алгоритмів управління	22.04.20-29.04.20	
5	Вибір інструментальних засобів	30.04.20-04.05.20	
6	Проведення експериментів	05.05.20-07.05.20	
7	Оформлення матеріалів атестаційної роботи	08.05.20-11.05.20	
8	Подання атестаційної роботи керівникові та	12.05.20-13.05.20	
	її попередній захист		
9	Подання атестаційної роботи на рецензування	14.05.20-15.05.20	

Дата видачі завдання 30 березня 2020 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка атестаційної роботи: 105 с., 42 рис., 2 табл., 1 дод., 15 джерел.

AQM, АЛГОРИТМ, МАРШРУТИЗАТОР, МЕРЕЖА, МЕРЕЖЕВИЙ РІВЕНЬ, МОДЕЛЮВАННЯ, ПРОТОКОЛ, ТРАНСПОРТИЙ РІВЕНЬ.

Метою атестаційної роботи є розробка моделей, алгоритмів та методів управління інформаційними потоками комп’ютерних мереж.

У ході виконання атестаційної роботи проведено огляд теоретичних та практичних аспектів функціонування протоколів транспортного та мережевого рівнів моделі OSI, зроблено аналіз сучасних інструментальних засобів оцінювання ефективності існуючих мережевих протоколів, запропоновано метод управління чергами маршрутизаторів та проведено моделювання роботи алгоритмів AQM при різних мережевих параметрах.

ABSTRACT

Master’s thesis: 105 pages, 42 figures, 2 tables, 1 appendices, 15 sources.

AQM, ALGORITHM, ROUTER, NETWORK, NETWORK LEVEL, SIMULATION, PROTOCOL, TRANSPORT LEVEL.

The purpose of this attestation work is to develop models, algorithms and methods for managing information flows of computer networks.

During the attestation work the review of theoretical and practical aspects of transport protocols and network layer model of OSI, the analysis of modern tools of evaluating the effectiveness of existing network protocols is made, the method of router queue management is offered and modeling of AQM algorithms operation at different network parameters is carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

AQM – активне управління чергою (англ., Active Queue Management)

ARED – адаптивне раннє випадкове виявлення (англ., Adaptive Random Early Detection)

AVQ – адаптивна віртуальна черга (англ., Adaptive Virtual Queue)

DRED – динамічне раннє випадкове виявлення (англ., Adaptive Random Early Detection)

ECN – явне повідомлення про перевантаження (англ., Explicit Congestion Notification)

FIFO – перший прийшов перший вийшов (англ., First In First Out)

FRED – потоковий RED (англ., Flow RED)

GRED – пологий RED (англ., Gentle RED)

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

MQL – середня довжина черги (англ., Mean Queue Length)

NS2 – мережевий симулятор (англ., Network Simulator)

QoS – якість обслуговування (англ., Quality of Service)

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

REM – випадкове раннє маркування (англ., Random Early Marking)

RTT – час кругового обертання (англ., Round-Trip Time)

SRED – стабілізований RED (англ., Stabilized RED)

ВСТУП

Широкий розвиток сучасних телекомунікаційних технологій, а також зниження витрат на передачу та обробку інформації викликає постійне зростання обсягу передаваного мережевого трафіку. Так, щорічний звіт компанії Sandvine демонструє загальне середнє збільшення агрегованого трафіку Північної Америки більш ніж на 30-40% щорічно. На 2019 рік він становив 121,4 Гб в місяць на одного абонента. При цьому джерелом більш ніж 60% трафіку є мультимедіа програми потокового аудіо та відео. Як звісно, трафік подібного типу найбільш чутливий до різних затримок у передачі пакетів. Наприклад, при передачі голосових даних затримка більше 10 мс стає критичною, для потокового відео вона не повинна перевищувати більше 100 мс. Таким чином, необхідні підвищені вимоги до комп'ютерних мереж передачі даних, до алгоритмівм управління трафіком та до застосованих методів боротьби з перевантаженнями мереж.

Проблеми управління трафіком стають одними з ключових при забезпеченні показників якості обслуговування (QoS) абонентів мережі. Як звісно, управління трафіком повинно включати в себе мережеве планування та оптимізацію. Планування мереж являє собою процес визначення топології мережі, а також пропускної здатності каналів мережі з урахуванням передбачуваного наперед навантаження. Оптимізація являє собою управління розподілом існуючого трафіку в мережі.

Як відомо, протокол IP для початку обміну даними не потребує попередньої установки з'єднання, тому мережевий пристрій не має змоги визначити необхідний обсяг мережевих ресурсів до отримання найпершого повідомлення, що іноді приводить до того, що вхідний потік займає майже всі ресурси маршрутизатора. Насамперед це пов'язано з використанням існуючих буферів кінцевої довжини в комутуючому устаткуванні. Під час заповнення таких буферів нові поступаючі пакети починають відкидатися в

вузлах мережі, що завжди викликає повторну передачу даних, і таким чином навантаження може зростати лавиноподібно. Основна причина такої ситуації полягає в тому, що протоколи на транспортному рівні в моделі OSI, наприклад UDP та TCP, не мають прямих засобів для визначення стану мережі, вони роблять висновки завдяки діалогу між кінцевими системами. Отримувач в рамках протоколу TCP відсилає джерелу підтвердження про прийом та обсяг інформації, який він готовий прийняти. Тому, вихідні дані для алгоритмів управління каналом мережі стають дуже грубим методом.

Управління ресурсами в комп’ютерній мережі утруднено, в першу чергу, тому, що за своєю природою майже завжди ця мережа є децентралізованою і, як правило, складається з безлічі різних автономних систем з локалізованим управлінням.

Для різнорідної мережі завдання управління наявними ресурсами доцільно вирішувати спільно "із кінця-в-кінець» та локально в узлах (в рамках кожного мережевого вузла). Визначено два базових класи різноманітних механізмів управління навантаженням каналів в мережі Інтернет: ті, що базуються в хостах (host-based) та ті, що базуються в маршрутизаторах (router-based).

1 АНАЛІЗ ПРОБЛЕМАТИКИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Необхідність управління трафіком

Одним з головних завдань при забезпеченні необхідної якості обслуговування в мережах передачі інформації є управління трафіком. Під поняттям управління трафіком розуміється сукупність алгоритмічних засобів, які реалізовані апаратно, або програмно, спрямовані на надання чіткого функціонування мережі з потрібною якістю обслуговування в умовах ефективного використанням існуючих ресурсів.

Як і в будь-якій комп’ютерній мережі, що забезпечує необхідну якість обслуговування, для реалізації QoS необхідно застосовувати певні функції «управління допустимостью з’єднання» CAC (Connection Admission Control). Такі функції являють собою набор дій, які повинні здійснюватися мережею під час встановлення з’єднання з метою визначити, можливе або ні забезпечення підтримки необхідних параметрів іта якості обслуговування. Таким чином, нове з’єднання може бути виконано мережею тільки тоді, коли маються в наявності необхідні ресурси, показники щодо якості обслуговування вже існуючих з’єднань в мережі виконуються, а встановлюване з’єднання їм не нашкодить. Дуже важливою частиною цього процесу обробки пакетів для забезпечення QoS в мережі є різноманітні алгоритми управління чергами в мережевому обладнанні, а саме маршрутизаторах.

Поняття алгоритма управління чергою включає до себе набір методів, які забезпечують надходження, зберігання або передачу пакетів, що надходять в систему для обслуговування. Вибір такого алгоритму управління чергамии доволі складний, тому що кожен вид алгоритмів має як переваги, так і недоліки.

Завдяки тому, що трафік і, відповідно, якість обслуговування трафіку є різнорідними, повинна бути наявність різних пріоритетів для його обслуговування. Пакети з різними показниками пріоритетів в мережевих маршрутизаторах повинні розміщуватися по різним чергам, завдяки чьому проблема розподілу ресурсів центрального процесора роутера стає досить актуальною.

Досвід розробки та експлуатації мереж передачі інформації показав, що системи з динамічним управлінням трафіком є обов'язковим елементом сучасного мережевого обладнання. Під терміном «управлінням потоками» завжди розуміється сукупність різноманітних механізмів, які надають доступ до мережевих ресурсів та забезпечують баланс параметрів мережі і трафіку, призначеного для абонента. Завданням любої системи управління потоками насамперед є запобігання перевантажень або зниження їх ймовірностей та вихід із тупикових ситуацій. Така система управління трафіком повинна функціонувати в режимі, коли забезпеченується принцип «справедливого розподілу ресурсів», а також гарантовані запитані показники якості для обслуговування користувачів.

Під перевантаженням мережі розуміється такий стан, під час якого основні показники QoS істотно погіршуються. Таким чином, в залежності від типу комп'ютерної мережі ці погіршення можуть викликати збільшення числа втрачених пакетів даних, або збільшення середніх значень затримки і джиттера. Перевантаження черг можуть виникати як на малих ділянках мережі (так звані, локальні перевантаження), так і мати тенденцію поширення на великі ділянки мережі (так звані, глобальні перевантаження).

В сучасній мережі Інтернет перевантаження викликає різкий зріст розмірів черг в деяких вузлах. У випадку, коли механізм контролю або управління перевантаженням відсутній, чи він не справляється з потоком покладених на нього завдань, це завжди приводить до неможливості доступу деяких користувачів до окремих ресурсів мережі, а в деяких випадках і до всіх ресурсів мережі. Такий стан називають як блокування мережі,

відповідно для визначення такого блокування в мережі Інтернет використовується спеціальний термін Internet-meltdown.

Продуктивність такої мережі (число переданих пакетів) знижається до нуля, а мережеві затримки зростають до нескінченності. Також стан перевантаження в мережі можна визначити як деякий стан мережевих вузлів, коли сама мережа не гарантує запрошену якість обслуговування для раніш встановлених з'єднань абонентів, так само як і для встановлюваних з'єднань.

Таким чином, перевантаження може бути викликано різноманітними флуктуаціями потоків даних або відмовленням деякого мережевого обладнання, що приводить до неможливості мережі надати необхідну якість обслуговування вже існуючих з'єднань, а так само і неможливістю встановлення інших нових з'єднаннь з запитаним QoS.

Ще в 1980 році в роботі [1] було доведено факт, що при наявності умов перевантаження загальна продуктивність мережі може мати меньші значення ніж мережа, в якій застосовуються механізми для динамічного управління розподілом ресурсів.

На рисунку 1.1 представлено залежності інтенсивностей вихідного мережевого трафіку та часу затримки в мережі від інтенсивності вхідного потоку даних, де P_{max} – максимальна пропускна спроможність каналу, $P_{вх}$ – інтенсивність вхідного трафіку, а $P_{вих}$ – інтенсивність вихідного трафіку, а $T_з$ – час затримки. Приведені графіки демонструють якісний характер щодо реакції продуктивності мережі та затримки на поточну зміну вхідного навантаження. Так, графік за номером 1 показує ідеальну поведінку комп'ютерної мережі під час збільшення навантаження, у той час, коли на виході доступна максимальна пропускна здатність.

Постійно зростаючий інтерес науковців в області комп'ютерних мереж до різноманітних механізмів боротьби з виникненням перевантажень пояснюється, не в останню чергу, тим, що, як розглянуто в [2], архітектура самої мережі, принципи її роботи та більшість характеристик визначаються різноманітними механізмами на фізичному, канальному, мережевому та

транспортному рівнях. Такі механізми мають дуже істотний вплив на застосування ресурсів самої мережі, а також дуже тісно пов’язані з процесом контроля потоків даних і обмеженням саме навантаження. Таким чином, любі схеми контролю, що створені для одного типу інформаційних мереж, не завжди ефективні в мережах другого типу.

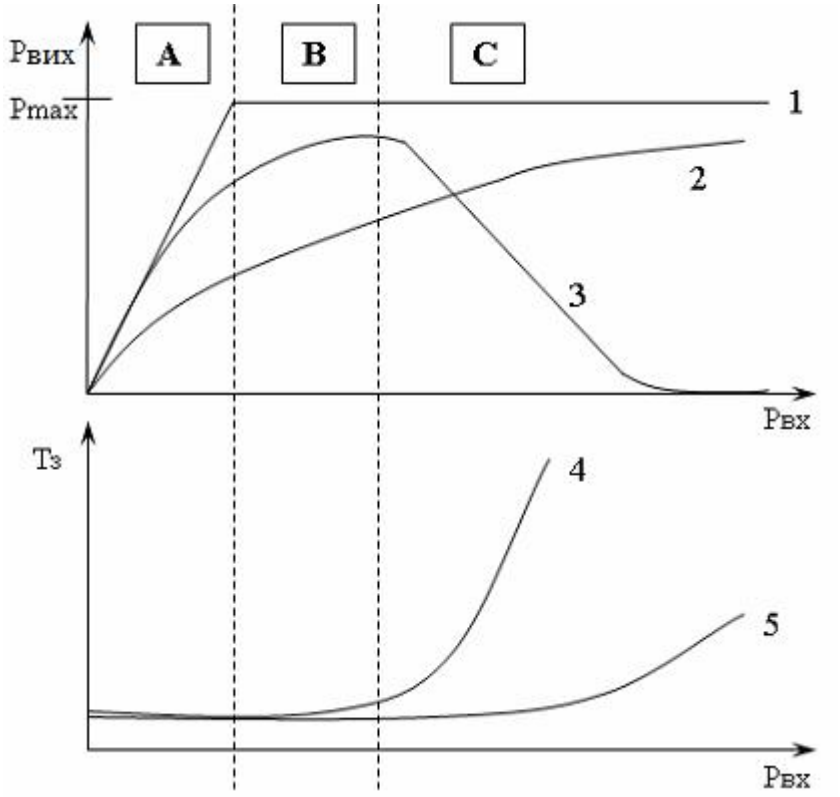

Рисунок 1.1 – Залежність інтенсивностей вихідного трафіку та часу затримки від інтенсивності вхідного трафіку

Графіки 2 та 4 відповідають випадку відсутності механізмів управління перевантаженнями, а графіки 3 та 5 – відповідають випадку з наявністю механізмів управління мережевими перевантаженнями і наявності управління потоком. На рисунку відрізок А відповідає низькому навантаженню в мережі, відрізок В називається зоною керованого перевантаження, коли мережау ще може справляється з існуючим навантаженням, хоча при цьму збільшується затримка.

При відсутності механізмів управління перевантаженнями на відрізку С

відбувається переповнення приймальних буферів приймачів, результатом чого є відкидання пакетів і спроби їх повторної передачі. В результаті затримки починають різко збільшуватися, а продуктивність починає прагнути до нуля.

Також необхідно виділити окремо фактор впливу поточного профілю навантаження на загальну якість функціонування механізмів. Застосування додатків в комунікаційних мережах, наприклад, передачі VoIP, відео, або іншого мультимедійного трафіку, визначає зовсім нові вимоги до застосованої методики контролю та управління потоками даних. Завжди кожний новий додаток потребує свої власні вимоги до наданих показників якості обслуговування трафіку. Отже, саме ці чинники і викликають необхідність дослідження нових методів та нових механізмів управління потоками трафіку і боротьби з перевантаженнями взагалі.

В сучасних мережах протоколи для передачі даних застосовують принціп «з-кінця-в-кінець» на транспортному рівні OSI, тобто вони реалізовані тільки в самих хостах. Тому у маршрутизаторів майже відсутня якась можливість впливати на генероване хостами навантаження. Також, з іншого боку, любе джерело трафіку повинно мати можливості для отримання якимось чином інформації про виникнення перевантаження в деякому транзитному вузлі мережі.

Оскільки маршрутизатор не має в собі реалізації транспортного рівня, іншими словами він не може модифікувати інформаційний потік, то єдиним розумним рішенням у такій ситуації може бути посилка джерелу спеціальних службових пакетів з попередженням про факт аба загозу виникнення перевантаження. Таким чином, джерелом такойї інформації є сама мережа, а її приймачем – саме джерело навантаження.

Цей механізм називають «зворотнім зв’язком» та відносять до так званого класу router-based. Така реалізація двох механізмів управління перевантаженням приносить досить хороші результати.

У випадку отримання джерелом від маршрутизатора інформації про

виникнення перевантаження, його механізм управління навантаженням, що реалізовано в самому хості, повинен зменьшити швидкість своєї передачі пакетів в мережу.

При відсутності механізму «зворотного зв’язку» джерело не спроможне приймати якесь рішення про зміну своєї швидкості передачі. За цим принципом джерело пакетів може отримувати інформацію по «зворотному зв’язку» або прямо від мережі, або від хоста-приймача.

1.2 Управління перевантаженням в протоколі TCP

На даний час існує цілий ряд різноманітних реалізацій методів управління перевантаженнями в транспортному протоколі TCP, що розрізняються в пристосованності до рішень різних завдань. За фактом, першою реалізацією такого алгоритма керування перевантаженнями в протоколі TCP була реалізація Tahoe, яку було запропоновано Якобсоном в 1988 році. Вже у 1989 році були визначені основні вимоги до реалізації механізмів роботи рівнів OSI: канального, мережевого та транспортного, а також алгоритм управління перевантаженнями в протоколі TCP. В наш час більшістю ОС використовується варіант TCP Reno, який було запропоновано Якобсоном вже в 1990 році, і який відрізняється від TCP Tahoe більше вдосконаленим алгоритмом "швидкої повторної передачі", а також додаванням алгоритму "швидкого відновлення".

В протоколі існують два параметра, які використовуються в алгоритмах управління перевантаженням в TCP/IP, реалізовані в протоколі TCP:

- rwnd (receiver window) – вікно приймача або кількість даних, що приймач може прийняти поспіль;

- cwnd (congestion window) – вікно перевантаження або вікно, що обмежує кількість даних, яке джерело має змогу послати приймачу без отримання підтвердження доставки.

Параметр rwnd розміщується в полі заголовка сегмента TCP та має

сигналізувати про те, скільки місця ще вільно на приймальній стороні. З його допомогою можливе управління швидкістю передачі даних на стороні джерела. Застосовуючи цей параметр в TCP реалізована саме функція управління потоком.

Так, за допомогою зміни cwnd вікна кількість переданих даних від джерела може змінюватися. Саме на параметрі cwnd зоснована функція управління перевантаженнями в TCP. Таким чином, середню пропускну спроможність з’єднання TCP можна визначити за формулою:

$$BW=cwnd \cdot MSS/RTT,$$

де BW – пропускна здатність каналу мережі, MSS – максимальний розмір сегменту, RTT – час кругового оберту сегмента. Таким чином, за допомогою управління cwnd вікном можна впливати на поточну пропускну здатність з’єднання.

За допомогою вікна cwnd приймач також має змогу управляти швидкістю передачі даних джерелом, наприклад, завдяки відкиданню одного з прийнятих сегментів. Це повинно зменшити вікно перевантаження, і тому джерело буде примусене посилати менше пакетів без підтвердження. Таким чином працюють деякі з алгоритмів управління трафіком.

Механізми управління трафіком в протоколі TCP складаються з декількох алгоритмів: аддитивноого збільшення – мультиплікативного зниження, алгоритму повільного старту, алгоритму запобігання перевантаженням, алгоритму швидкої повторної передачі та швидкого відновлення.

Слід відзначити, що всі існуючі у даний час реалізації протоколу TCP призначені для гарантованої доставки даних без суттевих обмежень на параметри мережевої затримки і ймовірності втрати пакетів. Протокол не має засобів безпосереднього визначення поточного стану мережі. Тому, сучасний механізм управління трафіком в протоколі TCP не має можливості ефективно

управляти перевантаженням, а тому не забезпечує в повній мірі необхідну якість обслуговування для випадків високої інтенсивності вхідного трафіку.

1.2.1 Алгоритм повільного старту

Алгоритм повільного старту (SS) був запропонований для контролю перевантажень в Інтернеті, і цей метод починає працювати, коли відбувається одна з наступних подій:

- коли встановлено нове tcp-з’єднання;
- коли з’єднання tcp становиться активним після періоду очікування;
- коли з’єднання tcp відновлюється після закінчення тайм-ауту.

Коли алгоритм повільного старту запускається внаслідок однієї з вищезазначених подій, джерело ініціалізує своє cwnd до розміру одного сегмента TCP. Об’єм cwnd відповідає кількості даних, які можуть бути надіслані від джерела до адресата по всій мережі до наявності перевантаженості. Причиною встановлення cwnd в однин сегменті TCP є запобігання швидкому нарощуванню черг в буферах маршрутизатора [3-5]. Алгоритм повільного старту починає надсилати сегмент TCP до місця призначення, потім чекає, поки він отримає підтвердження від пункту призначення. Після того, як джерело отримує підтвердження, воно збільшує свій розмір cwnd до двох сегментів, що дозволяє йому надсилати два сегменти даних TCP. Знову ж таки, після того, як джерело отримає підтвердження того, що обидва сегменти TCP успішно отримані пунктом призначення, він збільшує своє cwnd до чотирьох сегментів тощо. Слід зазначити, що джерело збільшує розмір свого вікна cwnd експоненціально (рисунок 1.2). На рисунку 1.3 вказано, як реалізується такий алгоритм.

На рисунку 1.3 показано, що джерело збільшує розмір свого вікна з тією ж кількістю підтверджень, які отримує від місця призначення протягом часу одного оберту (RTT), якщо кожен сегмент TCP був підтверджений окремо. Джерело залишається в режимі SS, поки або розмір cwnd не стане

рівним вікну, оголошеним пунктом призначення, або в мережі виникає перевантаженість. У цьому випадку джерело припиняє роботу алгоритмму SS та вступає на другу стадію, що називається алгоритмом запобігання перевантаженням.

