Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Методи підвищення пропускної здатності комп’ютерних мереж

(тема)

Виконав:
студент II курсу, групи КСМм-20-1
Петрук В.В.
(прізвище, ініціали)

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

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

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

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

Допускається до захисту

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

2021 р.

Харківський національний університет радіоелектроніки

Факультет комп’ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

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

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Петруку В’ячеславу Віталійовичу
(прізвище, ім’я, по батькові)

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

затверджена наказом по університету від “ 05 ” листопада 2021 р. № 1656 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 13 грудня 2021 р.

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

4. Перелік питань, що потрібно опрацювати в роботі
1) аналіз сучасного стану проблеми
2) огляд технологій управління перевантаженням та середньою затримкою
3) моделі управління мережним трафіком
4) вибір програмних та апаратних засобів реалізації
5) проведення експериментальних досліджень
6) висновки

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

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз стану проблеми та сучасних методів її	09.11.21–11.11.21	
	вирішення		
2	Огляд технологій управління перевантаженням	12.11.21–15.11.21	
	та середньою затримкою		
3	Розробка моделі управління мережним	16.11.21 –25.11.21	
	трафіком		
4	Вибір програмних та апаратних засобів	26.11.21 –01.12.21	
	реалізації		
5	Тестування запропонованого метода	02.12.21–03.12.21	
6	Оформлення пояснювальної записки	04.12.21–-07.12.21	

Дата видачі завдання 8 листопада 2021 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 92 с., 32 рис., 1 табл., 1 дод., 10 джерел.

AQM, АЛГОРИТМ, МАРШРУТИЗАЦІЯ, НАВАНТАЖЕННЯ, КАНАЛ ЗВ’ЯЗКУ, ПОРІГ СКИДАННЯ, ПРОПУСКНА СПРОМОЖНІСТЬ, ТОПОЛОГІЯ, ЧАС ПОДВІЙНОГО ОБЕРТУ.

Мета кваліфікаційної роботи – розробка моделей та методів управління перевантаженням та середньою затримкою в комп’ютерних мережах.

В роботі розглянуті основні питання програмно-алгоритмічних засобів підвищення ефективності роботи мереж. Також проведено роботу по аналізу факторів, що спливають на показники продуктивності комп’ютерних мереж. Представлено метод «придушення» агресивних потоків. Проведено імітаційне моделювання.

ABSTRACT

Master’s thesis: 92 pages, 32 figures, 1 tables, 1 appendices, 10 sources.

AQM, ALGORITHM, ROUTING, LOADING, COMMUNICATION CHANNEL, RESET THRESHOLD, CAPACITY, TOPOLOGY, ROUND TRIP TIME.

The purpose of the qualification work is to develop models and methods for managing congestion and average latency in computer networks.

The main issues of software and algorithmic tools to increase the efficiency of networks are considered in the paper. Work has also been done to analyze the factors that affect the performance of computer networks. The method of "suppression" of aggressive flows is presented. Simulation modeling is carried out.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

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

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

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

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

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

CWA – дія вікна перевантаження (англ., Congestion window action)

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

DACK – підтвердження повторного TCP-пакету (англ., Duplicate TCP packet acknowledgment)

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

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

HTTP – протокол передачі гіпертексту (англ., Hypertext transfer protocol)

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

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

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

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

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

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

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

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

ВСТУП

Протягом багатьох років інтернет-провайдери пропонували своїм клієнтам все більшу пропускну здатність. Велика пропускна здатність потрібна лише досвідченим користувачам, які дійсно використовують більшу пропускну здатність.

Однак для звичайного клієнта, який проводить більшу частину свого часу в Інтернеті, кількох мегабіт пропускної здатності може бути більш ніж достатньо. У таких випадках головною причиною «повільного Інтернету» часто є не пропускна здатність, а затримка в мережі.

Існує кілька причин затримки в мережі. По-перше, кожен пристрій має затримку обробки, тобто час, необхідний пристрою для обробки частини даних. Часто вона настільки низька, що її майже немає. Принаймні, не виявляється для людей.

Іншою причиною затримок є час, необхідний для поширення даних уздовж каналу, який майже постійний, оскільки це диктується фізичним середовищем.

Третя причина, і з якою можна щось зробити, це розмір буферів в Інтернеті. Оскільки кожен пристрій має буфер, який є тимчасовим сховищем для даних перед тим, як вони будуть передані на наступній пристрій, дані ставляться в чергу. Розмір буфера залежить від обсягу доступної пам’яті в пристрої. Оскільки ціни на пам’ять знизилися, обсяг пам’яті в пристроях збільшився.

Роль буфера полягає в здатності поглинати пакети даних, тому буфер повинен бути достатньо великим. Однак великі буфери (також звані роздутими буферами) через спосіб налаштування більшості інтернет-пристроїв викликають велику затримку при заповненні. Усі сегменти даних, що проходять через буфер, повинні зачекати, щоб їх переслати. Сегмент вхідних даних отримує додаткову затримку, а саме час, який потрібно

передати кожному пакету перед ним. Ця затримка може бути в кілька разів більшою, ніж початкова затримка мережі.

Оскільки буфери, заповнені даними, викликають більшу затримку, це часто є причиною повільного інтернет-з’єднання. Оскільки кожне з’єднання може мати однакову затримку, це помітніше при коротких з’єднаннях, ніж при передачі більш тривалих файлів. Якщо передача файлу займає на секунду більше до завершення, це, можливо, нічого страшного, але якщо кожна відкрита веб-сторінка затримується на додаткову секунду, це велика проблема.

1 МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Метою кваліфікаційної роботи є створення «очищувача черги» в мережевих пристроях, який виявляє черги, які викликають великі затримки в мережах. Оскільки черги формуються на вузькому місці, це проявляється як індукована затримка в мережі.

Індуковані затримки можуть варіюватися від незначних мілісекунд до кількох секунд. Затримка в кілька секунд є неприйнятною з точки зору користувача та деяких мережевих застосунків.

Під час виконання магістерської кваліфікаційної роботи необхідно:

- провести докладний аналіз літературних джерел, присвячених різноманітним проблемам пропускної спроможності в сучасних комп’ютерних мережах;
- виконати огляд існуючих моделей, а також методів управління перевантаженнями в комп’ютерних мережах;
- запропонувати метод управління чергами маршрутизаторів та представити результати імітаційного моделювання запропонованого метода;
- провести аналіз отриманих результатів.

2 РІВНІ МОДЕЛІ OSI

Існують рішення проблеми роздутих буферів і кількості даних, що знаходяться в черзі в них. Рішення потрібно впроваджувати та активувати на кожному проміжному пристрої, тому це може бути важко зробити швидко, навіть якщо алгоритми ретельно перевірені та дають розумні результати.

Можливе рішення може полягати в тому, щоб інфраструктура в Інтернеті була такою, як вона є, і спробувати вирішити проблеми з буфером іншим способом.

Рівні непрозорі, це означає, що програма, яка надсилає дані, повинна довіряти наступним шарам, щоб переслати дані правильному одержувачу.