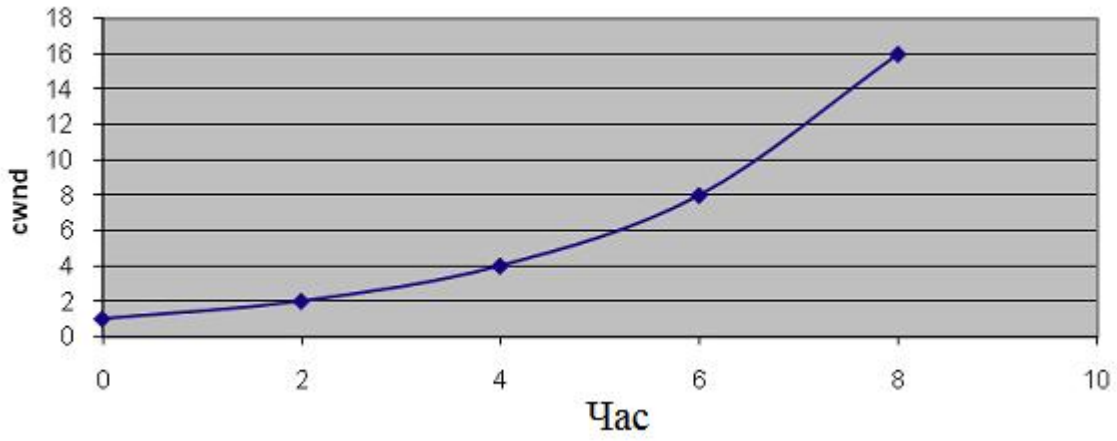

Рисунок 1.2 – Експоненціальне збільшення cwnd в алгоритмі повільного старту

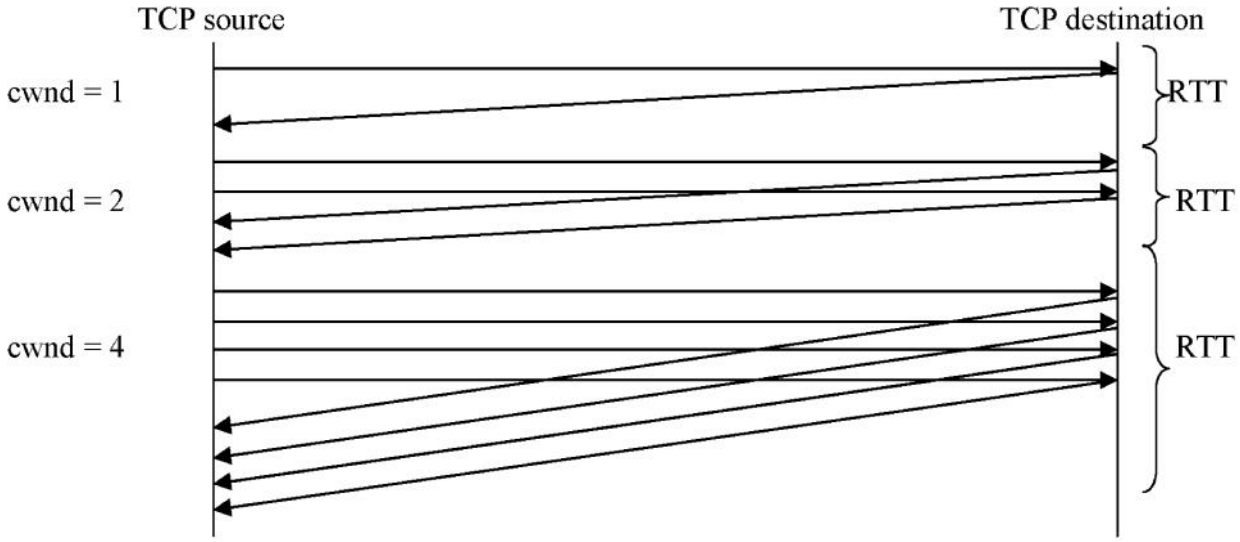

Рисунок 1.3 – Реалізація алгоритму повільного старту

1.2.2 Алгоритм запобігання перевантаженням

Алгоритм запобігання перевантаженням (CA) [6] був запропонований у 1988 році Ван Джейкобсоном для зменшення заторів пакетів в Інтернеті. Алгоритм CA ініціалізується при будь-якому перевантаженні або коли розмір вихідного cwnd перевищує розмір оголошений пунктом призначення.

Алгоритм CA покладається на два параметри: cwnd та поріг повільного старту (ssthreshold). Коли мережа стає перевантаженою, джерело зменшує розмір свого cwnd, встановлюючи параметр ssthreshold в половину значення величини cwnd. Після зменшення розміру свого cwnd джерело переходить до алгоритму CA, в якому кожного разу, коли джерело успішно отримує підтвердження, воно збільшує розмір cwnd лінійно на один сегмент TCP. Крім того, джерело збільшує розмір cwnd на один сегмент виключно для кожного RTT незалежно від кількості підтверджень, які воно отримує під час кожного RTT.

Існує два способи виявлення перевантаженості джерелом TCP:

- коли джерело отримує подвійні підтвердження;
- коли перевищується тайм-аут повторної передачі (закінчується період очікування повторної передачі).

Випадок, коли перевантаженість має місце через отримання повторних підтверджень джерелом представлено на рисунку 2.4. На рисунку 2.4 можна побачити, що джерело надсилає кілька сегментів TCP до місця призначення, та кожен сегмент TCP має розмір 600 байт.

Джерело передає перший сегмент TCP розміром 600 байт і чекає підтвердження від пункту призначення, чи успішно він отримав цей сегмент. Це робиться за допомогою підтвердження номером 601, і всі сегменти даних TCP з діапазоном даних нижче цього підтвердження (менше 601) вже успішно отримані пунктом призначення. Слід зазначити, що підтвердження з номером 601 означає, що наступний очікуваний сегмент TCP у пункті

призначення починається з 601 і всі сегменти TCP до 600 успішно отримані адресатом TCP.

На рисунку 1.4 також проілюстровано, що джерело посилає сегмент TCP, який починається з 1201, а не 601. Коли пункт призначення отримує цей сегмент, він враховує 601 сегмент, як загублений та генерує ще одне підтвердження із вмістом 601 (подвійне підтвердження) до його джерела. Надалі джерело або вирішує, що сегмент TCP з номером 601 пошкоджений або його втрачено через виникнення перевантаженості в мережі. Таким чином, джерело чекає довше і продовжує надсилати сегменти TCP до місця призначення.

Тепер, якщо джерело TCP отримує третій дублікат підтвердження (підтвердження з номером 601), він оголошує, що сегмент TCP втрачений, і повторно відправляє загублений сегмент TCP, обов’язково зменшуючи своє порогове значення cwnd до половини, для того, щоб контролювати перевантаженість. В іншому випадку джерело вирішує, що сегмент TCP пошкоджено, і повторно відправляє правильний сегмент TCP, не зменшуючи cwnd.

На рисунку 1.4 продемонстровано, що сегмент TCP з номером 601 пошкоджений, а сегмент TCP з номером 1801 – втрачений сегмент. Сегмент TCP з номером 1801 було визнано відправником як втрачений сегмент, оскільки відправник отримав три дублікати підтвердження номера 1801 сегмента TCP від місця призначення. Ці підтвердження означають, що сегмент TCP з номером 1801 не було доставлено до місця призначення. Після отримання третього дублюючого підтвердження джерелом TCP, джерело приймає рішення, що сегмент з номером 1801 втрачено через перевантаженість. Таким чином, основна мета дублікатів підтверджень – визначення відправником, чи певний сегмент TCP пошкоджений, чи його втрачено [5].

Другий спосіб виявлення втрачених пакетів – закінчення тайм-ауту повторної передачі, коли кожен раз, коли джерело надсилає сегмент TCP,

воно ініціалізує таймер і чекає, поки воно отримає підтвердження про те, що відправлений сегмент TCP доставлено до місця призначення. Пункт призначення продовжує генерувати підтвердження до тих пір, поки сегменти TCP надходять до нього від джерела, навіть якщо ці сегменти пошкоджені. Наприклад, якщо останній сегмент TCP загублено до досягнення пункту призначення, пункт призначення не зможе генерувати повторне підтвердження, оскільки джерело не може надсилати більше сегментів TCP, і це призводить до кінця тайм-ауту повторної передачі на стороні джерела. В результаті цього, джерело переходить у режим SS після скидання cwnd до розміру одного сегмента TCP.

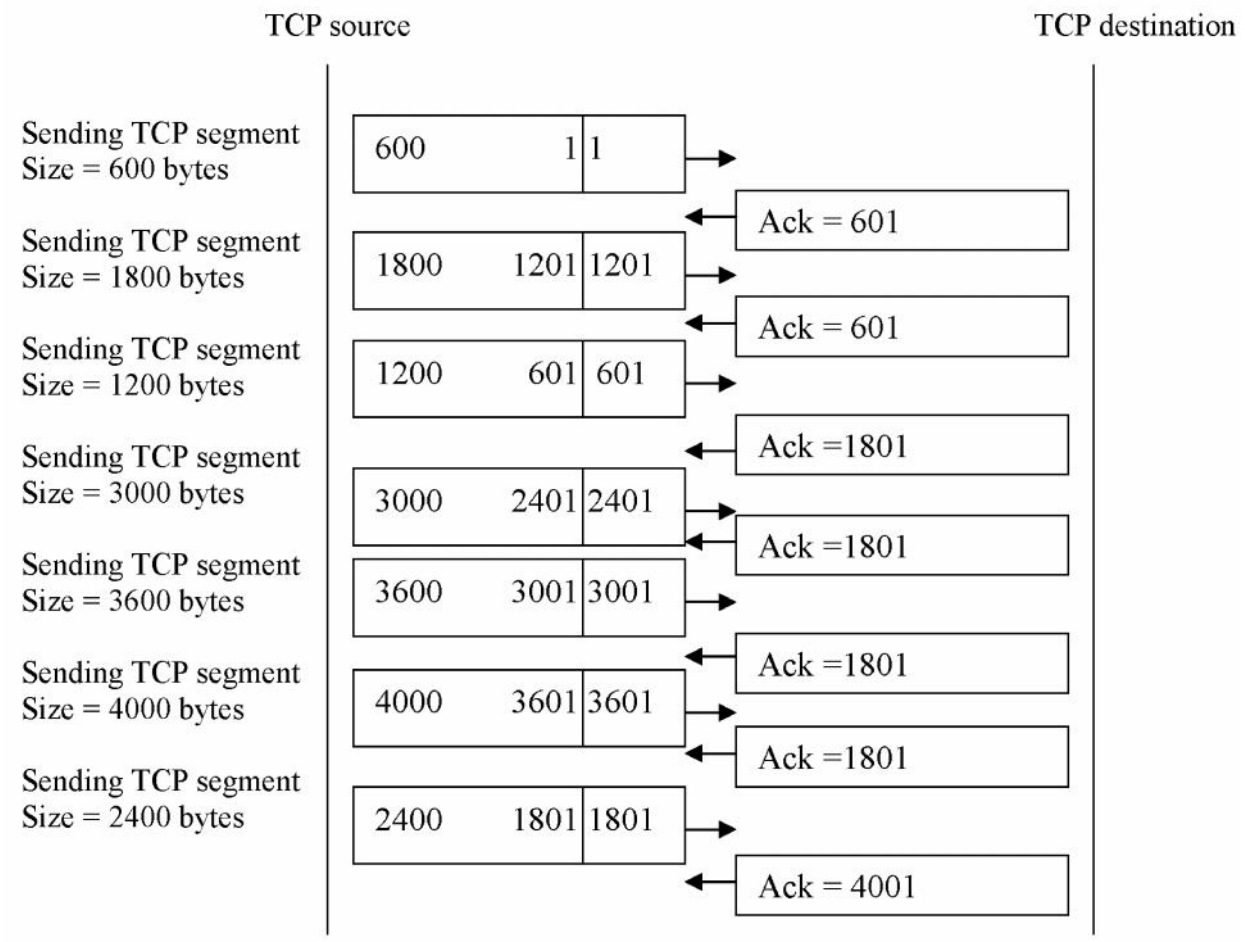

Рисунок 1.4 – Отримання подвійних підтверджень джерелом TCP сегментів

Перехід з режиму CA в режим SS відбувається, коли відбувається одна з наступних двох подій, перша подія – це закінчення тайм-ауту повторної передачі, а друга подія – виникнення перевантаженості, що спричиняє зменшення розміру cwnd.

На рисунку 1.5 показано, що коли розмір cwnd дорівнює або менший, ніж розмір ssthreshold, джерело знаходиться у режимі SS, інакше воно знаходиться у режимі CA.

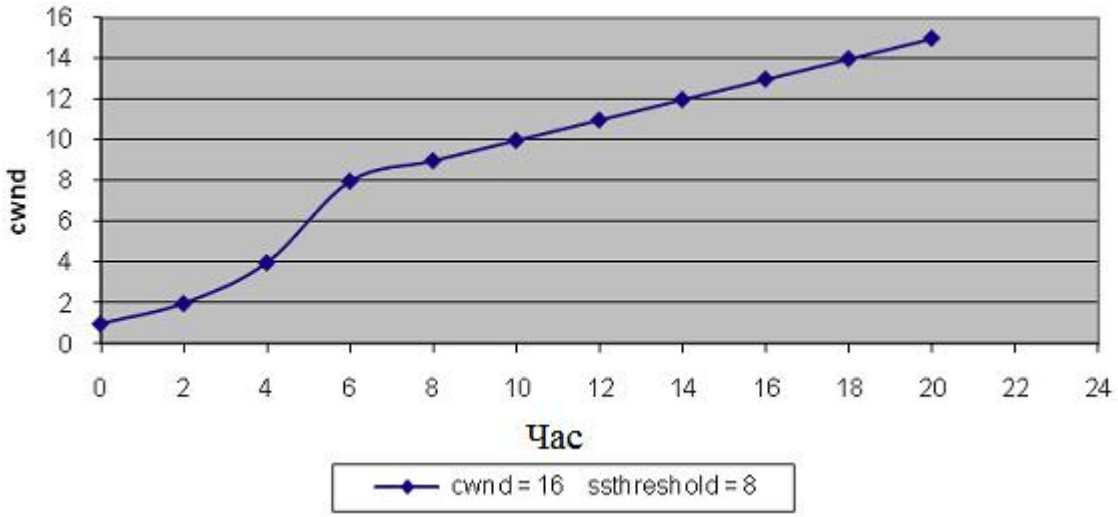

Рисунок 1.5 – Віконання джерелом TCP алгоритмів SS та CA

У випадках, коли сегмент втрачено, алгоритм FR починає працювати, надсилаючи втрачений сегмент до пункту призначення після зменшення розміру cwnd до одного сегмента через початок роботи алгоритму SS. Якщо встановити cwnd на один сегмент, це погіршить пропускну здатність TCP.

Щоб вирішити цю проблему в [7] запропоновано алгоритм управління перевантаженням TCP для підтримки продуктивності мережі при втраті сегментів.

З іншого боку, якщо сегмент пошкоджено, тоді не застосовується зменшення кількості сегментів, і тому продуктивність мережі буде підтримуватися. Алгоритм FR має ряд переваг, таких як:

- підтримиється пропускна здатністьі, коли джерело підтримує розмір свого cwnd, тобто отримання копій з тих же сегментів не зменшить обертів джерела TCP, тому продуктивність пропускної здатності буде підтримуватися;

- коли втрачаються сегменти TCP, джерело переходить у режим CA, а не в режим SS і не чекає появи тайм-ауту повторної передачі, замість цього воно повторно передає втрачені сегменти TCP, як тільки отримує третє дублікатне підтвердження від отримувача.

1.2.3 Алгоритм швидкого відновлення

Алгоритм швидкого відновлення (FRec) був запропонований Ван Джейкобсоном у 1990 році [7], щоб підвищити продуктивність комп’ютерних мереж, а також для управління перевантаженнями. Отримання повторних підтверджень джерелом не означає втрату сегмента TCP, оскільки цей сегмент може бути пошкодженим. Це пояснює очікування джерела, для того, щоб перевірити, чи отримає воно третє повторне підтвердження, щоб вирішити, що сегмент TCP втрачено. Крім того, кожен раз, коли адресат отримує сегмент поза замовленням, він генерує подвійне підтвердження до джерела; коли джерело отримує третє повторне підтвердження, воно вирішує про наявність перевантаженості та виконує метод FRec наступним чином:

- встановлюється значення ssthreshold на половину розміру cwnd, але не менше двох сегментів шляхом виконання алгоритму ca;

- вікликається повторна передача втраченого сегмента tcp, викликаючи алгоритм fr;

- встановлюється cwnd на ssthreshold плюс три сегменти tcp, які спричинили три подвійні підтвердження.

У випадках, коли джерело отримує менше трьох дублікатів підтвердження, джерело збільшує своє cwnd на сегментах TCP, які були підтверджені подвійними підтвердженнями. Нарешті, якщо джерело отримує

очікуване підтвердження після отримання третього дубліката підтвердження від пункту призначення, воно встановлює значення cwnd в значення ssthreshold. Це очікуване підтвердження визнає, що всі сегменти отримані пунктом призначення TCP.

Алгоритм FRec підтримує продуктивність, коли відбуваються перевантаження, воконуючи алгоритм CA, а не алгоритм SS, і це не дозволяє скоротити розмір cwnd на один сегмент. Тому багато сегментів, що стоять у чергах на маршрутизаторах, не відкидаються, що призводить до суттевого підвищення продуктивності.

Після введення джерела TCP в алгоритм CA воно повторно передає втрачені сегменти до своїх пунктів призначення, використовуючи алгоритм FR. В основному, реалізація алгоритму FRec включає реалізацію алгоритму CA з наступним алгоритмом FR. Метод FRec покращує продуктивність TCP, коли з певного джерела втрачається один сегмент TCP. На рисунку 1.6 представлено приклад реалізації всіх алгоритмів контролю перевантажень TCP (SS, CA, FR, FRec).

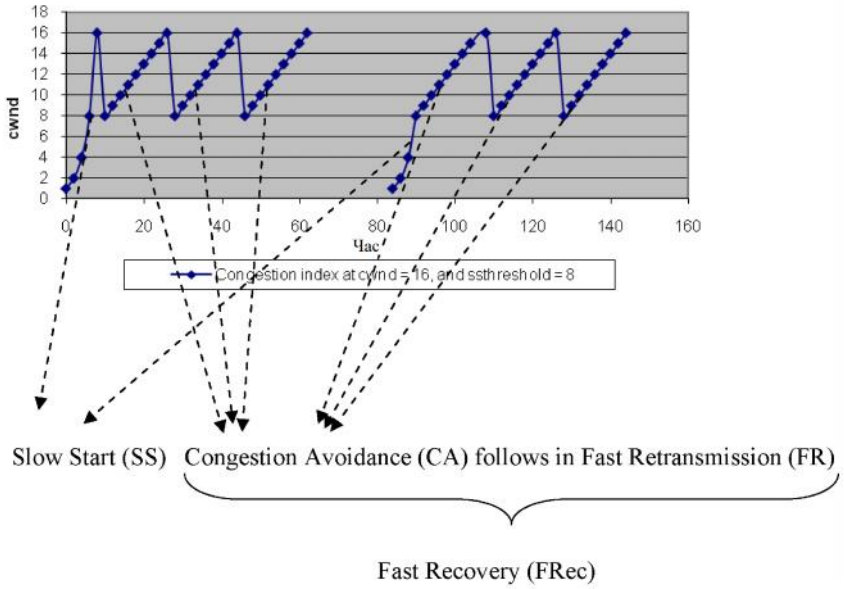

Рисунок 1.6 – Робота алгоритмів контролю перевантаження TCP

Рисунок 1.6 демонструє, що джерело починає виконувати алгоритм SS після встановлення з’єднання TCP з пунктом призначення, отже, він встановлює своє cwnd в один сегмент TCP і збільшує розмір cwnd, додаючи один сегмент TCP щоразу, коли отримує підтвердження від пункту призначення. На рисунку 1.6 зазначається, що джерело в алгоритмі SS збільшується експоненціально, доки в Інтернеті не виникає перевантаженість, оскільки існували втрачені сегменти TCP (це відбувається, коли розмір cwnd стає 16). У цей момент джерело перестає реалізовувати алгоритм SS і запускає алгоритм CA, після чого повторно передає втрачені сегменти TCP до їх пунктів призначення за допомогою алгоритму (методу) FR.

Послідовна реалізація алгоритмів CA та FR – це реалізація методу FRec. Цей процес повторюється до тих пір, поки не з’явиться тайм-аут повторної передачі, що може спричинити скорочення cwnd до одного сегменту TCP. У результаті цього джерело повертається до алгоритму SS. Прикладом переходу з CA в SS на рисунку 1.6 є наявність закінчення тайм-ауту повторної передачі. Нарешті, після закінчення тайм-ауту повторної передачі джерело надходить в SS та залишається там, доки розмір його cwnd не досягне значення ssthreshold. З іншого боку, якщо розмір cwnd для джерела стає більшим, ніж значення ssthreshold, джерело переходить з SS в CA. Тому, якщо розмір cwnd джерела дорівнює або менший, ніж розмір ssthreshold, джерело реалізує алгоритм SS, інакше реалізує алгоритм CA.

1.3 Управління перевантаженням

Одним із перших методів, який успішно застосовувався в управлінні перевантаженими був Drop Tail (DT) [8,9], який використовувався в мережах протягом кількох років. Метод DT покладається на великі розміри буферів маршрутизаторів, щоб досягти хорошої продуктивності мережі. Для подолання деяких недоліків DT, два типи підходів до контролю

перевантаженості були обговорені в роботі [10], тобто кінцева система та буфер маршрутизатора. Прикладом системи "з кінця в кінець" є TCP [3,4], в якому джерело TCP виявляє наявність перевантаження через спостереження за відкиданням пакетів у буферах маршрутизатора. Джерела реагують на цей тип перевантаженості за рахунок зниження швидкості їх передачі.

У другому типі контролю перевантаженості буфери маршрутизаторів відіграють значну роль у виявленні перевантаженості через оцінку поведінки мережі [11]. Зокрема, буфери маршрутизаторів виявляють перевантаженість, порівнюючи кількість пакетів , що надійшли з наявним буферним простором, в якому, якщо кількість пакетів перевищує певний рівень, то буфер маршрутизатора надсилає повідомлення про початок перевантаження.

1.3.1 Механізм управління ECN

У 1994 році було прийнято рішення застосовувати новий механізм ECN (Explicit Congestion Notification) – механізм явного повідомлення про виникнення перевантаження, на який в першу чергу були покладені функції керуваня навантаженням з’єднань протокола TCP, чутливих до істотних затримок та втрат пакетів. При цьому не застосовується відкидання пакету для пригнічення джерела передачі пакетів в якості явної індикації перевантаження.

Під час використання механізму ECN передавач повинен отримати інформацію про випадок перевантаження заздалегідь, ще до її виникнення, а не за фактом сформованого перевантаження. Зазвичай інформацією про стан перевантаження в мережі володіють саме маршрутизатори, які виконують свої функції на мережевому рівні моделі OSI, та саме через яких проходить потік даних TCP. Таким чином, для реалізації механізму ECN було необхідно застосовівати деякі біти в самому IP-заголовку. Для цього було запропоновано надати можливість встроювання ECN в IP-заголовок, а вже у 2001 році така реалізація ECN була ще доповнена та стандартизована. ECN використовує

два біта (6 і 7) полів ToS (Type of Service) IP-заголовку:

- біт ECT (ECN-capable transport) – біт номер 6: джерело встановлює його в 1 для інформування приймача про те, що він застосовує протокол TCP з підтримкою механізма ENC;

- біт CE (congestion experienced) – біт номер 7: маршрутизатор встановлює його в 1, інформуючи адресата про виникнення ситуації перевантаження.

Джерело та приймач повинні узгодити спільне використання механізму ECN саме під час активного з’єднання, таким чином, в результаті ECT-біти у всіх пакетах повинні будуть встановлені в 1. Завдяки цьму кожен з проміжних маршрутизаторів на шляху від джерела до адресата буде враховувати, що джерело і адресат використовують механізми ECN.

При виникненні ситуації перевантаження на будь-якому з транзитних маршрутизаторів, він шляхом зміни значення CE-біта в 1 може проінформувати приймальну сторону про перевантаження. Адресат у свою чергу, під час отримання від маршутізатора одиничного PE-біта, має змогу проінформувати джерело про виникнення перевантаження шляхом відсилання підтверження на сегмент, в якому буде встановленим прапор ENC-Echo. Цей прапор розташовано в заголовку TCP пакету та застосовує 9-й біт в полі reserved. Таким чином, після отримання сегменту, в якому встановлено прапор ENC-Echo, джерело повинно зменшити значення розміру свого вікна перевантаження cwnd.

Отримання такої інформації про виникнення перевантаження джерело повинно відобразити шляхом пересилки адресату сегмента уже з встановленим відповідно прапором CWR (congestion window reduced) – що позначає виконане зменшення розміру вікна перевантаження. Для цього застосовується 8-й біт поля reserved заголовка пакета TCP. Завдяки цьому, в механізмі ECN відбувається спільне використання мережевого та транспортного рівнів моделі OSI для цілей управління перевантаженням. Так на мережевому рівні сигнали про перевантаження йдуть від маршрутизатора,

а вже кінцеві адресати використовують сигналізацію про перевантаження за допомогою зворотного зв’язку із застосуванням транспортного рівня.

1.3.2 Метод Drop-tail

Метод DT вже багато років застосовується для контролю за випадками переповнення черг маршрутизаторів. Цей метод покладається на встановлення максимальної довжини буфера маршрутизатора для досягнення високої пропускної здатності. Однак це також може призвести до великої затримки черги пакетів, що не бажано особливо для трафіку в режимі реального часу. З іншого боку, якщо буфер маршрутизатора має невеликий розмір, буде досягнута низька затримка в черзі, але пропускна здатності погіршиться, а також зросте ймовірність втрат пакетів. Мережеві джерела дізнаються про перевантаженість за допомогою методу DT виключно після переповнення буфера маршрутизатора.

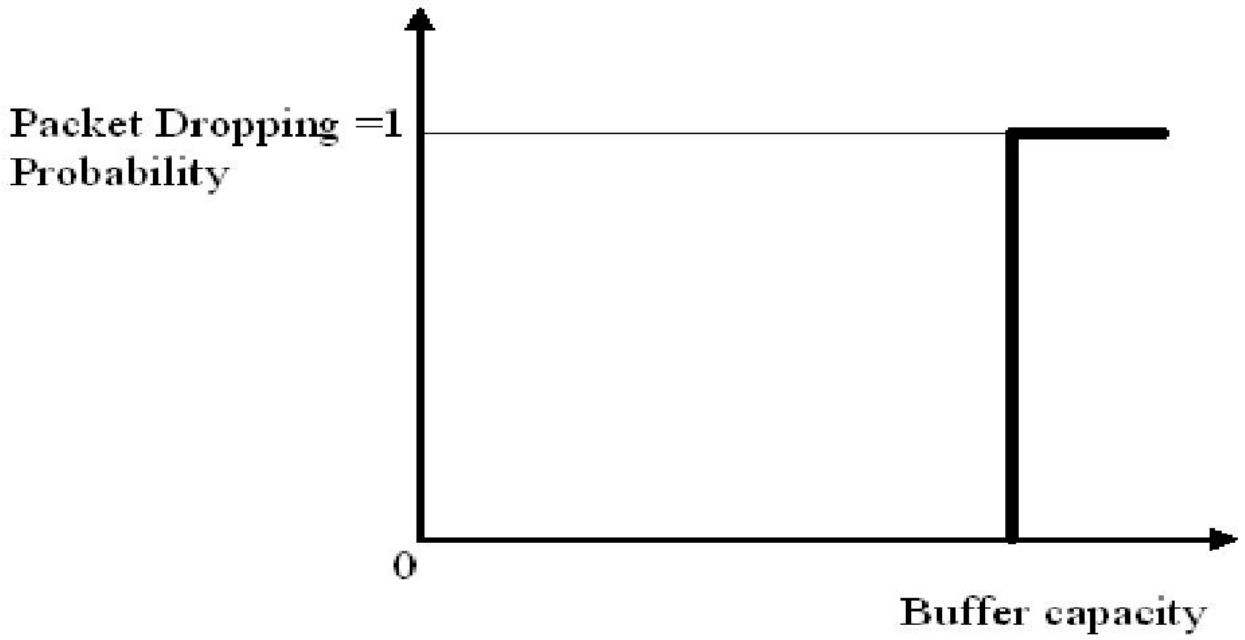

Рисунок 1.7 – Функція ймовірності випадання пакетів DT

На рисунку 1.7 показано D_p для методу DT, коли використання цього методу в мережах TCP може погіршити продуктивність TCP, оскільки джерела корегують свою швидкість передачі виключно після переповнення буферів маршрутизатора DT. Є й інші недоліки, пов’язані з методом DT, які також погіршують продуктивність TCP. Одним із таких явищ є явище блокування, яке виникає, коли одне чи кілька джерел монополізують простір буфера маршрутизатора тривалий час, а решта джерел чекають. Іншим недоліком є повні черги, які трапляються, коли буфери маршрутизатора майже заповнені, і вони залишаються такими на деякий час, таким чином збільшуються як затримка пакетів в черзі, так і швидкість їх видкидання. Третім недоліком є явище глобальної синхронізації, яка впливає на пропускну здатність перевантаженої мережі, в якій буфери маршрутизаторів повідомляють всім джерелам про початок перевантаженості з метою зниження їх швидкості передачі. Однак, оскільки декілька джерел не відіграють жодної ролі у спричиненні перевантаженості, зниження їх швидкості передачі є зайвим і може знизити продуктивність.

Ще одне важливе обмеження DT – це упередженість по відношенню до бурхливих потоків руху. Контроль швидкості TCP надає для досягнення високої продуктивності перевагу потокам, які мають не велике RTT. Такі потоки трафіку збільшують свої вікна передачі швидше, ніж ті, що мають велике RTT, і як результат, потоки трафіку з малим RTT захоплюють мережеві ресурси швидше, ніж ті, що мають велике RTT. Це може спричинити несправедливих розподіл ресурсів поміж мережевими потоками.

1.4 Переваги застосування комбінованих механізмів управління

Під комбінованими механізмами для управління перевантаженням слід розуміти спільне застосування алгоритмів, як на мережевому, так і на транспортному рівні.

Протокол передачі даних TCP реалізовано на транспортному рівні OSI

тільки на кінцевих абонентах. Проміжні маршрутизатори мають змогу використову лише 3 нижніх рівня моделі (самий верхній рівень мережевий), тому вони не маєть змоги безпосередньо впливати на навантаження, яке створюють кінцеві системи в мережі. Джерелу необхідно якимось чином отримувати інформацію про потенційне або вже маюче місце перевантаження, і таким чином, зменшити трафік переданих даних. Так як сам маршрутизатор не має механізмів впливу на дані транспортного рівня, то єдиним засобом сповіщення джерела має бути використання так званих сигнальних бітів в заголовку пакету на мережевому рівні. Такий механізм управління перевантаженнями виконується в маршрутизаторах і застосовує принцип зворотного зв’язку. Таке спільне застосування механізмів, що базуються в маршрутизаторах, та механізмів, реалізованих в самому протоколі TCP, дає змогу отримати гарні результати при вирішенні задач управління перевантаженнями в сучасних мережах TCP/IP. Таке гармонійне поєднання використовується в механізмі TCP ENC.

Одним з основних завдань під час управління перевантаженнями і відповідно забезпеченні необхідного рівня якості обслуговування є управління мережевим трафіком. У загальному вигляді, під терміном «керування трафіком» розуміється сукупність різних алгоритмів, які реалізовані як апаратно, так і програмно для забезпечення необхідного рівня якості обслуговування. Управління трафіком повинно включати в себе мережеве планування, за результатами якого визначається майбутня топологія мережі і пропускна спроможність її ліній, та оптимізації, що дозволяє виконати безпосередньо ефективний розподіл мережевого трафіку. З другого боку, боротьба з перевантаженнями має також підтримувати ефективне використання ресурсів мережі на високих рівнях її навантаження.

1.5 Постановка завдань дослідження

Для нормального функціонування каналів мережі, потрібна підтримка пропускної здатності на досить прийнятному рівні. Ряд існуючих методик на основі виконання зворотного зв’язку роблять можливим управління пропускною здатністю в залежності від існуючого навантаження: пригніченням, або стримуючим пакетом, сигналізацією про виникнення перевантаження. Способи забезпечення необхідного QoS без забезпечення зворотного зв’язку, такі як справедливий розподіл ресурсів мережі, низка алгоритмів управління чергами (в тому числі RED), а також різні алгоритми профілювання мережевого навантаження, функціонують на рівні мережних пристроїв. Перераховані підходи до забезпечення QoS розроблялися для пуассонівських моделей трафіку і не враховували самоподібної структури комп’ютерного трафіку. Таким чином, використання більш адекватних мережевих моделей трафіку повинно сприяти розробці більш ефективних протоколів передачі інформації і методик забезпечення QoS.

Під час виконання магістерської атестаційної роботи необхідно:

- провести аналіз сучасних літературних джерел, присвячених проблемам підвищення пропускної спроможності в комп’ютерних мережах;
- провести огляд сучасних моделей та методів управління інформаційними потоками;
- запропонувати метод управління чергами маршрутизаторів та провести імітаційне моделювання запропонованих в роботі теоретичних викладок;
- провести аналіз отриманих результатів.

Таким чином, мета магістерської атестаційній роботи є особливо актуальною у зв’язку зі стрімким ускладненням інформаційних технологій, які потребують все зростаючі вимоги до продуктивності комп’ютерних мереж.

2 АЛГОРИТМИ УПРАВЛІННЯ ЧЕРГАМИ МАРШРУТИЗАТОРІВ

2.1 Класифікація алгоритмів AQM

Активне керування чергою (AQM) – це набір методів, розроблених головним чином для раннього виявлення перевантаження у мережах та досягнення високих показників їх функціонування. Прикладами алгоритмів AQM є: RED, ARED, REM [14], BLUE [15], Stochastic Fair BLUE (SFB), Generalised Random Early Evasion Network (GREEN), Adjustment RED, Fuzzy BLUE, Fuzzy Exponential Marking (FEM) та ін.

Крім того, були запропоновані алгоритми AQM для подолання обмежень DT, розглянутих раніше. Наприклад, на відміну від методу DT, який починає скидати пакети просто після переповнення буферів маршрутизатора, методи AQM зазвичай починають скидати пакети на ранніх стадіях, отже, це дозволяє джерелам знижувати швидкість передачі даних раніше, ніж буфери маршрутизатора повністю переповняться. У методах AQM мережеві джерела ідентифікують перевантаженість або за допомогою скидання пакетів на буфери маршрутизатора (неявний зворотний зв’язок), або шляхом встановлення біта явного повідомлення про перевантаження (ECN) у заголовку пакета (явний відгук).

Можна розробити та реалізувати метод AQM, якщо можна відповісти на наступні три питання:

1) Яка метрика перевантаженості допоможе контролювати затори?

2) Як перевантажений буфер маршрутизатора інформує з’єднання, щоб змінити швидкість передачі?

3) Що таке функція ймовірності випадання пакетів і яка політика використовується для скидання пакетів?

Відповідно до біта ECN, коли відбувається перевантаження, буфер маршрутизатора встановлює біт ECN у заголовку пакета, що надходить, а

потім відправляє пакет до місця призначення. Коли пункт призначення отримує позначений пакет, він повідомляє джерело про затори через підтвердження.

На рисунку 2.1 демонструється класифікація AQM алгоритмів.

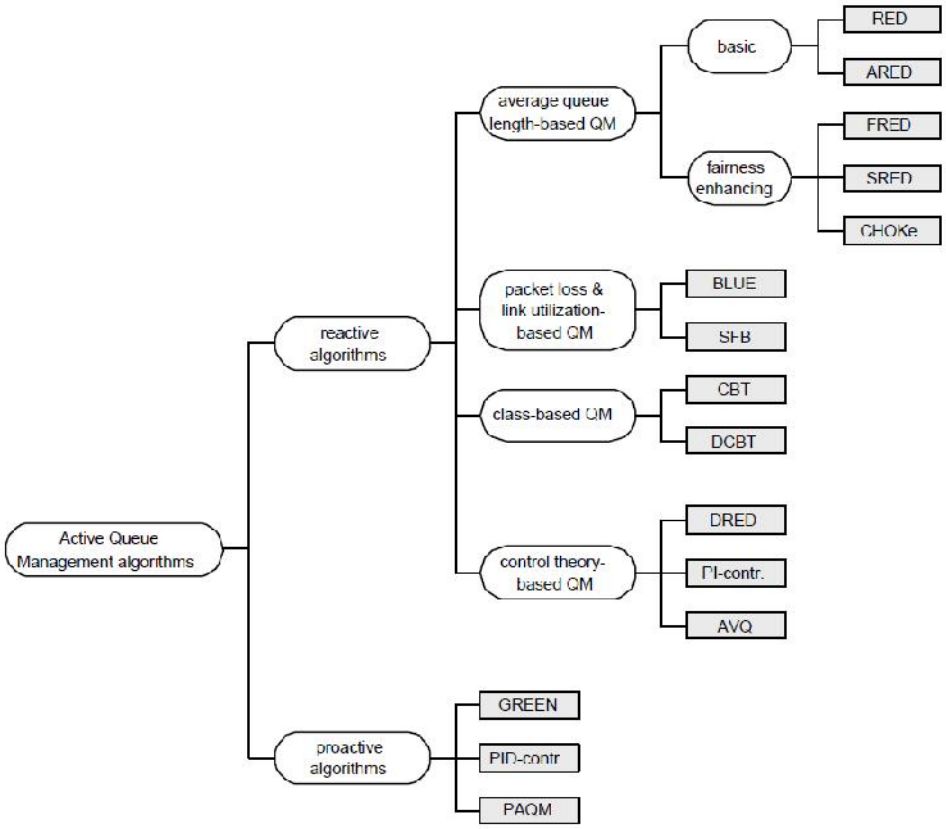

Рисунок 2.1 – Класифікація алгоритмів AQM

2.2 Алгоритм раннього випадкового відкидання

Метод раннього випадкового відкидання було запропоновано у 1989 році як метод контролю перевантаженості, який використовує довжину черги (ql) та фіксовану порогову позицію в буфері маршрутизатора для виявлення перевантаження. Коли ql стає більшим за фіксовану порогову позицію в буфері маршрутизатора, буфер маршрутизатора починає відкидати пакети, що надходять з постійною швидкостю. Крім того, оскільки Early Random

Drop є одним з алгоритмів AQM, він починає скидати пакети на ранніх стадіях ще до того, як буфер маршрутизатора переповниця, що врешті-решт знижує глобальну синхронізацію, і підтримує рівень пропускної здатність в мережі. Перевагою методу Early Random Drop є його здатність виявляти занадто активні потоки.

2.3 Алгоритм Decbit

Дебіт використовує бітний бінарний індикатор та середню довжину черги (aql – averege ql) в якості метрики перевантаженості в мережі. Для уточнення, для кожного пакета, який надходить до буфера маршрутизатора, Decbit обчислює розмір поточного aql, якщо він менший ніж один, жодні пакети не выдкидаються, тоді як якщо розмір aql перевищує один, Decbit позначає пакет, що надходить, і передає його до місця призначення. Нарешті, позначений пакет відправляється назад через підтвердження від місця призначення до джерела, і джерело може ідентифікувати перевантаженість зі вмісту підтвердження. Слід зазначити, що джерела змінюють свої вікна передачі один раз на два RTT, і якщо половина або більше пакетів останнього розміру вікна позначені бітом індикації перевантаженості, то розмір вікна передачі зменшується експоненціально, в іншому випадку вона збільшується лінійно.

2.4 Алгоритм випадкового раннього виявлення (RED)

Випадкове раннє виявлення (Random Early Detection – RED) було запропоновано в 1993 р. С. Флойдом та В. Якобсоном [12]. Це метод контролю заторів на маршрутизаторах, який був успішно прийнятий IETF в RFC 2309. RED відкидує пакети випадковим чином перед переповненням буфера маршрутизатора та використовує обчислені aql та два пороги (minthreshold та maxthreshold) для управління перевантаженістю.

Вищеописані два пороги відповідають мінімальному пороговому та максимальному позиційному положенню в буфері маршрутизатора відповідно.

Як правило, RED може виявити перевантаженість таким чином. Після того, як маршрутизатор обчислить aql, він порівнює його з мінімальним порогом і максимальним порогом. Якщо aql менше за мінімальний поріг, то перевантажень не відбудеться, і тому жодні пакети не відкидуються. Тоді як якщо aql знаходиться між двома порогами, то пакети, що надходять, будуть імовірнічно скинуті, і RED обчислює ймовірність D_p, щоб полегшити перевантаження.

Нарешті, коли aql перевищує максимальний поріг, тоді кожен пакет, що надходить, буде відкинуто, а значення D_p стане рівним 1. На рисунку 2.2 показано загальний вид RED.

For every arriving packet at a RED router buffer, the router buffer does the following:

1. Calculate the aql.
2. Check the aql position with two thresholds (min $threshold$ and max $threshold$) positions at a RED router buffer, then the router performs it's right action as follows:

   if $(aql < \min threshold)$
   No congestion is occurred.

   if $(aql \geq \max threshold)$
   A heavy congestion is existed. As a result, drops/marks every arriving packet with $D_p = 1$;

   if $(\min threshold \leq aql \;\&\&\; aql < \max threshold)$
   Congestion is presented; therefore marks/drops every arriving packet randomly with calculating its D_p value.

Рисунок 2.2 – Псевдокод алгоритму RED

RED використовує кілька параметрів для обчислення як aql, так і D_p. Ці параметри можна визначити наступним чином:

current_time – поточний час;

idle_time – час запуску простою в буфері маршрутизатора RED;

n – кількість пакетів, що надсилаються в буфер RED маршрутизатора протягом інтервалу;

C – лічильник, який представляє кількість пакетів, що надійшли до буфера маршрутизатора RED і не були скинуті з моменту скидання останнього пакету;

D_p – миттєва ймовірність випадання пакету;

D_{init} – початкова ймовірність відкиданнядання пакету;

q_instantaneous – миттєва довжина черги;

qw – вага черги;

D_{max} – максимальне значення D_p;

q(time) – лінійна функція для часу.

З рисунка 2.3 видно, що жоден пакет не викидається з черги, коли обчислене aql менше за minthreshold. Хоча пакети починають відкидатися ймовірнісно, коли aql знаходиться між minthreshold і maxthreshold, при цьому значення D_{init} лінійно збільшується, поки aql не досягне maxthreshold. Однак якщо aql більше, ніж або рівний maxthreshold, RED відкидає кожен пакет, що надходить, і значення D_{init} стає рівним 1.

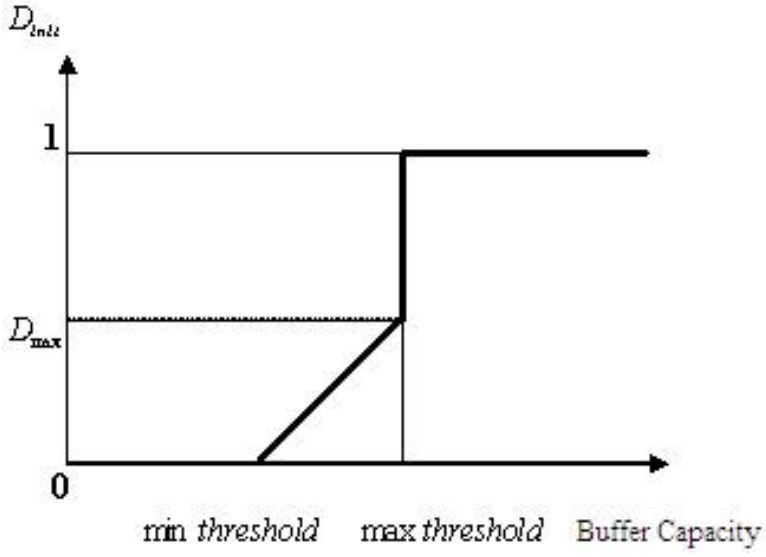

Рисунок 2.3 – Залежність D_{init} від aql

На рисунку 2.4 наведені повні деталі RED.

1. Initialisation stage
   $C = -1$;
   $aql = 0.0$;
2. For every arriving packet at a RED router buffer:
   2.1 Calculate the aql for this packet at the RED router buffer.
   2.2 Examine the queue status at a router buffer, is it empty or not?
   if (The queue at a RED router buffer = = empty)
   {
   Compute n, where $n = q(current\_time - idle\_time)$;
   $aql = aql \times (1 - qw)^n$;
   }
   else
   $aql = aql \times (1 - qw) + qw \times q\_instantaneous$;
3. Determine a congestion status at the RED router buffer:
   $if(aql < \min threshold)$
   {
   $D_p = 0$; // No congestion is occurred
   Set $C = -1$;
   }

   $else\,if(\min threshold \le aql \,\&\&\, aql < \max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:

   $$D_{init} = \frac{D_{max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$

   $$D_p = \frac{D_{init}}{(1 - C \times D_{init})};$$

   Marks/Drops arriving packet randomly with calculating its D_p value (probabilistically) due to a RED router buffer is congested.