Розшаровування стека також полегшує візуалізацію потоку даних, що надсилаються з однієї програми до іншої.

2.1 Канальний рівень

Канальний рівень відповідає за передачу даних до і з фізичного мережевого каналу. Дані, надіслані через канальний рівень, отримують заголовок (з інформацією про наступний вузол мережі).

На канальному рівні всі дані надсилаються до наступного доступного вузла, яким може бути як кінцевий одержувач, так і мережевий шлюз.

Усі вихідні дані з канального рівня тимчасово зберігаються в буфері, який забезпечує формування пакетів даних у постійний потік даних.

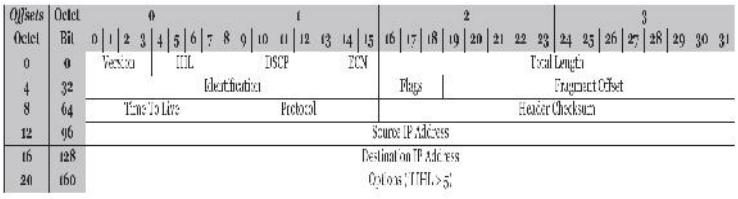

Рисунок 2.1 – Формат заголовка IPv4

2.2 Мережевий рівень

Мережевий рівень – це місце, де відбувається маршрутизація підмережі. Коли дані надходять із канального рівня, заголовок канального рівня (часто заголовок Ethernet) видаляється, а заголовок Інтернет-протоколу (IP) аналізується. Якщо одержувач не є поточним пристроєм, пакет пересилається до підмережі, яка, як передбачається, знає одержувача.

Заголовок IP містить інформацію про одержувача і відправника, який протокол використовується тощо. Заголовок IPv4 наведено на рисунку 1.1. Оскільки Інтернет був заповнений пристроями, кількість доступних унікальних адрес зменшується. Була розроблена нова версія під назвою IPv6, яка розширює кількість доступних адрес з 2^{32} до 2^{64}, чого має вистачити на найближче майбутнє.

Якщо дані надсилаються на поточний пристрій (тобто збігаються IP-адреси), IP-заголовок видаляється, і дані пересилаються вище по ланцюжку.

2.3 Транспортний рівень

Транспортний рівень обробляє перенаправлення даних з мережевого рівня до певної програми. На пристрої може одночасно працювати кілька програм.

Кожній програмі з підтримкою мережі призначається один або кілька номерів портів. Номер порту – це унікальний ідентифікатор, який ідентифікує програму та протокол, який використовується протягом усього життя програми.

UDP і TCP є найпоширенішими Інтернет-протоколами. UDP був розроблений для даних, які не потребують надійної передачі, таких як зв’язок у реальному часі, потокове передавання тощо, тоді як TCP був розроблений для надійної доставки даних.

2.4 Прикладний рівень

Прикладний рівень є загальним знаменником для всіх програм, що працюють на пристрої, як підключених до мережі, так і без них.

Він містить такі протоколи, як протокол передачі гіпертексту (HTTP).

Загальним портом HTTP є порт 80, що означає, що програма, яка прослуховує HTTP-трафік, повинна слухати порт 80 (якщо клієнт не знає про інший порт). Оскільки лише одна програма може бути прив’язана до певного порту, жодна інша програма не може слухати порт 80 протягом тривалості роботи веб-сервера.

3 ПРОТОКОЛ КЕРУВАННЯ ПЕРЕДАЧЕЮ ТА КОНТРОЛЬ ПЕРЕВАНТАЖЕНЬ TCP

3.1 Протокол керування передачею

TCP лежить в основі двох найбільш широко використовуваних мережевих архітектур, еталонних моделей OSI і TCP/IP. Він розташований на транспортному рівні в обох моделях, як показано на рисунку 3.1.

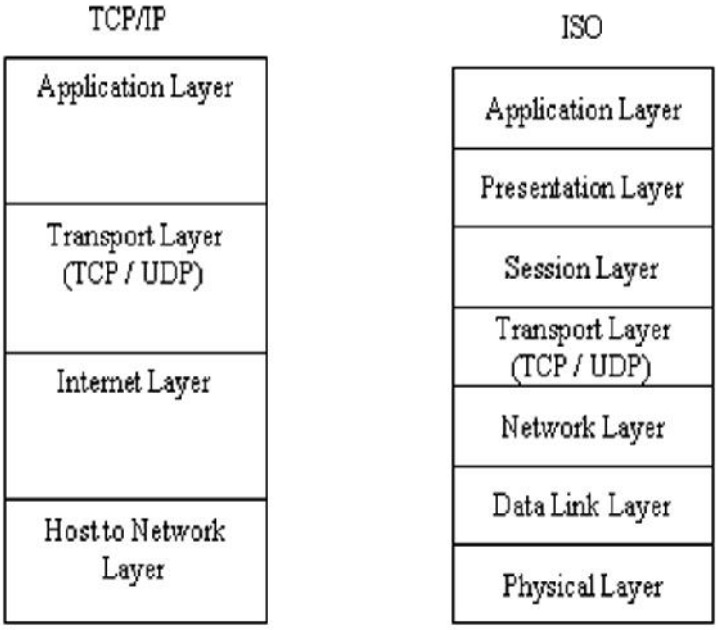

Рисунок 3.1 – Співвідношення рівнів моделей TCP/IP та OSI

Основна мета транспортного рівня полягає в тому, щоб надати наскрізний сервіс транспортування даних верхнім рівням, а також діяти таким чином, щоб верхні рівні могли нормально функціонувати, навіть якщо на нижчих рівнях відбуваються зміни, наприклад оновлення обладнання.

Транспортний рівень надає два види послуг. Одна з них базується на простій ідеї надсилання повідомлень без будь-якої гарантії того, чи прибуде повідомлення до місця призначення чи ні, і без упорядкування надісланих

повідомлень. Цей тип послуги корисний, якщо втрата деяких повідомлень не вплине на якість передачі, як у випадку потокового відео та аудіо, коли деякі пакети не вплинуть на службу, за умови, що кількість втрачених пакетів знаходиться в прийнятному діапазоні.

Також у цьому типі послуги (тобто відео та аудіо) повторна передача відсутніх даних неприйнятна, оскільки це послуга в реальному часі, коли більшість пакетів надходить вчасно, набагато важливіше, ніж повторна передача деяких пакетів.

Транспортний рівень реалізує ці типи послуг за допомогою дейтаграми користувача, протоколу керування передачею та протоколу керування перевантаженням TCP, широко відомого як UDP. UDP не надає гарантій щодо доставки та правильності повідомлень, а також не забезпечує замовлення.

Крім того, UDP підходить для програм, які забезпечують власну послідовність і механізми контролю потоку.

Другим типом послуг, що надаються транспортним рівнем, є гарантовані послуги. Тут гарантується правильність і впорядкованість доставлених даних. Реалізація цієї послуги здійснюється TCP.

Протокол керування передачею, широко відомий як TCP, є протоколом транспортного рівня для транспортування даних через мережу в надійному та безпомилковому режимі.

Він доставляє повідомлення з одного кінця на інший і гарантує, що всі пакети були доставлені непошкодженими та в порядку передачі. TCP - це протокол, орієнтований на з’єднання, тому перед тим, як TCP почне передавати дані між двома кінцями, він встановлює угоду про те, як має працювати з’єднання. Це здійснюється шляхом обміну керуючими пакетами перед передачею реальних даних.

Через природу IP-маршрутизації, що використовується в Інтернеті, не всі пакети рухатимуться по одному шляху від джерела до місця призначення. Цей факт може призвести до того, що пакети будуть переупорядковані, тому

деякі пакети можуть бути отримані не в порядку. TCP повинен бути готовий прийняти пакети, що були пошкоджені, і очікувати затримок деяких пакетів. Зазвичай TCP передбачає обмеження часу для досягнення кожного пакета місця призначення, і пакет буде вважатися втраченим після цього обмеження часу.

Іншими важливими функціями TCP є контроль потоку та контроль перевантажень. При управлінні потоком TCP забезпечує координацію між відправником і одержувачем, тому відправник не надсилатиме більше, ніж ємність буфера одержувача.

TCP повинен мати можливість координувати джерело та адресата, щоб джерело не переповнювало буфер одержувача. Для досягнення цього одержувач повідомляє відправнику свій максимальний розмір буфера під час фази встановлення з’єднання.

Під час контролю перевантажень TCP виявляє перевантаження, використовуючи втрату пакетів як ознаку перевантаженості в мережі, а потім намагається усунути перевантаження, зменшуючи швидкість передачі відправника, щоб запобігти створенню постійного перевантаження. TCP повинен мати певну інформацію про стан зв’язку, щоб уникнути введення в канал зв’язку більше даних, ніж його ємність. Для цього використовуються алгоритми контролю перевантажень.

Більше того, для того, щоб TCP працював у такому середовищі, як Інтернет, TCP повинен мати можливість обробляти з’єднання зі змінним часом передачі в обидва боки (час в обидва боки, RTT, це час від відправлення пакета до отримання підтвердження).

Оскільки TCP, як передбачається, може з’єднати будь-які два хости в Інтернеті, незалежно від того, наскільки вони віддалені один від одного, він повинен мати можливість приймати різний час RTT для різних з’єднань і навіть різний час для одного з’єднання. Таким чином, він повинен мати можливість налаштувати свій механізм тайм-ауту, щоб прийняти його з зміною часу RTT.

3.2 Вікно перевантаження

Алгоритм плаваючого вікна лежить в основі надійності та уникнення перевантажень, що надаються TCP. Розуміння алгоритму плаваючого вікна, що використовується в TCP, допоможе зрозуміти алгоритм уникнення перевантажень і те, як TCP забезпечує надійність і порядок.

В алгоритмі ковзаючого вікна TCP відправник підтримує вікно, яке називається Congestion Window або коротко cwnd, і надає порядковий номер для кожного пакета в цьому вікні. cwnd визначає, скільки пакетів можна надіслати до отримання будь-якого підтвердження. Таким чином, cwnd контролює швидкість надсилання TCP (чим більше cwnd, тим вище швидкість відправлення, і навпаки). Відправник відстежує останнє підтверджене підтвердження та останній надісланий пакет. Коли відправник отримує підтвердження, він надсилає новий пакет і оновлює останні надіслані та останні отримані змінні. Крім того, відправник додає таймер до кожного надісланого пакета. Якщо таймер закінчується до отримання підтвердження, пакет може бути відправлено повторно.

З іншого боку, одержувач обробляє вікно (буфер) і три змінні: розмір вікна, порядковий номер останнього отриманого пакета та найбільший порядковий номер, який можна прийняти, який обчислюється шляхом додавання розміру буфера до порядкового номеру останнього отриманого пакета. Коли одержувач отримує пакет, порядковий номер пакета повинен бути більшим, ніж порядковий номер останнього отриманого пакета, і меншим або дорівнювати найбільшому порядковому номеру, який може бути прийнятий. Якщо ні, то пакет відкидається, оскільки він знаходиться за межами вікна одержувача.

З іншого боку, якщо новий пакет знаходиться у вікні одержувача, одержувач приймає новий пакет, а якщо новий пакет є наступним очікуваним пакетом, то він оновлює змінну останнього отриманого пакета та надсилає підтвердження для цього пакета. Однак іноді пакет може надійти не в

порядку; в цьому випадку одержувач не визнає новий пакет. Замість цього він надсилає підтвердження для останнього в порядку отриманого пакета. Це називається повторним підтвердженням.

Наприклад, якщо останнім отриманим пакетом є 3, а потім було отримано 5 і 6 перед 4, тоді одержувач надішле дублікат підтвердження з порядковим номером 3, щоб повідомити відправника, що він все ще чекає на пакет 4. Коли пакет 4, нарешті, досягне одержувача, одержувач видасть підтвердження для пакета номер 6, яке підтвердить усі попередні пакети. З іншого боку, якщо пакет 4 ніколи не досягне одержувача (тобто загубиться на шляху), таймер відправника закінчиться, і він надішле пакет знову.

Основна особливість цього алгоритму полягає в тому, що він контролює кількість пакетів у мережі в будь-який момент, роблячи максимальну кількість пакетів, які відправник може одночасно помістити в мережу, рівною розміру вікна, і що відправник не може передати нові пакети у мережу до тих пір, поки не отримає підтвердження, що старі були отримані.

3.3 Контроль перевантаження TCP

TCP застосовує п’ять алгоритмів контролю перевантажень, а саме повільний старт, уникнення перевантажень, механізм повторної передачі (швидка повторна передача і швидке відновлення та механізм тайм-ауту. Алгоритми повільного запуску та уникнення перевантажень були додані до TCP, а потім повний опис алгоритмів було вперше задокументовано для Інтернету. Після цього більш детально вказали всі алгоритми і обговорили більше питань і проблем щодо дій, які слід виконати TCP після перезапуску ідеальних з’єднань, і вимоги, які одержувач TCP повинен гарантувати в підтвердження (ACK). Також вони підняли деякі проблеми безпеки, наприклад, можливість атаки на систему, яка запускає TCP, підробляючи повторювані підтвердження або викликаючи втрату пакетів.

3.3.1 Повільний старт

Коли TCP надсилає дані, він обмежує швидкість передачі мінімальним розміром вікна перевантаження cwnd і буфером одержувача. Проте було виявлено, що якщо TCP починає з’єднання, надсилаючи все вікно одночасно, це може спричинити непотрібну тимчасову перевантаженість і, отже, відкидання пакетів, що погано впливає на продуктивність TCP. Тому, коли TCP починає надсилати в перший раз або коли він перезапускає розірване з’єднання, він повинен надсилати пакети поступово.