   Set $C = 0$;
   }
   $else$ // $if(aql \ge \max threshold)$
   {
   Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is existed at a RED router buffer;

   Set $C = 0$;
   }
4. When the RED router buffer becomes empty
   Set $idle\_time = current\_time$;

Рисунок 2.4 – Псевдокод алгоритму RED

Відповідно до [12], є вказівки щодо встановлення деяких параметрів RED, тобто qw, minthreshold, maxthreshold та D_{max}. Ці вказівки можна пояснити наступним чином:

1) qw: Цей параметр використовується для обчислення aql і має межу діапазону, тобто нижню межу (0,001) та верхню межу (показано нижче). Якщо qw встановлюється на нижне значення, aql повільно збільшується, і тому RED може не виявити заторів на попередніх етапах. З іншого боку, верхнє значення qw можна пояснити, використовуючи наступний приклад: припустимо, що буфер маршрутизатора RED не працює, тобто aql дорівнює нулю.

Тоді через певний проміжок часу N пакетів надійшло в буфер маршрутизатора, aql обчислюється в цій ситуації наступним чином:

$$aql(N) = \sum_{i=1}^{N} i\left(1-qw\right)^{N-i} qw, \tag{2.1}$$

$$aql(N) = qw(1-qw)^{N} \sum_{i=1}^{N} i\left(1-qw\right)^{-i}, \tag{2.2}$$

$$aql(N) = qw(1-qw)^{N} \sum_{i=1}^{N} i\left(\frac{1}{(1-qw)}\right)^{i}. \tag{2.3}$$

У цьому випадку верхнє значення qw повинно бути встановлене відповідно до рівняння 2.3, оскільки aql(N)<minthreshold.

2) D_{max} можна знати, коли aql досягає maxthreshold позиції в буфері маршрутизатора. Як правило, RED зберігає aql між minthreshold і maxthreshold, щоб уникнути сильних перевантажень. Найкраще встановити значення D_{max}, яке не перевищує 0,1, щоб контролювати перевантаженість, зберігаючи значення aql між minthreshold і maxthreshold. Якщо для параметра D_{max} встановлено значення, що перевищує 0,1, то RED-маршрутизатор не

зможе утримувати aql між вищезгаданими двома порогами, і, таким чином, кожен пакет, який надійде до буфера маршрутизатора, буде скинутий, тобто D_p=1.

3) Встановлення minthreshold і maxthreshold. Ці пороги встановлюються значеннями, які гарантують бажане значення aql. minthreshold, як правило, залежить від стану навантаження на трафік, тобто якщо навантаження на трафік досить розривне, minthreshold встановлюється на велике значення для підтримки продуктивності. Тим не менш, якщо minthreshold встановлено на низьке значення, RED відкидає пакети на ранніх етапах і досягне низької пропускної здатності. З іншого боку, встановлення максимального порогу maxthreshold залежить від найбільшого значення затримки черги, досягнутого, коли aql досягає maxthreshold. Більше того, параметр maxthreshold часто встановлюється як мінімум удвічі більшим за minthreshold для підтримки продуктивності [12].

RED має багато переваг, деякі з яких узагальнені наступним чином:

1) Маршрутизатор RED дозволяє уникнути проблеми зміщення проти бурхливих потоків трафіку, яка виникає при прийнятті пакетів від потоків з малим RTT та відкидання пакетів з потоків з великим RTT. RED усуває цю проблему за допомогою методу випадання рандомізованих пакетів.

2) RED дозволяє уникнути явища глобальної синхронізації, яке виникає, коли всі джерела в мережі разом освідомляються про перевантаження. RED змушує всі джерела реагувати на затори, зменшуючи розміри вікон їх передачі, щоб полегшити первантаження.

Однак це рішення є не найкращим через те, що багато хто з цих джерел, можливо, не зробили внесок у спричинення цього затору. Отже, не потрібно, щоб ці джерела зменшували розміри вікон cwnd і таким чином, призводили до зниження пропускної здатності. Маршрутизатор RED уникає проблеми глобальної синхронізації, використовуючи метод рандомізації в скиданні пакетів, але буфер маршрутизатора RED повідомляє лише про джерела, які спричинили затори в мережі, і ці джерела відповідають на це

повідомлення, зменшуючи їх швидкість передачі.

3) маршрутизатор RED може визначити агресивний потік, тобто потік, який має більше, ніж його справедлива частка мережевих ресурсів. RED-маршрутизатор може визначити, пакети яких джерел мають найбільший D_p і тому потоки з найбільшим D_p являють собою ті, які мають найбільшу частку в пропускній здатності, таким чином, RED маршрутизатор може виявляти недобросовісні потоки.

Прикладами невідповідних потоків є джерела TCP, які мають велику долю пропускної здатності на маршрутизаторі і мають або великий розмір вікна, або малий RTT.

Незважаючи на переваги RED, які були обговорені вище, він страждає від таких недоліків:

1) aql змінюється залежно від рівня перевантаженості і може бути чутливим до початкових значень RED. Наприклад, коли виникає сильний затор, це вказує на те, що aql близьке до maxthreshold позиції в буфері маршрутизатора.

Тоді як, якщо затори трапляються на попередніх етапах, то aql буде близьким до мінімального порогу minthreshold. Іншими словами, aql неможливо передбачити, оскільки середня затримка черги залежить від параметрів RED та стану потоку трафіку (важкий або легкий).

2) Алгоритм RED не може стабілізувати aql між minthreshold і maxthreshold положенням в буферах маршрутизатора, коли навантаження трафіку змінюються раптово, наприклад, для бурхливого трафіку. Таким чином, aql може перевищувати максимальне порогове положення, що призведе до частого відкидання пакетів з буфера маршрутизатора.

2.5 Алгоритм ARED

Адаптивне випадкове раннє виявлення (ARED) [13] було запропоновано для усунення деяких недоліків RED, вказаних раніше.

Основною зміною в ARED є стабілізація aql між мінімальним порогом і максимальним порогом за рахунок динамічної модифікації параметра D_{max}. Як результат, алгоритм ARED уникає відкидання пакетів, що надходять, запобігаючи досягненню значенням aql максимального порогу. Модифікована версія ARED була розроблена в 2001 році і вона відрізняється від оригінальної ARED в четвертій властивості:

- мінімальний поріг встановлюється простим способом, аналогічним [13];
- максимальний поріг встановлюється автоматично на основі мінімального порогового значення, а вага черги (qw) встановлюється аналогічно red;
- D_{max} (з п. 2.4) застосовується з метою стабілізації aql на цільовому рівні, тобто:

$$(T_{aql}), \text{ де } T_{aql} = \frac{(\max threshold + \min threshold)}{2}, \qquad (2.4)$$

Де D_{max} завжди зберігається між [0,01, 0,5].

4) для встановлення значення D_{max} використовується метод метод збільшення додаткового збільшення мультиплікативного скорочення (AIMD) замість мультиплікативного збільшення мультиплікативного зменьшення (MIMD).

Псевдокод модифікованого алгоритму ARED показано на рисунку 2.5, де параметр періоду часу встановлено на 0,5 секунди, а T_{aql} обмежено наступним інтервалом:

$$\left\{ \begin{matrix} \min threshold + 0.4(\max threshold - \min threshold), \min threshold \\ +0.6(\max threshold - \min threshold) \end{matrix} \right\}.$$

Нарешті, d_1 і d_2 відповідають кількості, вилученій з D_{max}, коли aql меньшеT_{aql}, і кількості, доданій до D_{max}, коли aql вище T_{aql} відповідно.

$$d_1=0{,}9 \text{ і } d_2 = \min\left(0.01, \frac{D_{max}}{4}\right).$$

Every period of time in seconds
$if\left(aql < T_{aql} \;\&\&\; D_{\max} \geq 0.01\right)$
$D_{\max} = D_{\max} \times d1;$ // Decrease $D_{\max}$ value

$esleif\left(aql > T_{aql} \;\&\&\; D_{\max} \leq 0.50\right)$
$D_{\max} = D_{\max} + d2;$ // Increase $D_{\max}$ value

Рисунок 2.5 – Псевдокод алгоритму ARED

2.6 Алгоритм GRED

Для подолання деяких недоліків RED, обговорених раніше, було запропоновано алгоритм GRED (Gentle – пологий). Його основна мета – стабілізація aql на певному рівні. GRED використовує підхід RED при обчисленні D_P. Псевдокод для GRED містить три основні пороги (minthreshold ,maxthreshold, double maxthreshold)та цільовий рівень для середньої довжини черги T_{aql}.

Коли aql на маршрутизаторі нижче мінімального порогового значення, жодні пакети не відкидаються.

Тоді як, якщо aql знаходиться між мінімальним порогом і максимальним порогом, маршрутизатор скидає вхідні пакети випадковим чином схожим на алгоритм RED.

Більше того, якщо aql знаходиться між максимальним порогом і подвійним максимальним порогом, маршрутизатор відкидає прибуваючі пакети випадковим чином. Слід зазначити, що в цьому випадку aql стабілізується на T_{aql} (значення T_{aql} обчислюється рівнянням 2.5). Нарешті, якщо aql дорівнює або перевищує подвійний максимальний поріг, маршрутизатор GRED позначає пакети, що надходять D_p=1 (пакети відкидаються).

1. Initialisation stage
   $C = -1$;
   $aql = 0.0$;
2. For every arriving packet at a GRED router buffer:
   2.1 Calculate the aql for this packet at a GRED router buffer.
   2.2 Examine the queue status at the router buffer is it empty or not?
   if (The queue at the GRED router buffer = = empty)
   {
   Compute n, where $n = q(current\_time - idle\_time)$;
   $aql = aql \times (1 - qw)^n$;
   }
   else
   $aql = aql \times (1 - qw) + qw \times q\_instantaneous$;
3. Determine a congestion status at the GRED router buffer:
   $if(aql < \min threshold)$
   {
   $D_p = 0$; // No congestion is occurred
   Set $C = -1$;
   }
   $elseif(\min threshold \le aql \&\& aql < \max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:
   $$D_{tmp} = \frac{D_{max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$
   $$D_p = \frac{D_{tmp}}{(1 - C \times D_{tmp})};$$
   Marks/Drops arriving packet randomly using its calculated D_p value
   (probabilistically);
   Set $C = 0$;
   }
   $elseif(\max threshold \le aql \&\& aql < double\max threshold)$
   {
   $C = C + 1$;
   Calculate D_p value for the arriving packet as follows:
   $$D_{tmp} = D_{max} + \frac{(1 - D_{max}) \times (aql - \max threshold)}{(double\max threshold - \max threshold)};$$
   $$D_p = \frac{D_{tmp}}{(1 - C \times D_{tmp})};$$
   Marks/Drops arriving packet randomly using its calculated D_p value
   (probabilistically);
   Set $C = 0$;
   }
   $else\ //\ if(aql \ge double\max threshold)$
   {
   Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is existed;
   Set $C = 0$;
   }
4. When a GRED router buffer becomes empty
   Set $idle\_time = current\_time$;

Рисунок 2.6 – Алгоритм GRED

Рівняння 2.6 та 2.7 – приклади встановлення значень максимального порогового та подвійного максимального порогових значень, відповідно, мінімальний поріг та qw задаються значеннями, аналогічними значенням RED.

$$T_{aql} = 2\min threshold = \frac{(\max threshold + \min threshold)}{2}, \qquad (2.5)$$

$$\max threshold = 3\min threshold\ , \qquad (2.6)$$

$$double\ \max threshold = 2\max threshold\ . \qquad (2.7)$$

Нарешті, GRED має деякі обмеження:

- більше порогових значень, яких потрібно дотримуватися;
- параметризація;
- використання aql.

2.7 Алгоритм DRED

Динамічний Random Early Drop (DRED) було запропоновано в 2001 році, щоб подолати aql залежності від кількості TCP з'єднань. Як і більшість методів AQM, алгоритм DRED стабілізує ql на заздалегідь визначеному рівні незалежно від кількості з'єднань TCP в мережі. Більше того, DRED залежить від фіксованої одиниці часу (Ct), де для кожного Ct, ql та сигнал помилки Err(i) обчислюються відповідно. Обчислене Err(i), показано в рівнянні 2.8 та залежить як від ql, так і від T_{ql}.

$$Err(i) = ql(i) - T_{ql}\ . \qquad (2.8)$$

Тоді, виходячи зі значення Err(i), відфільтрований сигнал помилки Fil(i) виявляється таким чином:

$$Fil(i) = Fil(i-1)(1-qw) + Err(i)qw. \quad (2.9)$$

З рівняння 2.9 видно, що DRED використовує фільтр низьких частот, подібний до RED для обчислення Fil(i), де фільтр низьких частот є експоненціально зваженим середнім (EWMA). Параметр qw позначає вагу черги, і він використовується для оцінки значення Fil(i). Цей Fil(i) можна використовувати кожну одиницю часу для оновлення D_P. Нарешті, ємність буфера маршрутизатора DRED (K) та параметра під назвою ε використовується для ідентифікації посилення зворотного зв'язку D_P, що показано у рівнянні 2.10:

$$D_P(i) = \min\left\{\max\left(D_P(i-1) + \varepsilon\frac{Fil(i)}{K}, 0\right), 1\right\}. \quad (2.10)$$

Крім того, значення с оновлюється лише тоді, коли ql перевищує або дорівнює порогу падіння (th) для досягнення доброго використання канала. Іншими словами, при ql<th не відбувається відкидання пакетів, що вказує на те, що ql є вирішальним параметром для виявлення перевантаженості в DRED.

Нарешті, DRED відзначає пакети, додаючи біт ECN у їх заголовки або відкидає пакети в буфери маршрутизатора.

У таблиці 2.1 наведені параметри DRED та їх рекомендовані значення.

Таблиця 2.1 – Параметри DRED

Параметр	Значення
Одиниці часу C(t)	Час відправки 10 пакетів
Цільовий рівень для довжини (T_{ql})	$T_{ql}=0.5K$
Поріг перевантаженості	$th = 0.9T_{ql}$
Ємність буфера на маршрутизаторі	K
Вага черги (qw)	0.002
Параметр управління системою (ε)	0.00005

Алгоритм, представлений на рисунку 2.7, пояснює, як отримується D_p.

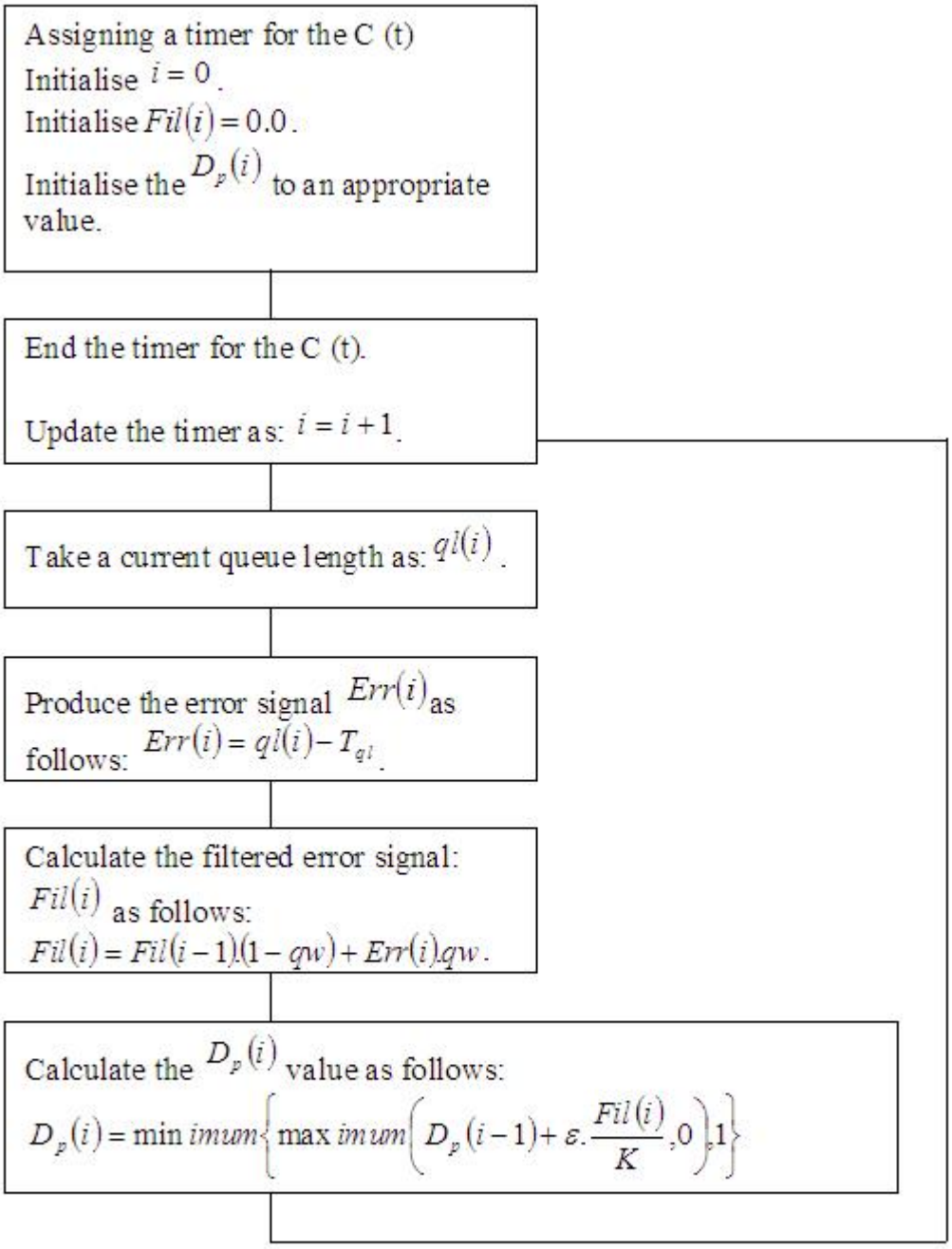

Рисунок 2.7 – Алгоритм обчислення D_p

DRED повинен встановити свої параметри на конкретні значення, щоб забезпечити хорошу продуктивність (параметризація). DRED використовує миттєву довжину черги, а не середню довжину черги, і це може призвести до зайвого ймовірнісного відкидання пакетів.

2.8 Алгоритм Adjustment RED

Багато методів AQM, включаючи RED, DRED, BLUE та SRED, використовують вихідні черги (OQ) як один з показників перевантаженості. Однак можна використовувати комбінацію черг на вхід (IQ) та OQ на маршрутизаторах як метрику перевантажень в алгоритмі під назвою RED Adjustment. IQ – черги, які існують на стороні входів маршрутизаторів. Коригування RED замінює параметр ql в рівнянні aql RED на загальне зважене значення IQ та OQ, як показано в рівнянні (3.11). У рівнянні qw позначає вагу черги, IQ відповідає загальній віртуальній черзі, а ω являє параметр зважування, який використовується з IQ, щоб допомогти забезпечити баланс між швидкістю втрати пакету та aql.

$$aql = aql(1 - qw) + qw(\omega IQ + OQ). \qquad (2.11)$$

2.9 Алгоритм BLUE

Алгоритм BLUE був запропонований для підвищення продуктивності класичного алгоритму RED. BLUE покладається на один D_P і певний поріг. Якщо довжина черги буфера на маршрутизаторі більша за поріг, BLUE збільшує D_P для управління перевантаженням. Крім того, якщо буфер порожній, D_P зменшується. На відміну від RED, який використовує aql як основний показник перевантаженості, BLUE аналізує втрати пакетів, ступінь використання каналу та величину буфера на маршрутизаторі.

Як і RED, BLUE уникає проблем, пов’язаних із агресивними потоками трафіку, надаючи місця для розміщення пакетів у своєму буфері маршрутизатора. BLUE використовує кілька параметрів для регулювання значення D_P, тобто freeze_time використовується для визначення найнижчого періоду часу між двома послідовними корегуваннями D_P. Значення freeze_time може бути встановлено як фіксований параметр, хоча він також

може бути встановлений як випадковий параметр, щоб уникнути глобальної синхронізації.

Initialisation stage

$C = -1;$

$aql = 0.0;$

2. For every arriving packet at a router buffer:

2.1 Calculate the aql for this packet at a router buffer.

2.2 Examine the queue status at the router is it empty or not?

if (The queue at the router buffer = = empty)

{

Compute n, where $n = q(current\_time - idle\_time);$

$aql = aql \times (1 - qw)^n;$

}

else

$aql = aql \times (1 - qw) + qw \times (\omega \times IQ + OQ);$

3. Determine a congestion status at the router buffer:

$if(aql < \min threshold)$

{

$D_p = 0$; // No congestion is occurred

Set $C = -1;$

}

$elseif(\min threshold \le aql \;\&\&\; aql < \max threshold)$

{

$C = C + 1;$

Calculate D_p value for the arriving packet as follows:

$$D_{init} = \frac{D_{\max} \times (aql - \min threshold)}{(\max threshold - \min threshold)};$$

$$D_p = \frac{D_{init}}{(1 - C \times D_{init})};$$

Marks/Drops the arriving packet randomly using its calculated D_p value (probabilistically).

Set $C = 0;$

}

$else$ // $if(aql \ge \max threshold)$

{

Marks/Drops every arriving packet with $D_p = 1$ since a heavy congestion is existed;

Set $C = 0;$

}

4. When the router buffer becomes empty

Set $idle\_time = current\_time;$

Рисунок 2.8 – Алгоритм Adjustment RED

P_{inc} і P_{dec} використовуються для ідентифікації величини, на яку D_p слід збільшувати або зменшувати. Крім того, P_{inc} повинен бути більшим, ніж P_{dec}, щоб уникнути недовикористання ресурсів. BLUE вирішує проблему глобальної синхронізації шляхом вибору випадкового параметра freeze_time або випадковим відкиданням пакетів, подібно до підходу RED. Обмеження BLUE такі ж, як у DRED.

On losing of the packets or $(threshold < B_{length})$

$$if((current\_time - last\_adjustment) > freeze\_time)$$

{

$$D_p = D_p + P_{inc};$$

$$last\_adjustment = current\_time;$$

}

When the buffer is empty (or $B_{length} = 0$**)**

$$if((current\_time - last\_adjustment) > freeze\_time)$$

{

$$D_p = D_p - P_{dec};$$

$$last\_adjustment = current\_time;$$

}

Рисунок 2.9 – Алгоритм BLUE

2.10 Алгоритм Stochastic Fair BLUE

Стохастичний алгоритм BLUE (SFB) – алгоритм, подібний до BLUE, що залежить від *Bv*-бункерів, де *B* являє собою кількість бункерів, розташованих у кожному *v* рівні, а *v* – кількість рівнів, де кожен *v* рівень має автономну хеш-функцію. Метою хеш-функцій є відображення потоку в бункер на певному рівні (рівень, з яким пов’язана хеш-функція).