Ідея механізму повільного запуску полягає в тому, що замість того, щоб одночасно надсилати ціле вікно, TCP починає з надсилання одного пакету, а потім збільшує розмір вікна експоненціально, поки не досягне максимально доступного розміру вікна (мінімум вікна відправника та буфера одержувача). Повільний запуск експоненціально збільшує вікно перевантаження, подвоюючи розмір вікна перевантаження з кожним часом RTT.

Це робиться шляхом збільшення розміру вікна перевантаження на один пакет для кожного нового підтвердження.

Оскільки повільний запуск визначає початкову поведінку TCP-з’єднань, одна з його цілей полягає в тому, щоб поступово виявляти пропускну здатність каналу шляхом постійного збільшення швидкості передачі, поки пропускна здатність каналу не буде досягнута, коли відбуваються втрати пакетів.

У цей момент TCP переходить до наступного механізму, механізму уникнення перевантажень, який оброблятиме решту часу життя з’єднання. З цієї причини TCP визначає поріг, який називається порогом повільного запуску (ssthresh), який визначає межу між повільним запуском і режимом уникнення перевантажень.

Таким чином, TCP збільшує cwnd експоненціально, поки не досягне ssthresh, де він перемикається в режим уникнення перевантажень.

3.3.2 Уникнення заторів

Фаза уникнення перевантажень є найважливішою фазою в часі життя TCP, оскільки вона представляє стан рівноваги часу життя з’єднання. Рівноважний стан визначається як стан, коли відправник TCP поміщає пакет в мережу, коли інший пакет з того самого з’єднання залишає мережу у одержувача.

Більше того, більшість наслідків помилок передачі матиме місце на цій фазі, оскільки це найдовша фаза.

Коли TCP починає надсилати дані, вікно перевантаження починає експоненційно зростати під час фази повільного запуску, поки пакет не буде відкинуто, що вказує на дві речі: по-перше, пропускна здатність каналу була досягнута, а по-друге, почалася фаза уникнення перевантажень. На цьому етапі бере свій контроль механізм уникнення перевантажень TCP. Механізм уникнення перевантажень TCP виконує в основному два різні та важливі завдання:

- перше – прийняти рішення про збільшення/зменшення розміру вікна перевантаження (напрямок зміни);
- друге – визначити значення збільшення/зменшення у вікні перевантаження (сума зміни).

Щоб визначити напрямок зміни, TCP використовує відкидання пакетів, як сигнал про те, що вікно перевантаження має бути зменшено (напрямок вниз) і відсутність падінь, а отже, отримання нового підтвердження, як сигнал, що вікно перевантаження потрібно бути збільшено (напрямок вгору).

TCP визначає величину зміни (зменшення/збільшення) на основі напрямку цієї зміни за допомогою механізму AIMD (адитивне збільшення, мультиплікаційне зменшення). TCP лінійно збільшує вікно перевантаження (адитивне збільшення) на 1/windowsize розмір вікна з кожним отриманим новим підтвердженням.

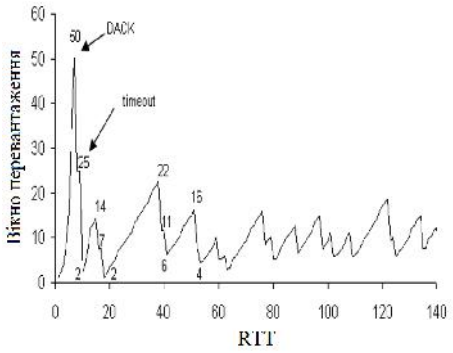

Рисунок 3.2 – Повільний запуск та уникнення перевантажень за наявності помилок

Це еквівалентно збільшенню вікна перевантаження на один пакет кожного RTT. Іншими словами, TCP буде чекати, поки все вікно буде відправлено безпечно (тобто без жодної втрати пакетів), а потім збільшить розмір вікна на один пакет.

Однак, якщо відбулася втрата пакету, TCP сприймає це як вказівку на те, що нове вікно перевищило ємність каналу або що в мережу було введено нове навантаження (наприклад, нові користувачі починають завантажувати FTP-файли). У цьому випадку TCP зменшить розмір вікна перевантаження до половини свого розміру до того, як відбулося відкидання.

На рисунку 3.2 показано типову поведінку перевантаження TCP (повільний старт і уникнення перевантажень) за наявності втрат пакетів, які викликають тайм-аути та повторювані підтвердження. Вісь Y представляє розмір вікна перевантаження в пакетах під час кожного циклу. Числа у верхній частині кожної кривої – це значення розміру вікна перевантаження до виникнення помилок, а числа в нижній частині – це розмір після того, як TCP відрізає вікно.

Причина вибору 0,5 як коефіцієнт скорочення у випадку повторюваних підтверджень (DACKs) полягає в тому, що зазвичай, коли відбувається перевантаження, це означає, що почалося нове з’єднання, і воно зазвичай використовує 50% пропускної здатності мережі, а отже, доступна пропускна

здатність зменшується на 50%, тому TCP потрібно зменшити розмір вікна на 50%, щоб забезпечити справедливе спільне використання з’єднання. Це зменшить продуктивність TCP мультиплікативно, оскільки з кожним відкиданням пакету TCP множить поточний розмір вікна на 0,5.

Метою механізму AIMD є досягнення двох основних цілей: по-перше, досягнення справедливості серед конкуруючих потоків TCP. По-друге, щоб досягти ефективного використання пропускної здатності вузького місця. Таким чином, використання мультиплікативного зменшення після скидання призведе до того, що з’єднання з більшими вікнами (тобто з більшою часткою в перевантаженні) скоротять більше даних (наприклад, з’єднання з розміром вікна 100 пакетів призведе до скорочення 50 пакетів, а з’єднання з вікном з 10 пакетів – тільки до 5 пакетів), що допоможе швидше усунути перевантаження та підвищить справедливість між конкуруючими потоками. Крім того, мультиплікативне зменшення змусить джерела швидко сповільнюватись, коли виникає перевантаження, щоб дати перевантаженим маршрутизаторам достатньо часу для усунення заторів.

З іншого боку, адитивне збільшення допомагає TCP м’яко досліджувати пропускну здатність каналу, щоб уникнути коливань, які можуть виникнути, якщо використовується агресивне мультиплікативне збільшення. Однак адитивне збільшення забезпечить лінійне збільшення швидкості відправки для всіх з’єднань, які мають однаковий RTT.

3.3.3 Виявлення втрати пакету

TCP використовує два способи виявлення втрат пакетів – повторного підтвердження та часу очікування повторної передачі. Для ідентифікації того, що пакет відсутній, використовується дублікат підтвердження. Наприклад, на рисунку 3.3 відправник TCP надсилає шість пакетів одержувачу TCP, а пакет номер три був відкинутий перевантаженим маршрутизатором на шляху з’єднання. Коли одержувач отримує перші два

пакети, він надсилає для них підтвердження (пакети один і другий). Однак пакет три відсутній, а замість нього отримано пакет чотири. Коли одержувач отримує непорядковий пакет (тобто пакет чотири), він не підтвердить його, а натомість він повторно відправить підтвердження останнього в порядку пакета, який є пакетом два. Це підтвердження називається повторним підтвердженням (DACK).