Крім того, хеш-функція покладається на ID з’єднання (адреса джерела, адреса призначення, номер порту джерела, номер порту призначення,

протокол) у політиці відображення. SFB використовує стратегію BLUE при обчисленні D_p, в якому кожен контейнер в SFB має D_p, який оновлюється відповідно до розміру черги бункера. Отже, коли пакет надходить до буфера маршрутизатора, він отримує відображення на один бункер на кожному рівні за допомогою хеш-функції.

Якщо кількість пакетів, відображених у певному бункері, перевищує заданий поріг (перевищує поріг заповнення бункера (B_{size})), то значення D_p для цього бункера буде збільшуватися, тоді як якщо кількість пакетів стане меншою B_{size}, то D_p зменшиться.

1. Identification stage
   Arr [v levels] [B bins]: array of v . B bins;
   $Idle = 0$;
   $Busy = 1$;
2. Maps the arriving packet into a single bin in every level using the hash functions, as follows:
   Compute $hash_0, hash_1, hash_2, \ldots, hash_{v-1}$

$for(j = 0; j < v; j++)$

{

$if(Arr[j][hash_j].q_{size} == Idle)$

$Arr[j][hash_j].D_p = Arr[j][hash_j].D_p - P_{dec}$;

$if(Arr[j][hash_j].q_{size} > B_{size})$

{

$Arr[j][hash_j].D_p = Arr[j][hash_j].D_p + P_{inc}$;

Drops/Marks the arriving packet similar to the BLUE algorithm;

}

$D_{min} = minimum(Arr[0][hash_0].D_{min}, Arr[1][hash_1].D_{min}, \ldots, Arr[v-1][hash_{v-1}].D_{min})$;

$if(D_{min} == Busy)$

Limit the transmission window size for the misbehaving flow;

$else$

Drops/Marks the arriving packet with D_{min} value;

}

Рисунок 2.10 – Алгоритм SFB

Псевдокод алгоритму SFB та приклад, який розрізняє хороші та недоброзичливі потоки, продемонстровано на рисунку 2.10. Параметр q_{size} представляє кількість пакетів, відображених у певному бункері, як P_{inc}, так і параметр P_{dec} схожі з параметрами BLUE.

Крім того, D_p для потоків, що поводяться погано, у SFB швидко змінюється на 1, тоді як, D_p для потоків, що поводяться коректно, обчислюється аналогічно BLUE. Алгоритм SFB приймає рішення про викидання пакетів на основі D_{min}, що відповідає мінімальному значенню D_p для всіх бункерів, на які відображається певний потік.

У випадках, коли D_{min} менший за 1, SFB ідентифікує поведінку потока і викидає пакети, що надходять, в межах цього D_{min}. В іншому випадку SFB ідентифікує потік як неправильно діючий і зменшує розмір вікна передачі для цього потока, скорочуючи кількість мережевих ресурсів, які він використовує.

3 FUZZY LOGIC АЛГОРИТМ

3.1 Fuzzy Logic

Теорія нечіткої логіки була запропонована професором L.A. Zadeh в 1965 році. Ідея про нечітку логіку насправді починається тоді, коли виникає думка або вимога, щоб комп’ютер міркував, використовуючи наближення, не засноване на точному введенні. Комп’ютер дуже точний, і він не може зробити наближення до ситуації, якщо йому не сказано так і не забезпечено нечіткою логікою. У реальному світі існує багато речей, які потребують евристичного судження відповідно до ситуації, а не за допомогою попередньо запланованих або перепрограмових значень. Іншими словами, нечітка логіка – це спосіб збору людського інтелекту, знань, досвіду, мисленнєвого процесу та судження.

Нечіткий набір – це набір, який використовується для групування в ньому об’єктів, але без чітких меж. Набір може мати групу речей, яка саме належить йому, і групу речей, яка йому не належить. Саме слово нечітке говорить про те, що воно незрозуміле або розмите.

Наприклад, можна використовувати набір голодних для опису групи голодних людей. За логікою група розділилася б на два набори значень: вони голодні, чи ні. Ці два згадувані тут значення є голодні та не голодні. Це дуже точний або чіткий метод групування. Якщо використати тривалість часу після останнього прийому їжі, щоб визначити, голодна людина чи ні, можна припустити, що через 6 годин після останнього прийому їжі людина зголоднієє. Тож людина, яка востаннє вживала їжу 4 години або 5 годин тому, вважається не голодною. Можливо також зробити висновок, що людина, яка щойно приймала останній прийом їжі 5 годин 59 хвилин тому, не голодна. Просто через хвилину різниці, висновок трохи надто драматичний. На рисунку 3.1 показана ситуація цього прикладу.

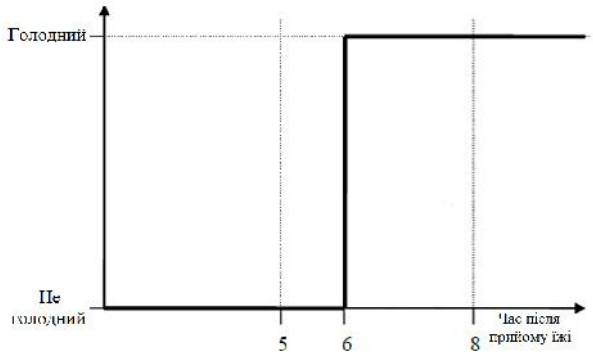

Рисунок 3.1 – Чітке подання терміну голодний

Ці кардинальні зміни у рішенні можна було б виправити, використовуючи ступінь, яка представляє, наскільки голодна людина. Значення між 0 і 1 буде використано для вимірювання цього ступеня. Використовуючи цей діапазон ступенів, можна обґрунтувати, що людина, яка востаннє їла 6 годин тому, вважається голодною із ступенем 0,7, 5 годин тому зі ступенем 0,5 та 8 годин тому дуже голодний зі значенням 1,0.

Крім того, щоб мати ступінь 0, щоб показати, що людина не голодна, і 1, щоб показати, що людина справді голодна, можна мати більше ніж один нечіткий набір того, як голодна людина. Щоб далі пояснити цю ситуацію 3 нечіткі набори будуть визначені для голодних, які не голодні, відносно голодні та надзвичайно голодні. На рисунку 3.2 представлені 3 ситуації.

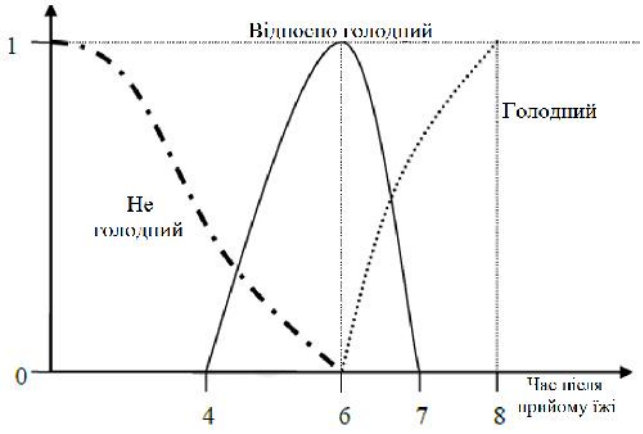

Рисунок 3.2 – Нечітке представлення терміну «голодний»

Кожна крива на графіку називається функціями членства. У цьому прикладі є лише 3 функції членства. Можна визначити більше 3 функцій членства. Наприклад, може бути додана функція голодного членства і крива може бути проведена між 5 та 7 годинною лінією.

Нечіткі множини можна розглядати як розширення булевої логіки, тому класичні набори оператора можуть бути виконані на нечітких множинах. Класичні операції набору наступні:

- союз;
- перехрестя;
- доповнення.

Союз – це операція, яка вибирає максимум ступенів членства з заданих наборів. Наприклад, якщо задати множини A і B. з обома максимальними ступенями членства 0,9 та 1,0 відповідно, оператор об’єднання вибере значення 1,0.

Перетин – це операція, яка приймає мінімум ступенів членства від заданих наборів. Наприклад, задані множини G і H. з обома мінімальними ступенями членства складають 0,8 і 0,2 від кожного набору; оператор перехрестя вибере 0,2.

Доповнення – це оператор, який приймає ступінь членства будь-яких заданих членів у наборі і мінусує його з 1. Наприклад, для заданого члена в наборі, зі ступенем приналежності 0,45, оператор комплементу поверне значення 1-0,45, що становить 0,55.

Обґрунтування – це процес, який можна описати, використовуючи звичайний жаргон як тлумачення. Нечіткі міркування або приблизні міркування – це спосіб отримати вихід з декількох входів або технічно відомий як нечіткі умовиводи. Цей метод можна пояснити як спосіб інтерпретації людини відповідно до ситуації за кількома обставинами, з різницею, комп’ютер робить це.

Нечіткі умовиводи – це такий тип логіки, який може бути описаний математикою. Тут основні речі – це правила, факти та висновки. Висновок

можна вкласти в математичне формулювання, як показано нижче.

Rule: IF x is A THEN y is B.

Fact: x is A.

Conclusion: y is B.

Для отримання результатів висновку необхідно визначити набір правил. Розмивання – це метод встановлення факту на основі нечіткої системи.

Композиція – це процес висновку, який враховує кожен вихід. Дефузіфікація – це завершальний процес, коли значення нечіткого висновку з єдиним висновком буде відображено у чітке значення. Це чітке значення необхідно для основної системи або користувача нечіткої системи, оскільки система або користувач може зрозуміти чітке значення або діяти, використовуючи чітке значення.

3.2 Пропонована модель активного управління чергою

Виникає необхідність ініціювати більш підходящий спосіб або більш динамічний метод задоволення поточної тенденції мережевого трафіку. Запропоноване активне управління чергою (FuzAQM) є одним із кроків, зроблених для вирішення цього завдання.

Запропонована модель буде використовувати нечітку логіку як інструмент для перетворення традиційної моделі управління чергою в динамічну модель. Є два параметри, які будуть використані для реалізації цієї моделі. Два параметри – це RED пороги та ймовірність відкидання. Ідея використання цих двох параметрів полягає в тому, що розмір черги та швидкість зміни черги пов’язані один з одним.

Ця запропонована модель FuzAQM використовує поточну зайнятість черги та швидкість зміни черги в кожен момент, коли пакет прибуває до маршрутизаторів, як центрів вузьких місць для визначення подальших дій.

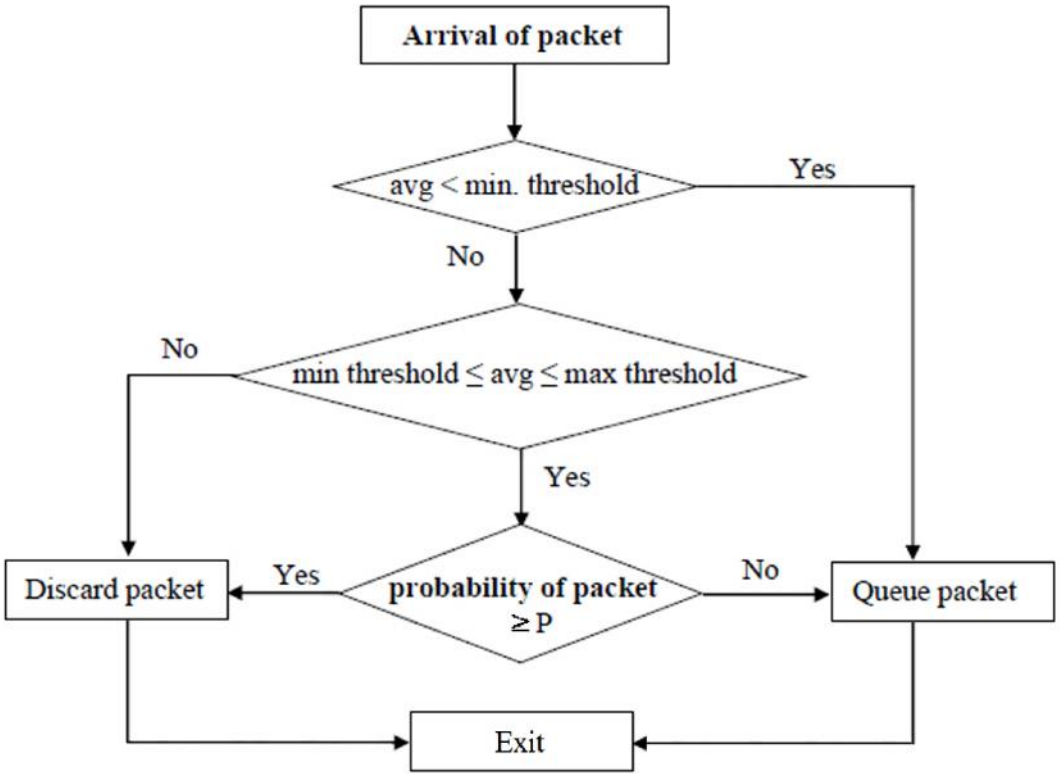

Рисунок 3.3 – Алгоритм розміщення пакету в черзі RED

3.2.1 Нечіткі реалізації логіки

Нечіткі логічні коди вставляються у функцію enqueue(), що знаходиться у файлі dsredq.cc. Код fuzzy_result=fuzzy(q_->length()-prev_20.qlim) буде використовуватися для виклику нечіткого логічного коду.

Перед тим, як викликати fuzzy() функцію, ряд обчислень потрібен, щоб забезпечити вхід для функції. Основна проблема – чітко вказати, в якому напрямку трафіку потрібно обчислити. Це потрібно, оскільки функція enqueue() – це загальна функція, яка використовується для встановлення черги пакетів у фізичній черзі, і вона диференціюється з напрямком пакету. Дія, яку слід здійснити в цілому, багато в чому залежить від того, яку частину симплексного посилання використовуватимуть як обчислення.

Диференціація може бути засвідчена пунктом "якщо і ще", який виглядає так:

```
      if (src_id == 14 || src_id = 15 || src_id = 16 || src_id == 17 || src_id
== 22 || src_id == 23 || src_id == 26 || src_id = 27 || src_id == 30 ||src_id
== 31 ||src_id == 34 || src_id == 35)
      {
      …….
      …….
      }
      else {
      …….
      …….
      }
```

Приклад 3.1 – Диференціація

Перша частина стосуватиметься маршрутизатора 6 до маршрутизатора 7 (зліва направо), а друга стосується трафіку, який рухається від маршрутизатора 7 до маршрутизатора 6 (справа наліво). На рисунку 3.4 нижче показано загальне подання вищезазначених напрямків.

q_->length() – це функція, яка повертає фізичну довжину черги для черги q_. У конструкторі використовується q_=newPacketQueue() для створення вказівника на чергу пакетів або на основну фізичну чергу.

r6prev_20 – змінна float типу. Використовується для зберігання попередніх 20 фізичних розмірів черги кожного разу, коли в фізичній черзі є черговий пакет. 20 значень вибирається для того, щоб зробити середній розрахунок більш релевантним, а не лише 5 або 10.

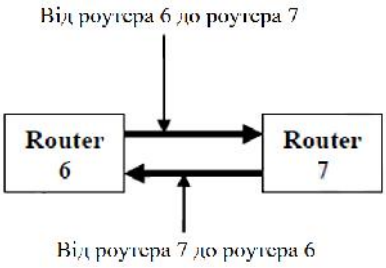

Рисунок 3.4 – Загальне представлення використовуваної мережевої топології

Кожен раз, коли пакет буде в черзі, довжина фізичної довжини черги буде відрізнятися. Час між останнім успішним пакетом у черзі та поточним пакетом у черзі може свідчити про те, що існуючі пакети фізичної черги

видаляються з черги чи вона спорожняються. Крім того, існує багато пакетів, які стоять у черзі та виводяться з черги, тому середнє значення довжини черги має бути обчислене з використанням максимуму попередніх 20 значень. Індикатор r6 посилається на чергу, яка відповідає за пересилання пакетів від маршрутизатора 6 до маршрутизатора 7. Функція для обчислення середнього значення попередніх 20 довжин черги знаходиться у функції enqueue(). Середнє значення попередніх 20 черг наведено нижче:

```
float r6prev_20;
int counter=l;
r6_travel=new node;
r6_travel=r6_cur;
if (r6_travel!=NULL) {
  while((r6_travel->next!=NULL) && (counter!=20))
  {
    j =j +r6_travel->num;
    r6_travel=r6_travel->next; counter++:
  }
  r6prev_20=j/counter;
} else
  {
    r6prev_20=0;
  }
```

Приклад 3.2 – Обчислення середньої довжини черги

Спочатку створюється вказівник на вузол і називається як r6_travel. Він використовується для пересування по списку посилань, щоб зібрати до 20 довжин черги з останнього доданого вузла. Слід зазначити, що він не враховує вузол, який буде доданий до списку посилань в кінці функції enqueue(). Це тому, що попередні 20 обчислень фізичної довжини черги не повинні враховувати новий вузол.

Лічильник використовується для відстеження кількості пройдених вузлів за вказівником r6_travel. Він матиме принаймні значення 1. Метою мати принаймні значення 1 є уникнути ділення на 0.

Лічильник r6_travel буде вказувати на вузол, на який вказує вказівник r6_cur. Мета використання if(r6_travel!=NULL) – переконатися в наявності списку посилань. На самому початку моделювання, у списку посилань не

буде вузла. На це вказують вказівник головного вузла та поточний покажчик вузла, рівний нулю. Коли функція enqueue() буде виконана вперше, покажчик r6_travel буде вказувати на покажчик r6_cur. Це означає, що r6_travel буде непрямо вказувати на нульовий вузол. У цьому випадку змінна r6prev_20 негайно буде мати значення 0 за допомогою призначення r6prev_20=0, розташованого в іншому пункті, як показано вище.

Якщо вищезазначений випадок не відповідає дійсності, цикл while ((r6_travel->next!=NULL)&&(counter!=20) буде виконаний. Перший аргумент r6_travel->next!=NULL – переконатися, що вузол, на який вказує r6_travel, не є першим вузлом у списку посилань. Перший вузол списку посилань називається head (куди вказує вказівник r6_head). Другий аргумент (counter!=20) призначений, щоб переконатися, що в списку посилань існувало більше 20 вузлів, середнє обчислення враховує лише останнє значення довжини черги 20 вузлів. Якщо будь-яке з цих правил виконано, цикл while закінчиться. Для трафіку, який рухається від маршрутизатора 7 до маршрутизатора 6, буде використовуватися однотипні код і процедура, за винятком того, що цього разу назва змінної буде іншою (буде ідентифікована ідентифікатором r7 перед змінною).

3.2.2 Нечіткі набори та функції членства

Коли обчислюються середні попередні 20 довжин черги, тоді нечітка логічна функція може бути викликана redQueue::fuzzy(int q_len, float avg, int q_lim). Вона отримає три аргументи від функції виклику. Перший аргумент буде використаний для представлення довжини черги, другий аргумент буде представляти середню чергу, а останній буде представляти межу черги, кожен з відповідним типом змінної.

float q_rate=q_len-avg буде використовуватися для обчислення швидкості зміни черги. Швидкість зміни – це фактично різниця між поточною довжиною черги та середнім показником попередньої довжини 20

черг. Значенням цієї різниці буде додатне значення, нульове або негативне значення. Максимальне значення різниці складе 500, а мінімальне -500. Це мінімальне значення виникне, якщо поточна довжина черги дорівнює нулю, а середня 500. Максимальне значення виникне, коли поточна довжина черги становить 500, а середня довжина черги 0.

У нечіткому логічному коді є дві функції членства. Перша функція членства – це поточна довжина черги, а друга – швидкість зміни черги.

3.2.3 Поточна довжина черги Fuzzy Set

Поточна функція членства в довжині черги буде представлена масивом myLen[2][7] float типу. Розмір цього масиву – 2, він містить 2 рядки та 7 стовпців, як показано нижче:

{1.0, 0.0, 0.0, q_lim/2, q_lim,0.0,0.0},
{0.0, q_lim/2, q_lim, q_lim, 0.0,0.0,0.0}

Обидва ряди мають загальний тип стовпців. Перший стовпець представляє форму першого графа у функції членства. 1,0 буде представляти графік з формою трапеції, а 0.0 представлятиме графік з формою трикутника.

Наступні 6 стовпців будуть використані для представлення 6 різних точок або областей на графіку. Значення, показане в кожному стовпці, буде представляти x-координати на графіку. Відповідні y-координати були встановлені у значенні {X.X 0, 1, 1, 1, 0, 0, 0}, де X.X представляють значення, що представляють форму трапеції (що буде 1,0). Для форми трикутника значення XX дорівнює 0,0, а відповідні координати у встановлено на значення {XX, 0, 1, 0, 0, 0, 0} і останнє значення стовпця залишається невикористаним.

Стовпці 6 та 7 масиву завжди будуть представляти два крайніх кінця графіка. У стовпці 6 буде представлено негативне значення нескінченності

або мінімальне найбільше значення x-координати, а стовпець 7 буде представляти позитивне значення нескінченності або максимальне найбільше значення x-координати для графіка трапеції. Для форми трикутника графік 5-го та 6-го стовпців буде використаний для представлення крайньої лівої та правої частини графіка, де лінії тягнуться до значення нескінченності осі x.

3.2.4 Трикутний графік

Графік трикутника зазвичай матиме форму, як показано на рисунку 3.5 нижче. Приймаючи вище наведені параметри трикутника графіка. {0,0, q_lim / 2, q_lim, q_lim. 0,0, 0,0, 0,0}, де q_lim = 500. Форму графіка зображено на рисунку 3.6.

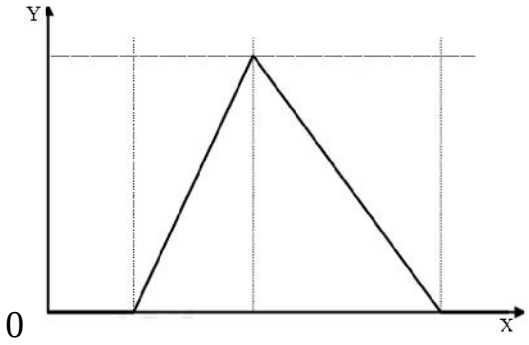

Рисунок 3.5 – Трикутний графік

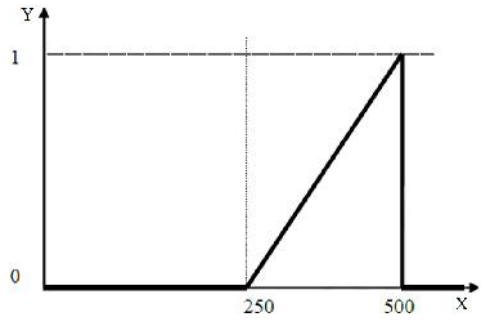

Рисунок 3.6 – Поточна довжина черги першого масиву з використанням трикутника

3.2.5 Графік трапеції

Графік трапеції, як правило, має форму, як показано на рисунку 3.7.