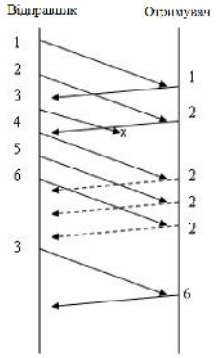

Рисунок 3.3 – Відкидання пакетів

Одержувач продовжуватиме надсилати повторювані підтвердження з кожним пакетом, що не в порядку, (пакети п’ять і шість на рисунку), доки не отримає потрібний пакет (пакет три).

Відправник зрештою повторно надсилає втрачений пакет (після третього DACK, як видно на рисунку 3.3). Коли одержувач отримує пакет 3, він підтверджує всі отримані пакети, надсилаючи підтвердження для останнього пакета, отриманого в порядку (пакет шість), яке вказує, що він отримав усі пакети до пакета номер шість правильно.

Відправник TCP чекає, поки не отримає три послідовних повторюваних підтвердження, перш ніж вирішити, що сталося відкидання, і повторно відправити втрачений пакет.

Номер три був обраний розробниками контролю перевантажень TCP і був прийнятий як стандарт у більшості версій TCP. Однак вибір кількості повторюваних підтверджень для прийняття рішення про втрату буде значною

мірою залежати від топології мережі та методів маршрутизації, що використовуються в мережі. Інший спосіб, який TCP використовує для виявлення падінь, це механізм тайм-ауту, який буде розглянуто далі.

3.3.4 Механізм тайм-ауту

Час очікування повторної передачі (RTO) для TCP був вперше описаний у RFC 793 Postel. RTO є одним із перших визначених методів відновлення від втрат у TCP через пошкодження пакетів або перевантаження мережі. Він працює наступним чином: коли TCP надсилає пакет, він встановлює локальний таймер для цього пакета, а якщо таймер закінчується до отримання підтвердження для цього пакета, тоді TCP припускає, що пакет втрачено.

Потім TCP повторно надсилає пакет і встановлює вікно перевантаження на мінімально дозволений розмір (один сегмент).

Однак обчислення точного тайм-ауту важливе, оскільки занадто довгі тайм-аути знижують продуктивність TCP, оскільки TCP буде чекати довгі періоди, перш ніж перезапустити передачу знову, а занадто короткі тайм-аути можуть збільшити перевантаження в мережі, оскільки мережа не матиме достатньо часу для вирішення проблеми заторів.

Крім того, для того, щоб TCP міг працювати в середовищах із змінами затримок, TCP використовує RTT для обчислення часу очікування повторної передачі. Розрахувати тайм-аут повторної передачі на основі середньозваженого RTT можна наступним чином:

$$AvgRTT = \alpha \times AvgRTT + (1-\alpha) \times RTT\,. \qquad (3.1)$$

Середньозважене значення використовується для фільтрації раптових коливань RTT та отримання довгострокового середнього. Специфікація TCP рекомендує значення α від 0,8 до 0,9.

RTO приймає значення середнього RTT за умови, що воно знаходиться між верхньою межею в 1 хвилину і нижньою межею в 1 секунду, як представлено в рівнянні 3.2. β – постійне значення, яке використовується як оцінка варіації RTT.

$$RTO = min(1mnt, max[\beta AvgRTT, 1sec]). \quad (3.2)$$

Однак використання таких констант, як β і обмеження в одну хвилину/секунду, виявилося неприйнятним для високошвидкісних і великих мереж, які можуть страждати від затримок, вищих або нижчих за межу однієї хвилини/секунди. Так, Джейкобсон запропонував використовувати динамічний розрахунок варіацій RTT наступним чином:

$$RTT\_Var = \theta \times RTT\_Var + (1-\theta)|RTT - AvgRTT| \quad (3.3)$$

де θ=0,75, як запропоновано Джейкобсоном. Обмеження в одну хвилину/секунду було вилучено з обчислення RTO наступним чином:

$$RTO = AvgRTT + 4 \times RTT\_Var. \quad (3.4)$$

Таким чином, RTO тепер є середнім часом передачі в обидва боки плюс чотири рази середній час в тарнзиті. Вибір 4 у рівнянні 3.4 ґрунтується на результатах, отриманих у реальних мережах.

Однією з проблем під час вибірки RTT є неоднозначність, яка виникає під час повторної передачі (тобто, чи належить підтвердження вихідному пакету чи повторно переданому). Ця проблема вирішується за допомогою алгоритму

Карна, просто відкидаючи будь-які показання RTT під час повторної передачі. Зміни Джейкобсона і Карна були додані, як обов’язкові для впровадження в TCP.

Нарешті, Брейден обговорив реалізацію RTO і запропонував два варіанти. Одним з них є використання черги повторної передачі для зберігання всіх пакетів, які були надіслані, але ще не підтверджені, і коли буде потрібна повторна передача, пакет буде готовий у черзі. Інший варіант пропонує не використовувати чергу повторної передачі, а натомість відтворювати кожен пакет після повторного запиту. Очевидно, що перший варіант полегшить процес повторної передачі, але він потребує більшої буферизації та потужності обробки.

3.3.5 Механізм ретрансляції

Як було вказано раніше, коли TCP надсилає пакет, він встановлює таймер для цього пакета. Якщо підтвердження не отримано до закінчення терміну дії таймера (тобто вийшов час очікування повторної передачі), пакет надсилається знову. Проте було запропоновано швидший спосіб відновлення втраченого пакета. Замість того, щоб чекати закінчення терміну дії таймера, якщо TCP отримує достатню кількість повторюваних підтверджень (зазвичай три DACK), можна сміливо вважати, що пакет відкинуто, і можна повторно надіслати пакет без необхідності чекати, поки закінчиться таймер пакета. Потім відправник зменшує швидкість відправлення, щоб уникнути збільшення перевантажень.

Отже, коли відправник отримує три повторювані підтвердження, він виконує наступні дії:

- уповільнює швидкість надсилання, зменшивши розмір вікна перевантаження до половини його розміру перед втратою пакету;
- повторно надсилає втрачений пакет;
- скидує таймер повторної передачі, щоб дати більше часу повторно переданому пакету досягти одержувача до того, як настане тайм-аут;
- TCP очікує на нове підтвердження, яке підтверджує повторно надісланий пакет і всі пакети, що перебувають у транзиті одночасно.

Також TCP збільшує розмір вікна перевантаження на один сегмент для кожного отриманого повторюваного підтвердження, доки не буде отримано нове підтвердження. Логіка цього полягає в тому, що оскільки дублікат підтвердження повідомляє нам, що один пакет (навіть якщо він не в порядку) був отриманий TCP-одержувачем і, отже, він залишив мережу, то можна безпечно передати інший пакет в мережу і, отже, TCP збільшує вікно перевантаження з кожним дублікатом підтвердження.

Перші три дії називаються швидкою повторною передачею. Третя і четверта дії називаються швидким відновленням і всі вони є частиною механізму повторної передачі в TCP.

Однак TCP може відновити лише один скинутий пакет на вікно даних. Якщо буде відкинуто більше одного пакета, продуктивність TCP різко знизиться. TCP не може відновити кілька втрат, оскільки він виходить з режиму повторної передачі (швидка повторна передача/швидке відновлення) після будь-якого нового підтвердження.

Отже, якщо два або більше пакетів втрачаються з одних і тих самих меж вікна (тобто кілька втрат відбуваються з одного набору пакетів під час передачі), TCP повторно надсилає перший, а потім безпосередньо виходить з режиму повторної передачі після будь-якого нового підтвердження, і це запобігатиме повторній відправці решти пакетів.

Однак TCP може впоратися з цією проблемою, використовуючи тайм-аут повторної передачі. Тому, оскільки деякі пакети втрачені, але ще не підтверджені, TCP чекатиме їх підтвердження, яке не відбудеться, тому настає тайм-аут повторної передачі, а потім TCP знову надішле їх.

Швидка повторна передача TCP передбачає, що один пакет буде втрачено на кожне вікно, і, отже, нове підтвердження після повторної передачі пакету зазвичай вказує, що всі пакети отримано, і, отже, TCP безпечно вийде з режиму повторної передачі. Однак іноді, навіть якщо TCP отримує нове підтвердження, це може бути підтвердженням для частини, але не всього надісланого вікна.

Це підтвердження називається частковим підтвердженням, і TCP не готовий мати справу з ним. Отже, якщо набір пакетів було відкинуто, то тільки перший пакет буде оброблено шляхом швидкої повторної передачі, а решта спричинить тайм-аут.

Однак, хоча механізм тайм-ауту гарантує повторну відправку всіх втрачених пакетів, він також зменшить вікно перевантаження TCP (швидкість надсилання) до мінімуму.

На рисунку 3.4 показано приклад цього сценарію. У цьому прикладі пакети 3 і 5 були відкинуті. Відправник повторно надсилає пакет 3 після отримання 3 DACK, а потім отримує нове підтвердження (часткове підтвердження).

Однак TCP завершує швидку повторну передачу, навіть якщо пакет 5 відкидається і не передається повторно, і він чекає нового підтвердження для пакета 5, яке не відбудеться і в кінцевому підсумку закінчиться тайм-аут. Тайм-аут ініціює повторну передачу скинутого пакета 5, а також непотрібну передачу пакетів 6 і 7.

3.3.5 Реакція TCP на втрати передачі

Говорячи про продуктивність TCP в мережах із втратами, більшість літератури спрямовано на провідні, безпровідні, спеціальні та мобільні мережі.

Це пов’язано зі зростанням значення та використання таких мереж, а також тим, що вони стають важливою частиною сучасних мереж. Також це включає будь-яку мережу з каналами низької якості та надійності (хоча основним застосуванням на практиці все ж можуть бути безпровідні та мобільні мережі).

Зниження продуктивності TCP, коли він працює в мережах з втратами, є добре відомою проблемою. TCP використовує помилки як індикатор перевантаженості і, виходячи з цього, знижує швидкість передачі.

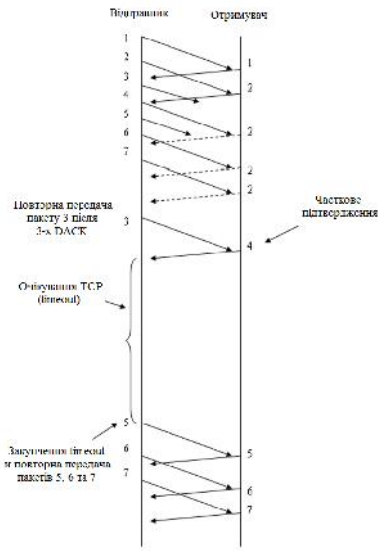

Рисунок 3.4 – Робота TCP з множинніми втратами пакетів

Однак він плутає помилки перевантаження з помилками, викликаними ненадійними каналами, і припускає, що перевантаження виникає щоразу, коли виявляється помилка передачі. Таким чином, TCP знижує швидкість передачі, навіть якщо немає заторів. Таким чином, TCP розглядає обидва види помилок (тобто помилки перевантаження та помилки передачі), як перевантаження. У разі помилок передачі, наприклад, викликаних бездротовими каналами, не потрібно різко знижувати пропускну здатність TCP, як у випадку перевантажень. Багато рішень повторно надсилають втрачений пакет і взагалі не зменшують продуктивність TCP (вікно перевантаження). Механізм контролю перевантажень, який був введений в TCP, відповідальний за виникнення цієї проблеми.

На рисунку 3.5 показано нормальну поведінку TCP, коли немає помилок. Першою фазою є повільний старт з експоненційним зростанням, а потім уникнення заторів (лінійне збільшення). Однак, коли TCP працює через канали, які страждають від помилок без перевантажень, як-от помилки бездротового зв’язку, він зменшує розмір вікна перевантаження, припускаючи, що відбувається перевантаження, як це можна побачіти на рисунку 3.6.

Рівень помилок передачі на рисунку 3.6 становить 1%, і вказано на перші три втрати на рисунку. На обох рисунках вісь Y показує розмір вікна перевантаження в пакетах, а вісь X показує час з’єднання.

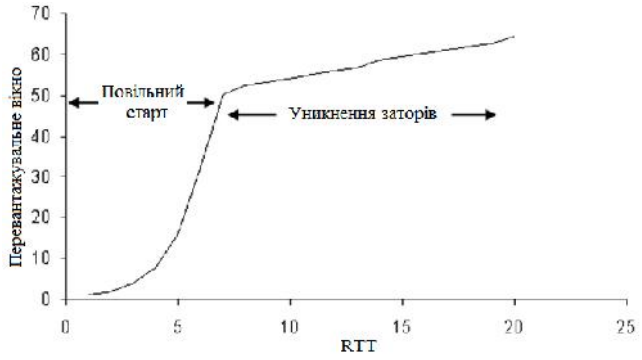

Рисунок 3.5 – Поведінка TCP за відсутності помилок

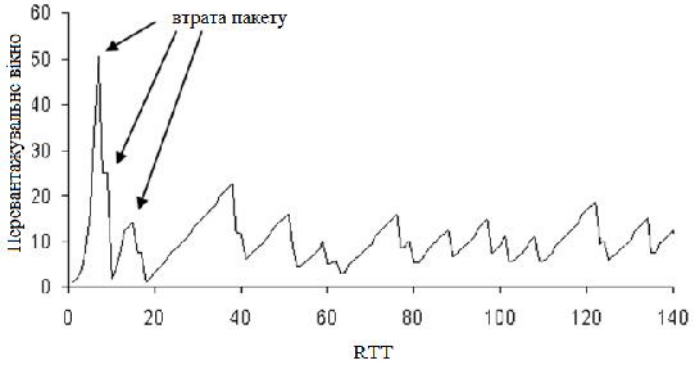

Рисунок 3.6 – TCP з помилками передачі

4 ЧЕРГИ МАРШРУТИЗАТОРІВ

4.1 Поняття черги

Усі пристрої, підключені до мережі, мають один або кілька контролерів мережевого інтерфейсу (NIC), фізичне підключення до мережі. Хоча більшості пристроїв потрібен лише один інтерфейс, деякі пристрої (наприклад, маршрутизатори) мають кілька мережевих інтерфейсів, які підключають пристрій до різних мереж. Коли дані надсилаються з програми, вихідні пакети поміщаються в буфер, перш ніж вони будуть відправлені через NIC в мережу. Кожен NIC має свій внутрішній буфер, а мережевий рівень також має свій власний.