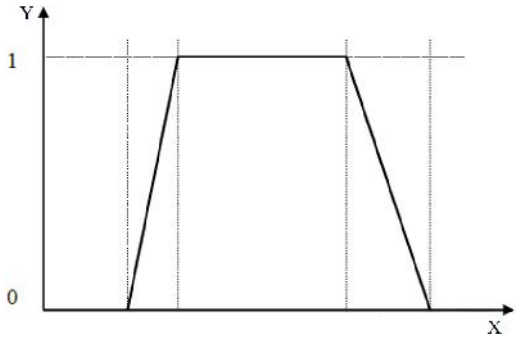

Рисунок 3.7 – Графік трапеції

Враховуючи наведені вище параметри графіка трапеції {1.0,0.0,0.0,q_lim/2,q_lim,0.0,0.0}, де q_lim=500. Зображення графіка показано на рисунку 3.8.

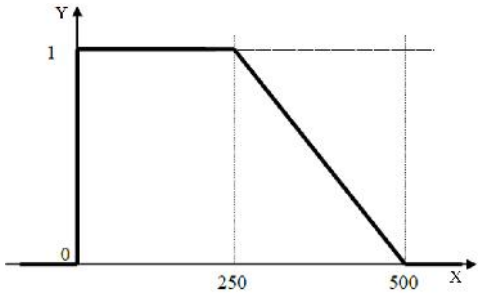

Рисунок 3.8 – Поточна довжина черги другого массиву з використанням графіка трапеції

3.2.6 Графік функції поточної довжини черги

Щоб отримати уявлення про весь поточний графік функції членства в довжині черги, два існуючі графіки об’єднані. Об’єднаний графік показано на рисунку 3.9.

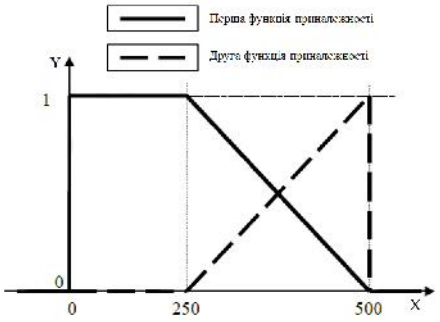

Рисунок 3.9 – Функції поточної довжини черги

3.2.7 Діючі правила довжини черги

Функція поточної довжини черги керується набором правил. Довжина черги розділена на дві зони: "повна" та "не повна". Перша функція приналежності стосується функції, яка представляє не повну чергу, а друга – повну.

Для функції ’не повна’, вихід буде 0, якщо вхід потрапить у зону до 0 або зону після 500 осі x. Вихід буде 1 істина, коли черга порожня або поки вона не буде заповнена на 50%. Після порогового рівня 50% функція стане лінійним рівнянням, результат якого знижується з 1 до 0 лінійно. Лінійна функція, яка використовується для нечіткості, наведена нижче:

preoutl[i]=(myLen[i][4]-q_len)/(myLen[i][4]-myLen[i][3]).

Для функції "повна" вихід буде 0, якщо значення потрапляє в зону до 250 або в зону після 500 осі x. Вихід буде 1 або істина, коли черга становить 100%. Перед 50-відсотковою межею емності черги, функція буде 0, щоб вказати, що черга зовсім не заповнена. Після порогу 50% функція лінійно зростає від 0 до 1. Лінійна функція, яка використовується для нечіткості є такою:

preout1[i]=(q_len-myLen[i][1])/(myLen[i][2]-myLen[i][1])

3.2.8 Швидкість зміни чергового нечіткого набору

Швидкість зміни функції принадлежності в черзі представлена масивом myRate[4][6] типу double. Розмір цього масиву – 2, він містить 4 рядки та 6 стовпців, як показано:

{-q_lim, -q_lim, 20-q_lim, 0.0, 0.0, 0.0},
{-q_lim/2, -10.0, 0.0, 0.0, 0.0, 0.0},
{0.0, 0.0, 10.0, q_lim/2, 0.0, 0.0},
{0.0, q_lim-20, q_lim, q_lim, 0.0, 0.0}

Оскільки масив у myRate використовується для представлення лише графіка трапеції, немає необхідності мати перший стовпець для визначення форми графіка, як у функції членства в поточній черзі.

Усі рядки мають загальний тип стовпців, які представляють 6 різних точок або областей на графіку. Значення, показане в кожному стовпці, представляє x-координати на графіку. Відповідні y-координати встановлені на значення {0,1,1,0, 0,0}.

3.2.9 Правила швидкості зміни черги

Побудувавши все значення масиву myRate на одному графіку, співвідношення між усіма функціями можна побачити, як показано на рисунку 3.10

Швидкість зміни функції довжини черги регулюється набором правил. Швидкість зміни черги розділена на чотири зони, які швидко зменшуються, зменшуються, швидко збільшуються та збільшуються. Перша функція членства буде посилатися на функцію, яка представляє чергу, що зменшується швидко, а друга – посилається на швидкість зміни. З іншого боку, функція членства 3 буде використовуватися для позначення швидкості

зменьшення, тоді як функція 4 буде використовуватися для позначення швидкості збільшення.

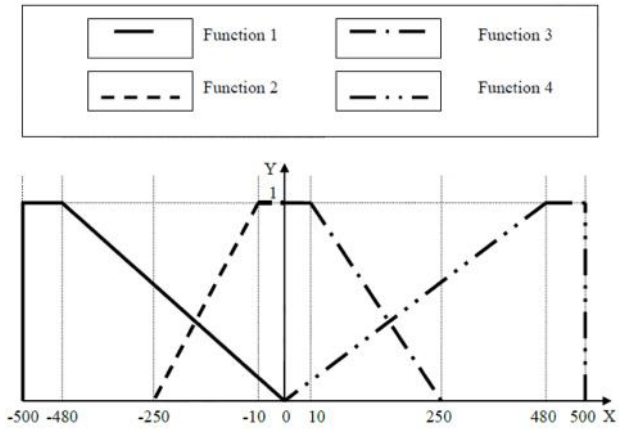

Рисунок 3.10 – Функції швидкості зміни довжини черги

Для функції 1. Кажуть, що швидкість зменшується швидко (дорівнює 1), якщо різниця між поточною довжиною черги та середньою довжиною становить від -500 до -480 пакетів. Швидкість буде лінійно зменшуватися між -480 та 0 пакетами. Ступінь зниження швидкості стане неправдивою, коли вона досягне 0 в різниці пакетів.

Для функції 2 кажуть, що швидкість зменшується (дорівнює 1), якщо різниця між поточною довжиною черги та середньою довжиною становить від -10 до 0 пакетів. І швидкість буде лінійно зростати між -250 та -10 пакетами. Ступінь зменшення стане неправдивою, коли вона досягне -250 пакетних різниць. Причина, чому ступінь зменшення зменшується, полягає в тому, що черга зменшується швидко.

Для функції 3 кажуть, що швидкість збільшується (дорівнює 1), якщо різниця між поточною довжиною черги та середньою довжиною становить від 0 до 10 пакетів. І швидкість буде лінійно зменшуватися між 10 та 250 пакетами. Ступінь збільшення стане неправдивою, коли різниця досягне 250 пакетів.Причина, по якій ступінь збільшення збільшується, полягає в тому, що черга входить у зону зростаючої швидкості.

Для функції 4 кажуть, що швидкість зростає швидко (дорівнює 1), якщо різниця між поточною довжиною черги та середньою довжиною становить від 480 до 500 пакетів. І швидкість лінійно зростає між 0 і 480 пакетами. Ступінь швидкого зростання стане неправдивою, якщо буде 0 пакетів різниці.

3.2.10 Нечіткий логічний вивід

Весь процес можна розділити на дві частини, отримання нечіткої інформації на вході та застосування нечіткого оператора, а також виведення результату функції приналежності.

Модель нечіткого логічного виводу має типове правило, яке має наступний вигляд: якщо на Вході *X* та на Вході *Y*, то Вихід дорівнює *Z*, де *X* і *Y* представляють перший і другий вхід.

Вихід *Z* кожного правила зважено на вагове значення *w*, відповідне цьому правилу. Значення ваги заздалегідь визначене відповідно до ситуації вузького міста мережі. Використовуються два загальних нечітких оператора, але тут використано лише один оператор AND. Оператор AND виберає найменше значення двох виходів.

```
for (i=0;i<8:i++)
{
   totalY+=myWeight[i]*foutputHeight[i];
   totalH+=foutputHeight[i];
}
if (totalH>0)
  finalOutput=totalY/totalH;
else
  final0utput=0;
  return finalOutput;
}
```

Приклад 3.3 – Реалізація логічного висновка

Кінцевий вихід обчислюється за показаним вище блоком коду, і значення повертається до функції виклику. Змінна finalOutput

використовується для зберігання результату обчислення. Результатом буде сума всіх результатів правил виводу (w_i,z_i) усереднена сумою всієї вихідної чисельності (z_i). Підсумковий розрахунок в може бути представлений формулою (3.1):

$$fuzzy\_result = \frac{\sum_{i=1}^{N} w_i z_i}{\sum_{i=1}^{N} z_i}, \qquad (3.1)$$

Нечітка частина виводу будується на правилі:

- якщо Вхід *X* та Вхід *Y*, то Вихід дорівнює *Z*,=.

Вхід X посилається на поточну функцію довжини черги, тоді як вхід Y посилається на швидкість зміни черги, порядок роботи нечіткого виводу, набір ваг оголошено у нечіткій функції. Вага, як показано нижче float myWeight[8]={1,2,3,4,5,6,7,8}.

Всього використовується 8 ваг, тому що існує 8 можливих результатів у процесі виводу. Вага вибирається таким чином, щоб відображати ступінь критичності в «bottle neck» черзі. Найбільш критичній ситуації буде відповідати вага 8, а найменш критичній – вага 1. Командний рядок, представлений нижче використовується для оголошення масиву зберігання результатів виводу, перш ніж він буде використаний для обчислення кінцевого виводу:

float preoutl[2]={0},
float preout2[4]={0}.

float foutputHeight[8]={0} застосовується для збереження результату після застосування оператора нечіткого AND.

Весь процес починається, коли представлені два параметри. Перший параметр – поточна довжина черги. Поточна довжина черги буде

використовуватися для пошуку результату функції поточної довжини черги. Обидва графіки функції використовуються для отримання результату.

А другий вхід, швидкість зміни черги, буде використаний для з’ясування відповідного виводу. Кожен вихід зважено відповідно до вагового масиву. Наприклад, якщо поточна довжина черги становить 100 і швидкість зміни становить -100 пакетів, вихід буде таким, як показано. Число крім фігури являє собою номер правила.

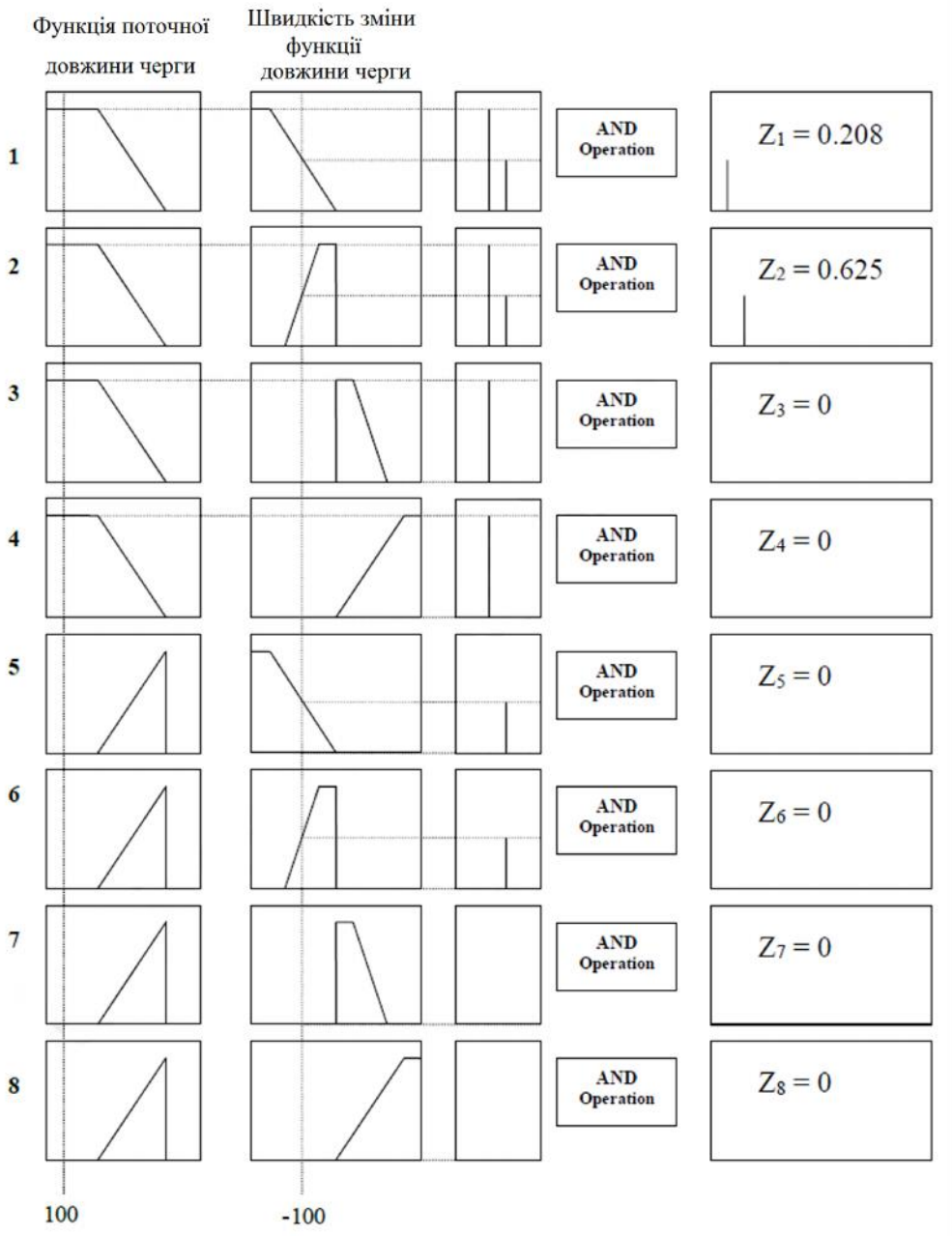

Рисунок 3.11 – Процесс нечіткого виводу

Кожне правило зважено таким чином відповідно до його ваги у масиві (приклад: Правило 1 буде зважене myWeight[l]=l и Rule 8 буде зважено myWeight[S]=8). Для отримання остаточного необхідного результату результати висновку необхідно дефазіфікувати. Процес дефазіфікації для наведеного вище прикладу буде розраховано за формулою 3.1 наступним чином:

$$finalOutput = \frac{(1 \cdot 0.208) + (2 \cdot 0.625) + (3 \cdot 0) + (4 \cdot 0) + ... + (8 \cdot 0)}{0.208 + 0.625} = 1.75 .$$

Значення кінцевого виходу говорить про те, що перевантаженість не є серйозною. У результаті пакет може бути в черзі. Подальші дії будуть вжиті відповідно залежно від значення кінцевої вихідної величини.

У наступних прикладах будуть враховані два випадки, перший – випадок низької перевантаженості, а другий – випадок великого затору.

Для випадку низької перевантаженості значеннями, які будуть використані для обчислення, є поточна довжина черги в 10 пакетів та швидкість зміни черги -480 пакетів. За допомогою цих значень можна зробити висновок, що черга заповнена лише на 2%, а швидкість зміни зменшується.

Для серйозного випадку перевантаженості значеннями, які використані для обчислення, є поточна довжина черги в 480 пакетів та швидкість зміни черги 480 пакетів. За допомогою цих значень можна зробити висновок, що черга заповнена на 96%, и черга дуже швидко збільшується в розмірах.

На наступних двох діаграмах показані нечіткі процеси виводу відповідних результатыв обох ситуацій. Розрахунок дефазіфікації для сценарію з низьким перевантаженням показано у формулі 3.2:

$$finalOutput = \frac{(1 \cdot 1.0) + (2 \cdot 0) + (3 \cdot 0) + (4 \cdot 0) + ... + (8 \cdot 0)}{1.0} = 1.00 . \qquad (3.2)$$

Розрахунок дефізифікації для сценарію серйозного перевантаженя показано у формулы (3.3).

$$finalOutput = \frac{... + (3 \cdot 0) + (4 \cdot 0.08) + (5 \cdot 0) + ... + (7 \cdot 0) + (8 \cdot 0.92)}{0.08 + 0.92} = 7.68. \quad (3.3)$$

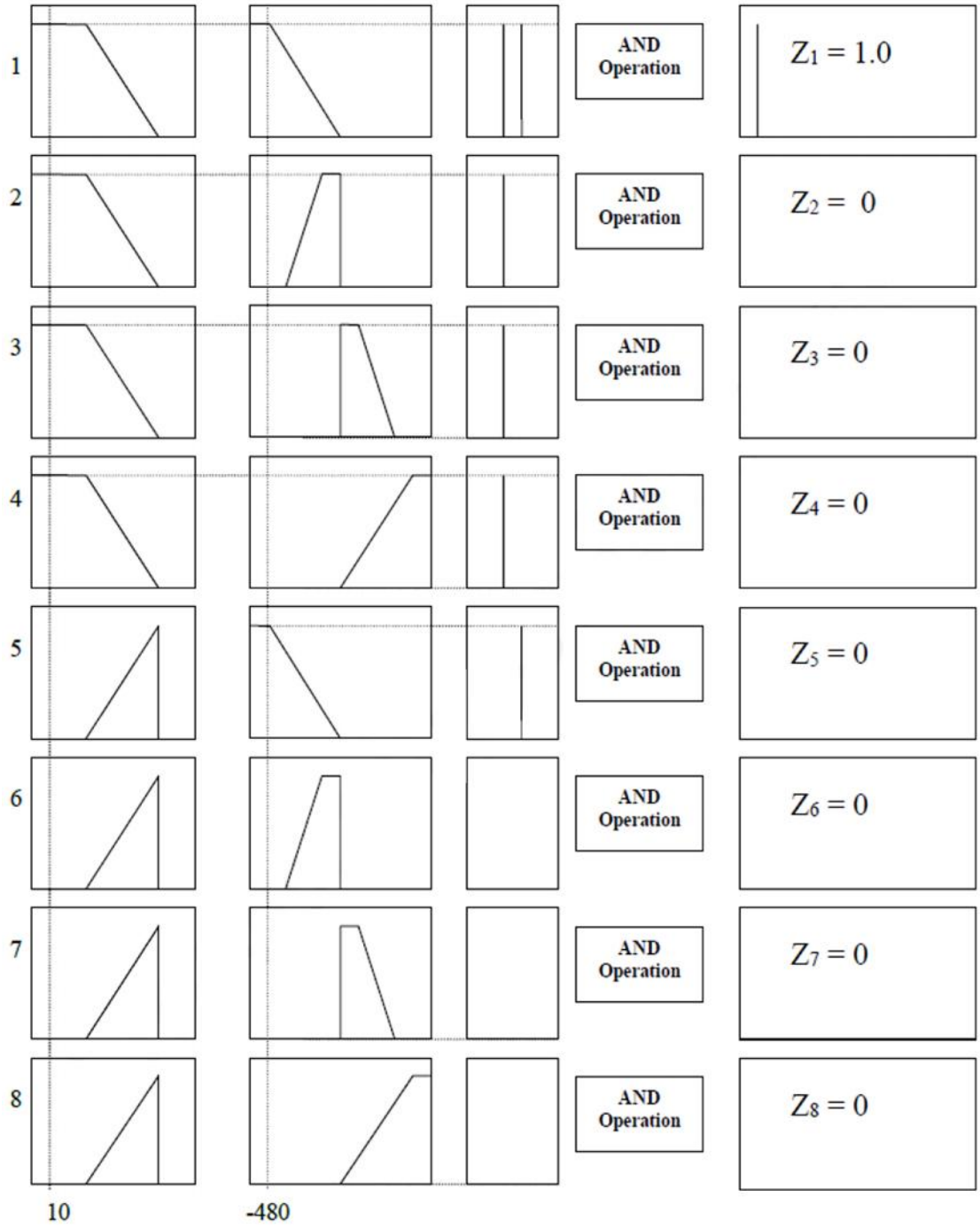

Рисунок 3.12 – Процеси нечыткого виводу з низьким перевантаженням

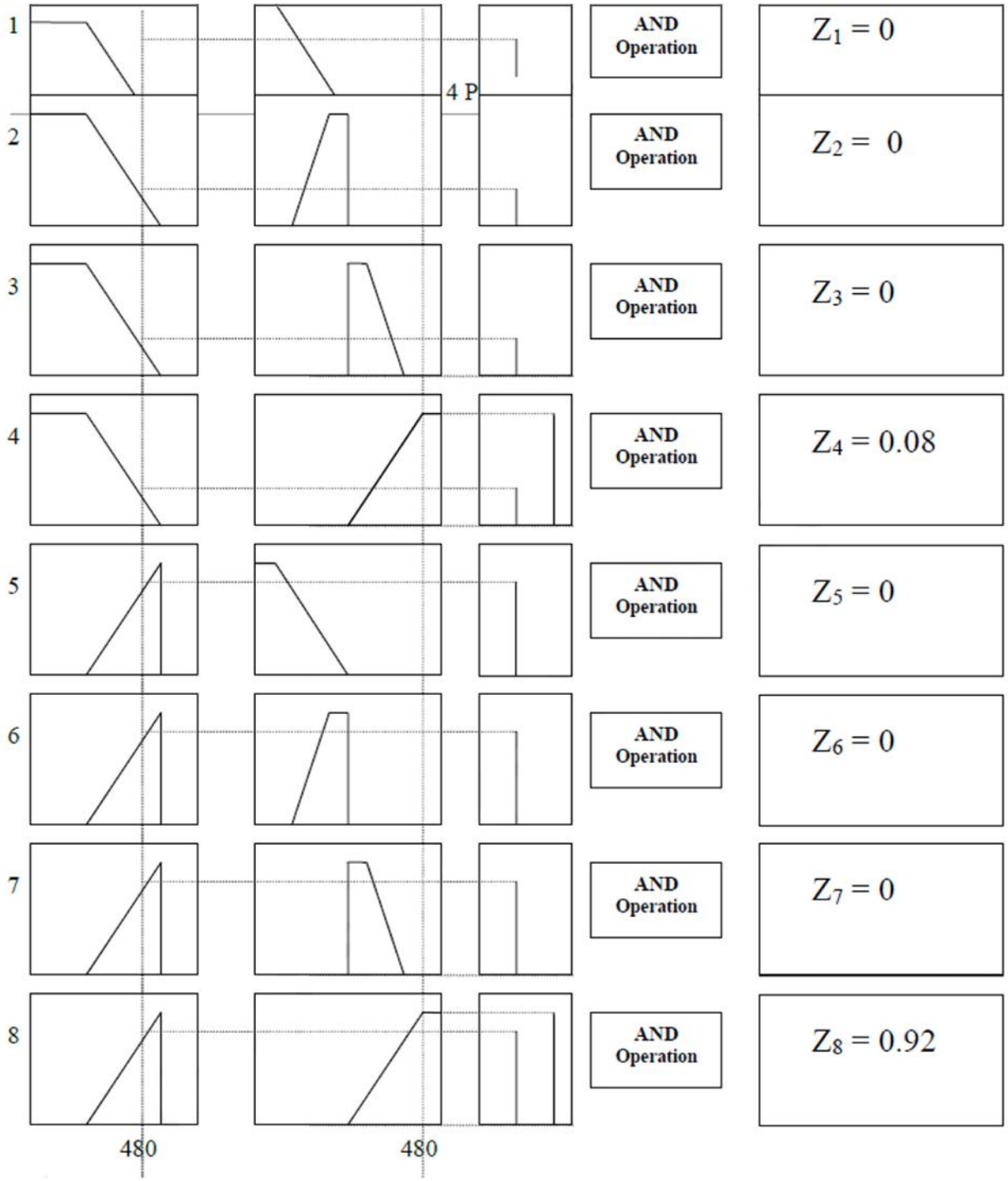

Рисунок 3.13 – Процеси нечыткого виводу з вагомим перевантаженням

Після обчислення розмитих результатів нечітка логічна функція поверне значення float типу функції, яка її викликала. Функція, яка називається нечіткою логічною функцією, є fuzzy_result=fuzzy(q_->length(),prev_5,qlim).