Буфер – це тимчасове сховище (із попередньо визначеним розміром) даних. Основною функцією буфера є згладжування бурхливого потоку даних. На початку розвитку мереж, коли пам’яті було мало, буфери мали обмежений розмір через високі ціни на пам’ять.

Оскільки ціни на пам’ять знижувалися, до пристроїв було додано додаткову пам’ять, що призвело до надмірно великих буферів, які могли вмістити багато даних.

Дані поміщаються в чергу в буфері перед подальшою ретрансляцією. Усі черги викликають певну затримку, оскільки потрібен час, щоб черга опустошилася. Цей час викликає затримку при кожному з’єднанні, що використовує буфер.

Зазвичай черги швидко зникають, коли пакети надсилаються до наступного вузла мережі, але якщо черга утворюється в найповільнішій точці, вузькому місці, потрібен час, перш ніж вона зникне.

Якщо пропускна здатність для вхідних пакетів вища за пропускну здатність вихідних, довжина черги буде збільшуватися, доки буфер не переповниться.

4.2 Як формуються черги

Черги утворюються, коли швидкість надходження даних вища за швидкість передачі даних. Коли швидкість виведення нижча за швидкість введення, пакети накопичуються і створюють тимчасову чергу.

$$L = \lambda W. \tag{4.1}$$

Визначення поведінки в черзі можна побачити в законі Літтла (рівняння 4.1), який визначає довжину черги L шляхом множення швидкості надходження λ на середній час W, коли кожен елемент залишається в черзі.

$$W = L/\lambda. \tag{4.2}$$

Середній час перебування елемента в черзі визначається рівнянням 4.2.

Існує два типи черг: «хороші» черги та «погані» черги. Вони відрізняються тим, як черга впливає на продуктивність мережі. Хороша черга – це черга, яка спустошується за один RTT. Після одного RTT черга зникає, і буфер готовий отримати більше даних. Хороша черга не впливає негативно на RTT, оскільки черга не створює суттєвої затримки для інших з’єднань, будь то трафік в реальному часі чи передача великих файлів, і зникає після одного RTT.

Погана черга – це черга, яка не спустошується після одного RTT. Коли утворюється погана черга, перед тим, як черга спорожниться, надходить більше пакетів. Таким чином, черга обмежує трафік і створює затримку. Причиною поганих черг може бути неправильно розраховане вікно перевантаження в потоці TCP, який надсилає дані. Якщо пакети затримуються в буфері після кількох RTT, це визначається як погана черга. «Стояча черга» — це інша назва «поганої черги». Постійна черга – це черга, яка не спорожняється, коли відбувається відкидання пакета.

Погана черга може виникнути, коли вікно перевантаження TCP більше, ніж фактичний «розмір каналу», кількість даних, яку може передати з’єднаня.

Через те, як працює алгоритм уникнення перевантажень TCP, погана черга не обов’язково виявляється, оскільки вона створює затримку, а не вказує на перевантаження, як показано в наступному прикладі.

На рисунку 4.1 на показано шлях із вузьким місцем. Найшвидший канал має пропускну здатність 100 Мбіт/с і односторонню затримку 1 мс, а повільніший канал має пропускну здатність 10 Мбіт/с. Під час запуску з’єднання вікно перевантаження у відправника обговорюється як 25 пакетів, а не 20 пакетів, що є розміром каналу зв’язку, і перше вікно перевантаження надсилається та ставиться в чергу в буфері вузького місця.

На рисунку 4.2 пакети надсилаються через вузьке місце до одержувача. Після одного RTT відправник отримує перший ACK і повільний запуск TCP зупиняється, це призводить до зменшення швидкості зростання вікна перевантаження. ACK приходять із затримкою, оскільки пакети передаються з буферу.

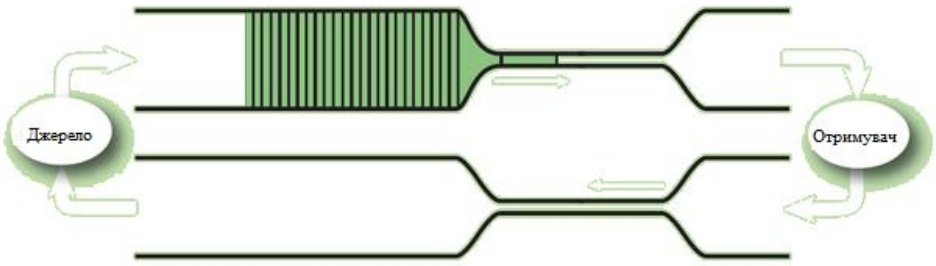

Рисунок 4.1 – Запуск TCP-з’єднання

Коли ACK отримано, алгоритм TCP надсилає наступний пакет у вікні перевантаження на передачу. Час між пакетами достатньо довгий, щоб пакет покинув буфер. Оскільки вікно перевантаження більше, ніж «розмір труби», у буфері вузького місця накопичується черга. Оскільки нові пакети

надходять, буфер матиме постійну чергу, яка продовжуватиме зростати через постійно зростаючий розмір вікна перевантаження. Це також відомо як «черга».

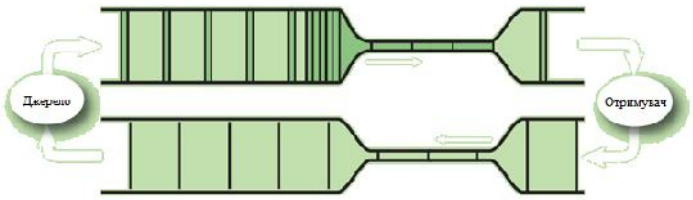

Рисунок 4.2 – З’єднання після одного RTT

На рисунку 4.3 показано, як змінюються довжина черги в буфері та розмір вікна перевантаження TCP, коли черга зберігається. Довжина черги – це різниця між розміром вікна перевантаження та «розміром труби», що в крайніх випадках може призвести до значної затримки. Невеликі зміни довжини черги пов’язані з невеликими часовими змінами в відправленні та отриманні пакетів.

На рисунку 4.4 показано «гарну» чергу, черга повністю зникає до надходження нових даних.