Нечітке значення тепер буде збережено у змінній fuzzy_result. Це значення буде оцінено для здійснення відповідних дій. Перший крок – це розподілити значення на 9 категорій серйозності вузького місця (переповненої черги). Використовуване значення від 1 до 9, де 9 використовується для представлення найбільш серйозного перевантаження черги. Для того, щоб класифікувати нечіткий результат застосовується блоковий код:

```
if (fuzzy_result>=7.5)
fuzzy_result=9.0;
else if (fuzzy_result>=7)
fuzzy_result=8.0:
else if (fuzzy_result>=6)
fuzzy_result=7.0;
else if (fuzzy_result>=5)
fuzzy_result=6.0;
else if (fuzzy_result>=4)
fuzzy_result=5.0;
else if (fuzzy_result>=3)
fuzzy_result=4.0;
else if (fuzzy_result>=2)
fuzzy_result=3.0;
else if (fuzzy_result>=l)
fuzzy_result=2.0;
else if (fuzzy_result>=0)
fuzzy_result=l .0;
```

Приклад 3.4 – Класифікація нечіткого результату

Будь-який нечіткий результат, повернутий зі значенням 7,5 і вище, класифікується як 9,0 (дуже серйозний стан). Якщо значення менше 7,5, дорівнює 7 або вище, класифікується як 8,0. Цей процес продовжується до значення класу 1,0.

Виходячи з цих 9 ситуацій або ступеня серйозності, вживаються відповідні дії. Для кожного ступеня призначається набір RED порогів та ймовірності відкидання пакетів.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА МОДЕЛЮВАННЯ

4.1 Модельний сценарій

Сценарій моделювання написаний мовою tcl. Основним файлом для моделювання є файл diffnet.tcl. При запуску моделювання, він буде викликати інші файли таким чином:

- peer_setup.tcl;
- 2q2p.tcl;
- topology.tcl;
- monitoring.tcl.

Файл peer_setup.tcl складається з сценарію tcl, який створює HTTP. FTP та VoIP-трафік. Файл 2q2p.tcl складається із сценарію tcl, який встановлює всі специфічні процедури. Файл topology.tcl складається із скрипту tcl, який створює топологію. Файл monitoring.tcl використовується для завантаження функцій моніторингу для черг та потоків трафіку.

Команда запуску викличе процедуру launchSession у файлі peer_setup.tcl, щоб почати створювати та генерувати веб-трафік. Команда launchHttp httpCl http S4 використовується для запуску трафіку HTTP, починаючи з того, що http-клієнт C1 видає http-запит на http-сервер S4. Команда launchFtp ftpCl ftp S4 використовується для запуску FTP-трафіку, починаючи з клієнта ftp C1, видаючи ftp-запит на ftp-сервер S4. launchVoip voipl voip5 0 використовується для запуску VoIP-трафіку між VoIP 1 та VoIP 5 вузлом з присвоєним id 0.

Значення CIR використовується для визначення допущеної швидкості передачі для кожної програми. Значення CIR, вибраЄТЬСЯ для кожного трафіку різне, воно вказане нижче:

- set cir_exp 50000 (to set the VoIP CIR rate);
- set cir_http 70000 (to set the HTTP CIR rate);

- set cir_ftp 150000 (to set the FTP CIR rate).

Команда confDSCore використовується для налаштування бажаної черги та параметрів політики для двох вузлів горловини пляшки. Деталі того, як або які параметри потрібно встановити, розміщені у файлі 2q2p.tcl. Команда confDSCore r6 r7 використовується для встановлення параметрів потоку трафіку від маршрутизатора 6 до маршрутизатора 7. Потік трафіку від маршрутизатора 7 до маршрутизатора 6 виконується за допомогою іншої команди, оскільки зв’язки між двома вузькими маршрутизаторами є простою ланкою. Симплексне з’єднання може передавати пакети лише в одному напрямку.

Команда confDSEdges використовується для встановлення бажаних параметрів черги та політики для пари з’єднання. Наприклад confDSEdges voipl voip5 $cir_exp 29_app AF налаштує швидкість CIR з’єднання або трафік між значенням, встановленим у наборі cir_exp. 29_app – це код для встановлення ID, який визначає, що трафік з номером 29 – це VoIP-трафік, а початковий тип послуги, який він вимагає AF. 27_app позначає FTP-трафік, тоді як 31_app – трафік HTTP.

$ns 50 "timeStats 50" буде виводити статистику моделювання на екрані кожні 50 інтервалів симуляційних секунд. По закінченні моделювання команда $ns в $testTime "закрити-з’єднання" спробує закрити пару з’єднань, надіславши підтвердження закінчення маршрутизатору пари з’єднання. $ns в [expr StestTime + 5] "kill-em-all" змусить усі інші з’єднання закритись, [expr $testTime + 5] означає, що після закінчення часу моделювання він змусить усі залишки з’єднань закрити після ще 5 симуляційних секунд.

$testTime+11, $ns в [expr $testTime+11] "закінчити" викликає процедуру завершення. "Процедура обробки" завершить все моделювання. Процедура приведена нижче:

```
proc finish {} {
global ns alku trace_on pf q
$ns flush-trace
$q(r6r7) printStats
$q(r7r6) printStats
flush $pf close $pf
exit 0
}
```

Приклад 4.1 –Процедура завершення

Вона збереже всю кінцеву статистику черг моделювання в секундах та закінчить процес моделювання.

4.1.1 Топологія мережі

Топологія створюється у файлі topology.tcl. Моделювання використовує топологію гантелі з 14 маршрутизаторів і різним числом джерел в залежності від трафіку, який повинен бути згенерований. Для моделювання мережі з «легким» трафіком кількість вузлів обмежена 24. Для «важкого» трафіку кількість джерел збільшується до 48 вузлів.

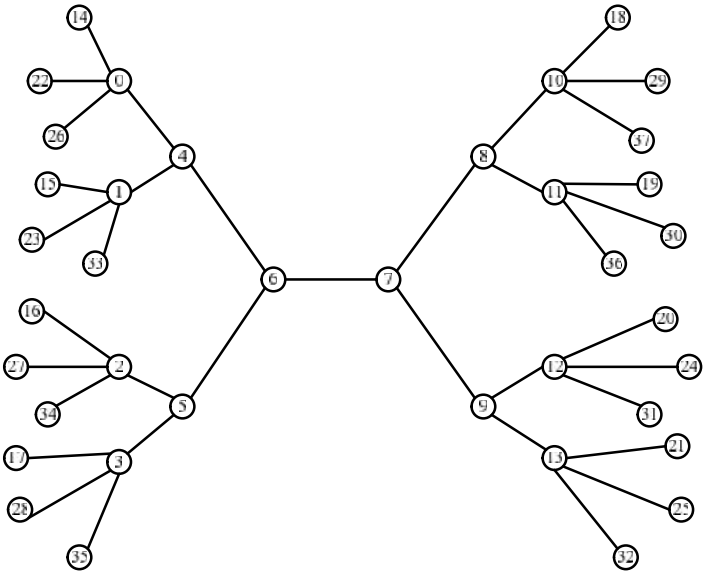

Рисунок 4.1 – Топологія мережі для випадку «легкого» трафіку

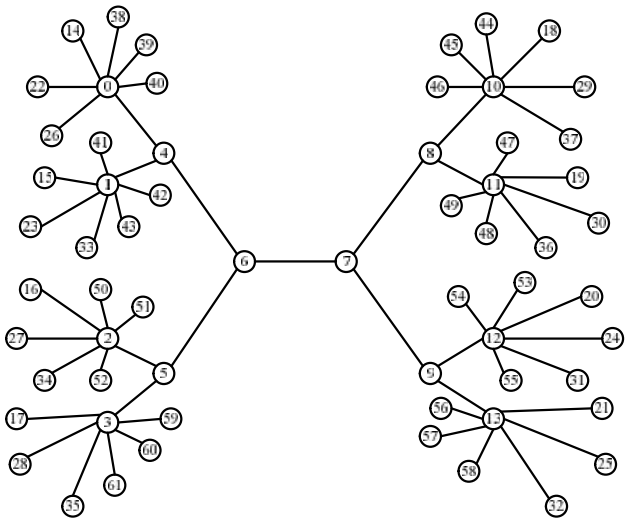

Рисунок 4.2 – Топологія мережі для випадку «важкого» трафіку

Нижче представлені мережеві топології для моделювання з «легким» та «важким» трафіками.Вузли, пронумеровані від 4 до 9, представляють основні маршрутизатори, тоді як вузли з номером 0,1,2, 3,10, 11, 12, 13 є крайовими маршрутизаторами. Решта вузлів представляють джерела трафіку.

Програмний код, застосований для опису вузлів має наступний вигляд:

```
for {set i 0}{$i < $num_r}{incr i} {
  set n(r$i)[$ns node]
  set r_list" [set r_list][list r$i]"
}
for {set i 1}{$i<=$num voip}{incr i}{
  set n(voip$i)[$ns node]
  set voip_list "[set voip_list][list voip$i]"
}
for {set i 1}{$i<=$num_ftpC}{incr i}{
  set n(ftpC$i)[$ns node]
  set ftpC_list "[set ftpC_list][list ftpC$i]"
}
for {set i 1}{$i<=$num_ftpS}{incr i}{
  set n(ftpS$i)[$ns node]
  set ftpS_list"[set ftpS_list][list ftpS$i]"
}
for {set i 1}{$i<=$num_httpC}{incr i}{
  set n(httpCSi)[$ns node]
  set httpC_list"[set httpC_list][list httpC$i]"
}for {set i 1}{$i<=$num_httpS}{incr i}{
  set n(httpS$i)[$ns node]
  set httpSlist"[set httpS_list][list httpS$i]"
}
```

Приклад 4.2 – Опис вузлів

Програмний код використовує цикл for для створення потрібної кількості вузлів. Усі вузли або маршрутизатори, згенеровані за допомогою коду [$ns node]. Для подальшої класифікації вузлів будуть вказані спеціальні назви.

4.1.2 Пропускна здатність зв’язку та затримка

Всі джерела трафіку підключені до крайових маршрутизаторів за допомогою симплексних зв’язків 10,0 Мб із затримкою 10 мс. Симплексні зв’язки створені в двох різних напрямках. Між двома вузлами 6 та 7 створюються два окремих симплексних канали, кожен з пропускною здатністю 10 Мб. Цього достатньо, щоб створити вузьке місце в мережі.

Від крайових маршрутизаторів до наступного внутрішнього маршрутизатора (вузол 4. 5. 8 і 9) створюється пропускна здатність 10,0 Мб з дуплексним зв’язком затримки 10 мс. Від маршрутизаторів 4, 5, 8 і 9 до маршрутизаторів вузьких місць використовується смуга пропускання 10 Мб з дуплексним зв’язком. Ці зв’язки використовуються для створення більш цілеспрямованого трафіку перед входом у вузьке місце. На рисунку 4.3 показані існуючі зв’язки та відповідна пропускна здатність для моделювання.

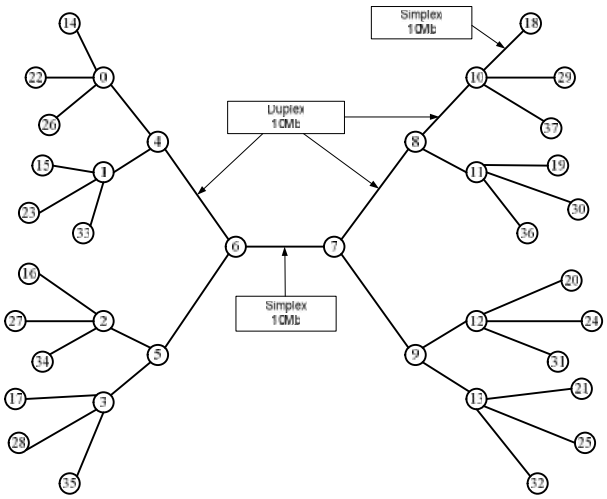

Рисунок 4.3 – Існуючі зв’язки та відповідна пропускна здатність

4.1.3 Джерело трафіку

Джерела трафіку поділяються на три різні категорії, які є:

- протокол передачі гіпертексту (HTTP);
- протокол передачі файлів (FTP);
- голос через IP (VoIP).

```
proc launchSession {pool clnt svr sessionSize_ interSession_ pageSize_
interPage_ objSize_ interObj_}{
  global ns n testTime
  set numSession_ expr round(1.4*$testTime/$interSession_)]
  puts "Creating max $numSession_sessions"
  $pool set testTime_$testTime
  $ns set sessionList "[$ns set sessionList]Spool"
  $pool set-num-client [llength $clnt]
  Spool set-num-server [llength $svr]
  set i 0
  foreach s $clnt {
    $pool set-client $i $n($s)
    incr i
  set i 0
  foreach s $svr {
    $pool set-server $i $n($s)
    incr i
  set numSession $numSession_
  set interSession [new RandomVariable/Exponential]
  $interSession set avg_$interSession_
  set sessionSize [new RandomVariable/Exponential]
  $sessionSize set avg_$sessionSize_
  $pool set-num-session $numSession
  set launchTime [$interSession value]
  for {set i 0} {$i<$numSession} {incr i} {
    if {[$pool set application_]==27} {
      set pageSize [new Random Variable/Constant]
      $pageSize set val_$pageSize_
    } else {
      set pageSize [new RandomVariable/Exponential]
      $paaeSize set avg_ $pageSize_
    }
    set interPage [new RandomVariable/Exponential]
    $interPage set avg_ $interPage_
    set interObj [new RandomVariable/Exponential]
    $interObj set avg_ $interObj_
    set objSize [new RandomVariable/Exponential]
    $objSize set avg_ $objSize_
    if {$launchTime<$testTime} {
      $pool create-session $i [$sessionSize value] $launchTime\
      $interPage $pageSize $interObj $objSize
    }
    set launchTime [expr $launchTime+[$interSession value]]
  }}
```

Приклад 4.3 – Процедура launchSession

Кожен тип трафіку має задані параметри і генерує випадковим чином кількість пакетів відповідно до цих початкових параметрів.

Сценарій tcl у файлі peer_setup.tl створить пул HTTP-сторінок із визначеними параметрами сеансу та буде викликаний командою launchSession у файлі diffnet.tcl. Призначені параметри трафіку – це середні значення експоненціального розподілу. Сценарій tcl у файлі peer_setup.tl створюэ пул FTP-сторінок, із визначеними параметрами сеансу.

Сценарій tcl у файлі peer_setup.tl створить VoIP-трафік із визначеними параметрами сеансу та викликаний командою launchSession у файлі diffnet.tcl.

Процедура launchSession використовується для встановлення параметрів, визначених для HTTP та FTP, та для запуску генерації трафіку.

Використовується для створення випадкової змінної та часу запуску для кожного сеансу (кожного запиту). Блок коду if {[$pool set application_]==27} встановить змінну розміру сторінки для FTP, інакше встановить для програми з номером 31 (HTTP) змінну розміру сторінки.

4.1.4 Конфігурація з’єднання

Між двома комунікаційними вузлами потрібно встановити кілька процедур. Ці процедури знаходяться у файлі 2q2p.tcl у процедурі confDSEdge.

```
proc confDSEdges {svr chit cir app service_type} {
  global ns q n cir_exp cir_http cir_ftp BE EF AF
  if {$service_type=="AF"} {
    set cp_ $AF(cp)
    set in_min_ $AF(in_min)
    set in_max_ $AF(in_max)
    set in_prob_ $AF(in_prob)
    set out_min_ $AF(out_min)
    set out_max_ $AF(out_max)
    set out_prob_ $AF(out_prob)
    set qlimit_ $AF(qlimit)
  }
```

Приклад 4.4 – Конфігурація з’єднання

```
  set q($svr$clnt) [[$ns link $n($svr) [$n($svr) neighbors]] queue]
  set q($clnt$svr) [[$ns link $n($clnt) [$n($clnt) neighbors]] queue]
  foreach qe "$svr$clnt $chit$svr" {
    $q($qe) set numQueues_ 1
    $q($qe) setNumPrec 2
    $q($qe) meanPktSize 300
    $q($qe) setPhysQueueSize 0 $qlimit_
    $q($qe) setMREDMode WRED
    $q($qe) addPolicyEntry -1 -1 TSW2CM $cp_ $cir $app
    $q($qe) addPolicerEntry TSW2CM $cp_[expr $cp_ + 1]
    $q($qe) addPHBEntry $cp_ 0 0
    $q($qe) addPHBEntry [expr $cp_ + 1] 0 1
    $q($qe) configQ 0 0 $in_min_ $in_max_ $in_prob_
    $q($qe) configQ 0 1 $out_min_ $out_max_ $out_prob_
    }
  }
  set in_min_ $AF(in_min)
  set in_max_ $AF(in_max)
  set in_prob_ $AF(in_prob)
  set out_min_ $AF(out_min)
  set out_max_ $AF(out_max)
  set out_prob_ $AF(out_prob)
}
```

Приклад 4.5 – Конфігурація з’єднання

4.1.5 Конфігурація вузького місця

Процедура confDSCore використовується для налаштування маршрутизаторів вузького місця. Викликається у файлі diffhet.tcl confDSCore r6 r7 та confDSCore r7 r6. Процедура confDSCore має два аргументи, a і b, які посилаються на ідентифікатор вузла, такий як у цьому випадку r6 або r7. Весь блок коду процедури показаний нижче.

```
proc confDSCore { a b } {
  global ns q n AF
  set q($a$b) [[$ns link $n($a) $n($b)] queue]
  $q($a$b) set numQueues_ 1   $q($a$b) setNumPrec 2
  $q($a$b) meanPktSize 300
  $q($a$b) setMREDMode WRED
  $q($a$b) setSchedularMode PRI
  # Options: RR. WRR. WIRR. PRI
  #AF   $q($a$b) addPHBEntry $AF(cp) 0 0
  $q($a$b) addPHBEntry [expr $AF(cp)+l] 0 1
  $q($a$b) configQ 0 0 $AF(in_min) $AF(in_max) $AF(in_prob)
  $q($a$b) configQ 0 1 $AF(out_min) $AF(out_max) $AF(out_prob)
  $q($a$b) setPhysQueueSize 0 $AF(qlimit)
}
```

Приклад 4.6 – Процедура confDSCore

4.2 Реалізація алгоритму

Розглянемо файли C ++, які були змінені з метою реалізації нечіткої логічної системи. Усі необхідні файли знаходяться в папці diffserv в головній папці ns. Необхідні файли:

- dsredq.h;
- dsredq.cc.

Дві основні функції були додані у файли dsredq, які є структурою даних (список посилань) та нечітким логічним кодом.

4.2.1 dsredq.h

У цьому файлі додається опис функції як публічної. Підпис функції, який було додано, є float fuzzy(int q_len, float avg, int q_lim). Код структури даних реалізовано у існуючій функції int enque (Packet * pkt, int prec, int ecn).

Структура даних додана для того, щоб відслідковувати всі значення довжини черги, включаючи фізичну довжину черги. Тип використовуваної структури даних – зв’язаний список. Зв’язаний список починається з оголошення структури даних у файлі dsredq.h. Кожен вузол у структурі списку має властивості, як показано нижче:

```
struct node {
int num
node *next;
};
```

Приклад 4.7 – Структура даних

Кожен вузол має одну змінну цілого числа. Int num використовується для зберігання фізичного розміру черги. Останній елемент у структурі вузла – це вказівник на вузол. Він використовується для з’єднання вузла з іншим вузлом. Структура вузла представлена на рисунку 4.4.

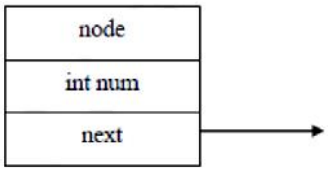

Рисунок 4.4 – Структура даних вузла

Перед тим, як використовувати зв’язаний список об’єкт повинен бути оголошений щоразу коли запускається моделювання. Це робиться шляхом виконання коду, як показано нижче:

```
node *r6_head;
node *r6_cur;
node *r6_temp;
node *r6_travel;

node *r7_head;
node *r7_cur;
node *r7_temp;
node *r7_travel;
```

Приклад 4.8 – Оголошення об’єкту

Мета цього блоку коду – пов’язати його з класом функції (redQueue), яка може бути використана в класі. За допомогою блоку коду кожного разу, коли розпочнається моделювання, створюється 8 покажчиків вузлів.

4.2.2 dsredq.cc

Dsredq.cc містить всі процедури або функції, які використовуються для моделювання. Коди для моделювання були вставлені у функцію enqueue() у цьому файлі. У файлі dsredq.cc вузли оголошені в конструкторі. Структура зв’язаного списку створюється, починаючи з першого пакету, який знаходиться в черзі. Новий запис додається до зв’язаного списку кожного разу, коли пакети успішно розміщуються в черзі.

```
redQueue::redQueue() {
  r6_head=NULL;
  r6_cur=NULL;
  r6_temp=NULL;
  r6_tra vel=NULL;
  r7_head=NULL;
  r7_cur=NULL;
  r7_temp=NULL;
  r7_travel=NULL;
  numPrec = MAX_PREC;
  mredMode = rio_c:
  q_=new PacketQueue();
}
```

Приклад 4.9 – Конструктор

Програмний код структури зв’язаного списку поділяється на дві частини за допомогою оператору if else, як показано нижче:

```
if(src_id==14 || src_id=15 || src_id==16 || src_id== 17|| src_id==22
|| src_id==23 || src_id==26 || src_id==27 || src_id=30 || src_id=31
|| src_id=34 || src_id==35)
{
if(r6_head==NULL)
  {
    r6_head=new node;
    r6_head->num=q_->length();
    r6_head->next=NULL;
    r6_cur=r6_headж
  } else {
    r6_temp=new node;
    r6_temp->num=q_->length();
    r6_temp->next=r6_cur;
    r6_cur=r6_temp;
  } } else {
if(r7_head==NULL)
  {
    r7_head=new node;
    r7_head->num=q_->length();
    r7_head->next=NULL;
    r7_cur=r7_head;
  } else {
    r7_temp=new node;
    r7_temp->num=q_->length();
    r7_temp->next=r7_cur;
    r7_cur=r7_temp;
}
```

Приклад 4.10 – Корекція структури зв’язаного списку

Призначення коду полягає у створенні та зберіганні зв’язаного списку для тих пакетів, які подорожують від маршрутизатора 6 до маршрутизатора 7

через вузьке місце. Список джерел id (14, 15, 16, 17, 22, 23, 26, 27, 30, 31, 34 і 35) посилається на ті джерела, які знаходяться праворуч від топології. Це важливо, щоб явно переконатися, що всі пакети передавані через вузьке місце від маршрутизатора 6 відстежуються.

4.3 Fuzzy Logic

Fuzzy Logic код вставляється у функцію enqueue(). Код fuzzy_result=fuzzy(q_->length(), prev_20, qlim) використовується для виклику коду, що знаходиться у самому файлі dsreqq.cc.

Виклик нечіткої логічної функції має набір параметрів, що передаються йому. Функції потрібно передавати три параметри, поточну фізичну довжину черги, середню довжину черги та межу фізичної черги. Функція буде передавати значення назад в змінну типу float для подальших дій, які було наведено у розділі 4.

4.4 Результати моделювання

Моделювання проводиться з двома видами трафіку:

- «легким» трафіком;
- «важким» трафіком.

У кожному випадку збираються два набори даних. Один набір є результатом моделювання без застосування нечіткої логіки. Інший набір є результатом моделювання з нечіткою логікою.

Результати моделювання представлені у формі таблиці. Таблиця статистики заповнюється з інтервалом 100 симуляційних секунд. Таблиця має два набори статистичних даних. Перший набір представляє статистику передачі пакетів для трафіку від маршрутизатора 6 до маршрутизатора 7. У той час як другий набір представляє статистику передачі пакетів для трафіку від маршрутизатора 7 до маршрутизатора 6.

Результати статистики передачі пакетів записуються починаючи з 100с моделювання. Приклад таблиці статистики передачі пакетів наведено в таблиці 4.1.

Таблиця 4.1 – Статистика передачі пакетів

Маршрутизатор	Отримані пакети	Передані пакети	Хвіст скинув пакет	RED скинув пакет
r6/r7	412493	350767	45826	16200
r6	295920	260106	25234	10180
r7	116573	90661	20392	5520

Терміни пропускної здатності, які будуть використовуватися далі, насправді мають на увазі надіслані пакети. Цей показник відображатиме кількість пакетів, які успішно пройшли через вузьке місце мережі. Вважається, що кожен пакет, який успішно пройшов через «вузьке місце», гарантійно досягне місця призначення. Вузьке місце – це фактично місце з найменшою пропускною здатністю та найвищою затримкою в мережі.

Моделювання з «легким» трафіком – це моделювання з 24 вузлами, які є одночасно, як джерелами, так і адресатами. Моделювання триває 800 моделюючих секунд. Загальна кількість наборів моделювання 25, з яких обчислюється середнє значення пропускної здатності кожного маршрутизатора та загальна пропускна здатність за кожні 100 секунд.

Моделювання з «важким» трафіком – це моделювання з 48 джерелами та вузлами призначення.

4.5 Аналіз отриманих результатів

В процесі моделювання для випадку «легкого» трафіку реалізація алгоритму RED із застосуванням нечіткої логіки показала кращу пропускну здатність (рисунки 4.5, 4.6). Значення пропускної здатності вузького місця

розраховувалися за формулою:

$$Throughput = \frac{packets\ sent}{packets\ received}, \qquad (4.1)$$

де: *packets sent* – кількість пакетів, які пройшли вузьке місце;

packets received – всього пакетів, надісланих до вузького місця.

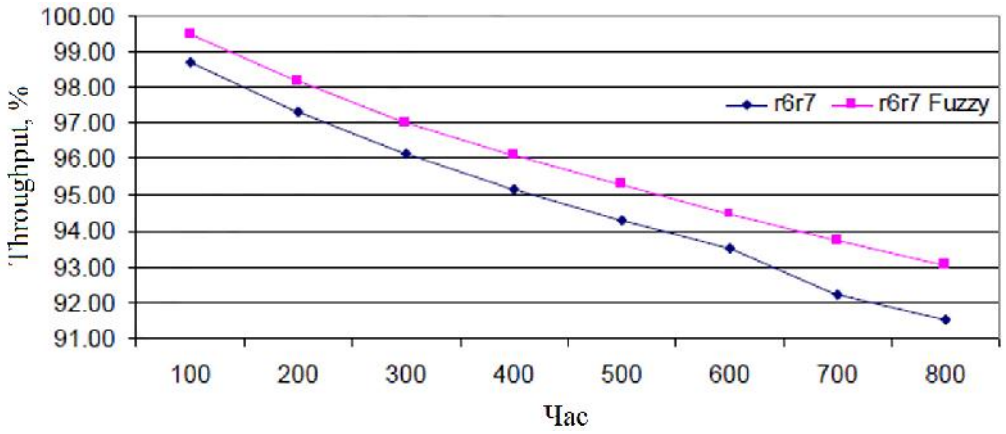

Рисунок 4.5 – Залежність пропускної здатності r6-r7 від часу для «легкого» трафіку

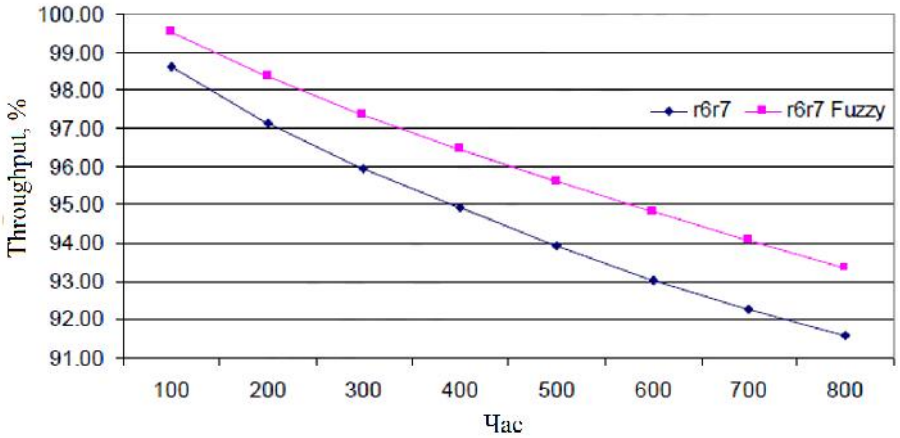

Рисунок 4.6 – Залежність пропускної здатності r7-r6 від часу для «легкого» трафіку

Розрахунок пропускної здатності проводився в одному напрямку (від r6 до r7 або від r7 до r6). Середній відсоток пакетів, успішно переданих через вузьке місце, показано на рисунках 4.5 та 4.6.

Реалізація алгоритму FuzzyRED покращує середній відсоток пропускної здатності пакетів. Поліпшення можна помітити в межах від 0,79% до 1,79% протягом 800 секунд моделювання. На рисунку 4.7 показано середній відсоток покращення пропускної здатності в каналі r6-r7.

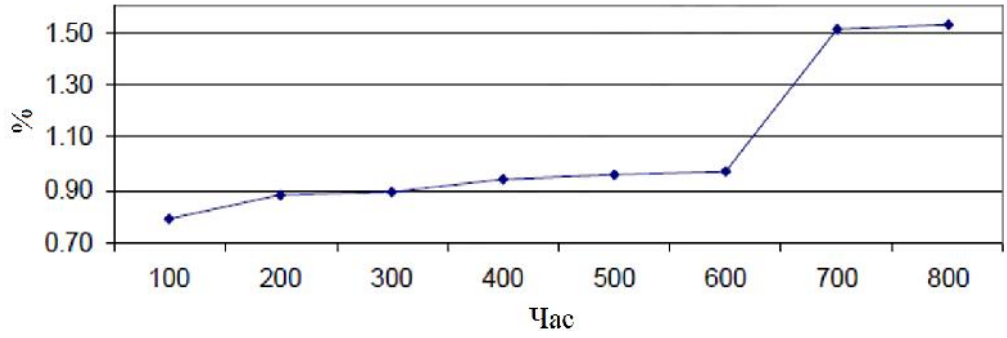

Рисунок 4.7 – Покращення пропускної здатності r6-r7 для «легкого» трафіку

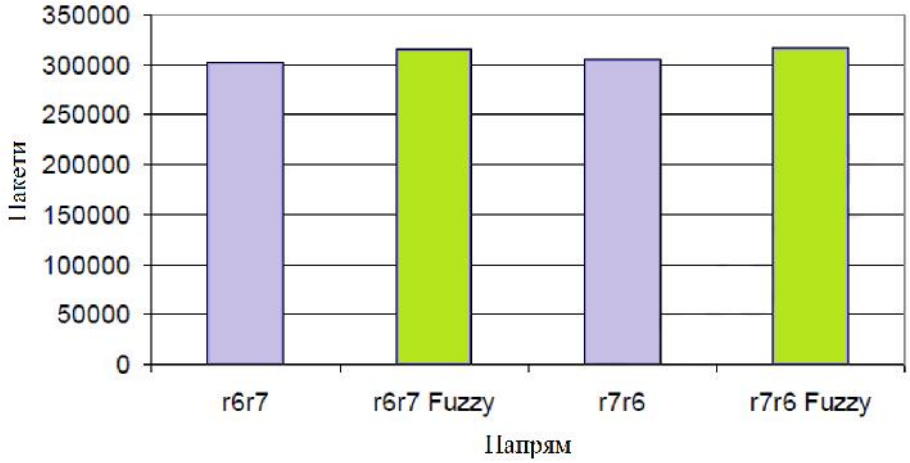

Рисунок 4.8 – Середня загальна кількість пакетів пройдених через вузьке місце

Загальна пропускна здатність – це кількість пакетів даних та пакетів підтвердження, які успішно були передані через вузьке місце. Для вузького

каналу r6-r7 загальна пропускна здатність показана на рисунку 4.8.

З наведеного рисунка можна зробити висновок, що за допомогою реалізації алгоритму FuzzyRED можна збільшити загальну кількість пакетів, які успішно пройшли через вузьке місце в мережі.

Залежність пропускної здатності r6-r7 від часу для випадку «важкого» трафіку, показано на рисунках 4.9 та 4.10.

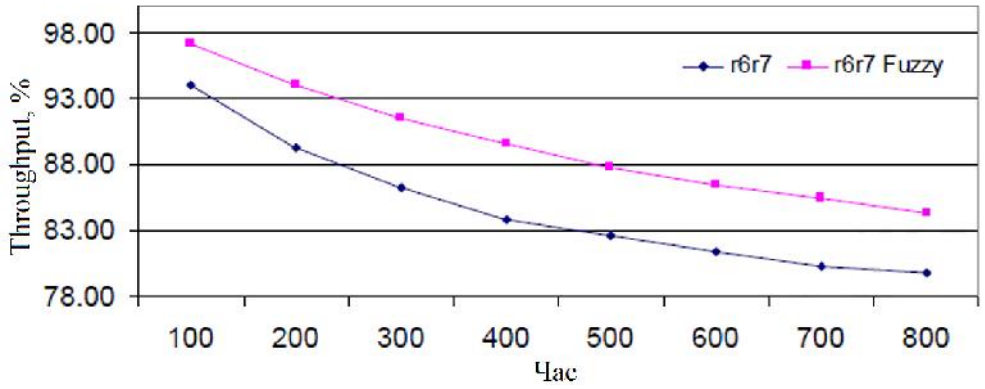

Рисунок 4.9 – Залежність пропускної здатності r6-r7 від часу для «важкого» трафіку

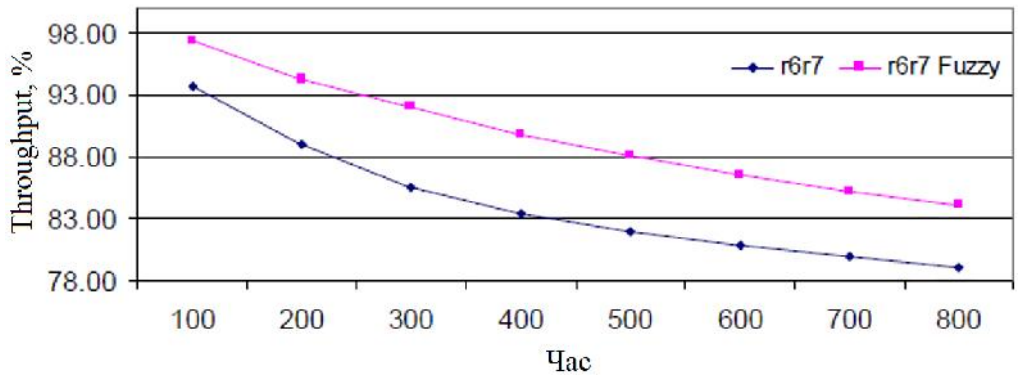

Рисунок 4.10 – Залежність пропускної здатності r7-r6 від часу для «важкого» трафіку

Поліпшення можна помітити в межах від 3,12% до 6,46% протягом 800 секунд моделювання. На рисунку 4.11 показано середній відсоток

покращення пропускної здатності в каналі r6-r7.

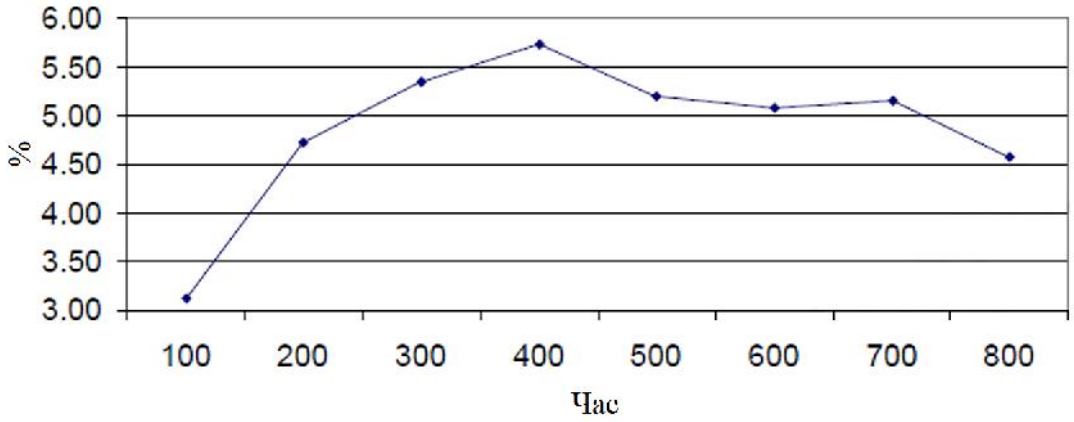

Рисунок 4.11 – Покращення пропускної здатності r6-r7 для «важкого» трафіку

Графіки на рисунку 4.11 показує, що відсоток поліпшення пропускної здатності зменшується в часі. Це викликано тим, що кількість пакетів, які намагаються пройти через вузьке місце, збільшується настільки, що буфер не встигае їх обробляти.

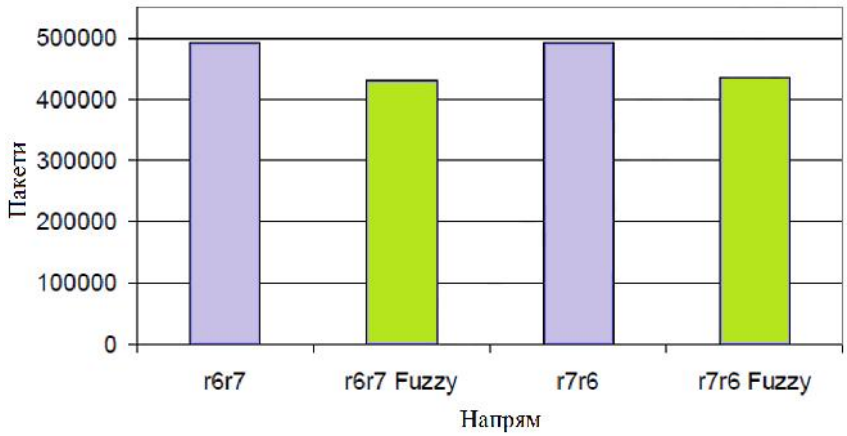

Рисунок 4.12 – Середня загальна кількість пакетів пройдених через вузьке місце для «важкого» трафіку

На рисунку 4.12 може спостерігатися незначне зменшення загальної кількісті пакетів для випадку застосування алгоритму FuzzyRED. Незважаючи на незначне зменшення, FuzzyRED підтримує середній загальний відсоток пропускної здатності на мінімальному рівні 87%.

ВИСНОВКИ

Боротьба з перевантаженнями каналів в комп’ютерних мережах – дуже важлива частина проблеми забезпечення якості обслуговивання та управління трафіком. Перевантаження в сучасних комп’ютерних мережах виникає тоді, коли загальний обсяг інформації, що передається, стримко наближається до максимальної припустимої пропускної здатності мережі.

За останні роки було розроблено та досліджено цілу низку різноманітних методів активного керування чергами для раннього виявлення перевантаження маршрутизаторів, але стрімке зростання глобальної мережі Інтернет та проникнення інформаційних технологій в усі сфери життєдіяльності людини викликають необхідність подальшого вдосконалення методів боротьби з перевантаженнями.

Під час виконання атестаційної роботи були успішно вирішені наступні завдання:

- проведено аналіз впливу різноманітних чинників на продуктивність комп’ютерних мереж та розглянуті сучасні методи управління мережевим трафіком;
- запропоновано метод управління чергами маршрутизаторів із застосуванням нечіткої логіки на основі алгоритма AQM RED;
- проведено імітаційне моделювання з метою підтвердження приведених теоретичних викладок;
- проведено аналіз отриманих результатів моделювання.

Під час виконання атестаційної роботи було вирішено поставлену задачу – запропоновано метод управління чергою FuzAQM, який сприяє зростанню пропускної здатності в IP мережах.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Gerla M., Kleinrok L. «Flow Control: A Comparative Survey», IEEE Transactions on Communications, vol.28 (4), 1980, pp.553-574.

2. Mahajan, R., Floyd, S. and Wetherall, D. (2001), —Controlling high-bandwidth flows at the congested router‖, Network Protocols, 2001. Ninth International Conference on, IEEE, pp. 192–201.

3. Jean-Philippe, Martin-Flatin, Sylvain Ravot. “TCP Congestion Control in Fast Long-Distance Networks”. Technical Report CALT-68-2398 California Institute of Technology, pp. 89-97, USA July 10, 2002.

4. Танненбаум Э. Компьютерные сети [Текст] / Э. Танненбаум – СПб.: Питер, 2003. – 992 с.

5. Jacobson, V., “Congestion Avoidance and Control,” Computer Communication Review, volume 18, no. 4, pp. 314-329, August 1988.

6. Brandauer, C., Iannaccone, G., Diot, C., Ziegler, T., Fdida, S., and May, M., “Comparison of Tail Drop and Active Queue Management Performance for bulk-data and Web-like Internet Traffic,” In Proceeding of ISCC, pp. 122-129, IEEE, July 2001.

7. Dimitriou S., Tsaoussidis V., Head-to-Tail: Managing Network Load through Random Delay Increase, Proceedings of IEEE ISCC, Aveiro, Portugal, July 2007.

8. Lin D., Morris R., (1997), “Dynamics of Random Early Detection”, In Proc. of ACM SIGCOMM, 127-137.

9. Claypool, M., Kinicki, R. and Hartling, M. (2004), —Active queue management for web traffic”, Performance, Computing, and Communications, 2004 IEEE International Conference on, IEEE, pp. 531–538.

10. Athuraliya, S., Low, S.H., Li, V.H. and Yin, Q. (2001), —REM: Active queue management”, Network, IEEE, Vol. 15 No. 3, pp. 48–53.

11. Feng W., Kandlur D., Saha D., Shin K., (1999/b), “Blue: A new class of

active queue management schemes," Technical Report, CSE-TR-387-99, U. Michigan 286-299.

12. Floyd, S., & Fall, K. (1999). "Promoting the use of end-to-end congestion control in the Internet". IEEE/ACM Transactions on Networking (TON), 7(4), 458-472.

13. Hollot C. V., Misra V., Towsley D., Gong W., (2002), "Analysis and Design от Controllers for AQM Routers Supporting TCP Flows", IEEE Transactions on Automatic Control, Vol. 47, No: 6, 945-959.

14. Feng W., Shin K. G., Kandlur D. D., Saha D., (2002/b), "The BLUE active queue management algorithms", IEEE/ACM Transactions on Networking, Vol.10, No:4, 513- 528.

15. Xu H., Wu T., Zhou X.-J. and Zhu, Y. (2004), —IP network control and AQM‖, Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on, IEEE, Vol. 1, pp. 500–504.