На рисунку одержувач надсилає ACK для кожного розміру вікна, що є крихким методом, але він працює, оскільки відправник надсилає все вікно перевантаження, коли надходить ACK.

4.3 Планування

Планування – це інша частина обробки черги. Планувальник переміщує вхідні дані в праву чергу. Найпростіший варіант планування – перемістити кожен пакет в одну чергу. Використовуючи планування, можна створити розширені механізми черги, коли різні типи трафіку отримують індивідуальну обробку.

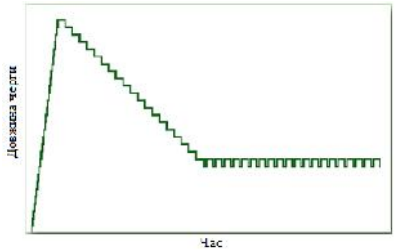

Рисунок 4.3 – Довжина черги залежно від часу

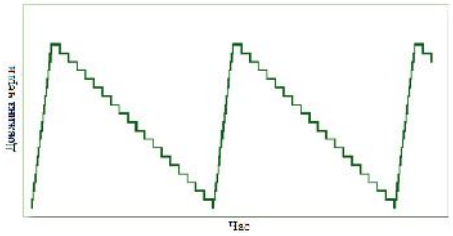

Рисунок 4.4 – Довжина черги при відправці одного підтвердження на кожне вікно перевантаження

Завдання планувальника – організувати дані в черзі буфера. Існує дві основні групи планувальників, пасивні та активні. Пасивне планування досягається, коли планувальник не реалізує багато логіки. Більшість пасивних планувальників використовують або лічильник, який визначає, до якої черги спрямовуються дані, або не реалізують форму для логіки. Планування на основі лічильників може бути засноване на цикловому порядку, де черги обходяться в круговому порядку. Щоразу, коли надходить новий пакет, активна черга перемикається.

4.4 Дисципліна відкидання хвоста Drop Tail

Черги з відкиданням хвоста є стандартним методом обмеження черги в сучасних комп’ютерних мережах. Вони є реалізацією на основі FIFO, яка скидає дані, коли черга заповнена, і, таким чином, її легко реалізувати на пристроях з обмеженою потужністю обробки та обмеженою кількістю

доступної пам’яті. Сучасні пристрої створюють черги, виділяючи доступну пам’ять. Оскільки вихідна пропускна здатність обмежує кількість переданих даних за певний інтервал часу, мережеві пристрої повинні поставити вихідні дані в чергу.

Коли черга заповнена, дані, що надходять, відкидаються. Це змушує відповідні з’єднання знижувати швидкість надсилання пакетів. Відповідні з’єднання виявляють, коли відбувається відкидання пакета. Вони можуть бути реалізовані за допомогою UDP, але найпоширенішими алгоритмами є TCP, що дозволяє з’єднанню виконувати відповідні дії, коли виникає перевантаження.

Рисунок 4.5 – Використання смуги пропускання з механізмом Drop Tail

З TCP черга tail-drop змушує деякі з’єднання отримувати більше ресурсів, ніж інші. При нижчому RTT протокол передачі здатний направляти більше пакетів на канал, оскільки кожен ACK надходить протягом короткого часу. Оскільки TCP надсилає дані лише тоді, коли отримує підтвердження, обсяг виділеної пропускної здатності вищий для з’єднань з низьким RTT.

Черга tail-drop ігнорує програми в реальному часі з метою низької затримки. Програми в режимі реального часу надсилають невеликі обсяги даних через регулярні проміжки часу, і вони зазвичай не відповідають на сторону відправника. Передачі в режимі реального часу залежать від низької затримки та низької зміни швидкості надсилання, щоб забезпечити найкращу

якість для користувача. Оскільки такі черги скидають пакети, коли черга заповнена, а передача в реальному часі може не реагувати, одержувач може відчувати падіння продуктивності або часті затримки.

Черга з максимальним розміром 50 пакетів спричинить затримку на 60 мс при з’єднанні зі швидкістю 10 Мбіт/с, коли черга заповнена. Людський мозок здатний обробляти безперервний потік зображень у плавні анімації, коли зображення змінюються щонайменше 24 рази на секунду. Фільм у кінотеатрі має нижчу частоту 24 герца, затримка між зображеннями близько 40 мс. Якщо пакет може містити зображення, повна черга спричинить затримку кадру в 60 мс, що становить приблизно 16 герц або 16 кадрів в секунду, і потік буде сприйматися як поривчастий.

При використанні IP-телефонії цільова затримка становить 20 мс, максимальна – 150 мс. Затримка в 60 мс може бути непомітною, але її слід уникати для сприйняття якості потоку в реальному часі.

Блокування.

Однією з проблем із чергами з відкиданням хвоста є блокування. Блокування відбувається, коли вхідні потоки синхронізовані. Синхронізація відбувається, коли черга заповнена і відбулося відкидання пакетів. Коли пакети відкидаються, потоки реагують, знижуючи швидкість передачі. Потоки з низькими RTT можуть отримати несправедливу частку черги, оскільки потоки з високими RTT можуть отримати багато пакетів, які скинуто через майже повну чергу.

Глобальна синхронізація.

Проблема глобальної синхронізації впливає на всі черги з відкиданням хвоста, оскільки відкидання відбуваються частіше, коли черга заповнена. Кожне з’єднання, яке зустрічає повну чергу, втрачає пакети та зменшує швидкість передачі.\

Оскільки майже всі пакети відкидаються, кожне з’єднання одночасно знижує швидкість передачі, внаслідок чого пропускна здатність різко падає, оскільки дані не передаються. Завдяки протоколу TCP кожне з’єднання

збільшує швидкість надсилання одночасно, що призводить до повної черги та відкидання пакетів. Таким чином чергуючи порожні та повні черги, збільшуючи проблему.

Рішення глобальної синхронізації полягає в тому, щоб випадковим чином викидати пакети з черги або застосувати політику фронтального скидання. Політика переднього відкидання скидає пакети з початку черги, а не з кінця. Таким чином, це не призводить до глобальної синхронізації, оскільки не кожне з’єднання отримує відкинуті пакети одночасно.

Повні черги.

Ще одна проблема з чергами tail-drop – це проблема повних черг. Це проблема для всіх алгоритмів, які використовують дисципліну tail-drop. Черга повинна мати можливість обробляти пакети в трафік, оскільки пакети мають тенденцію надходити групами до маршрутизатора.

Майже порожня черга здатна обробляти пакети, не скидаючи великої кількості пакетів, тоді як повна черга або майже повна черга повинна скидати пакети, коли вони надходять. Це призводить до великої кількості скинутих пакетів, меншого використання пропускної здатності, глобальної синхронізації та блокування. Черга в неповному стані є добре працюючою чергою через її здатність поглинати пакети даних. Довжина черги не повинна відображати кількість даних, яку вона може вмістити, а радше кількість пакетів, які вона здатна поглинати. Розмір черги повинен бути пропорційним вихідній пропускній здатності пристрою. Більш висока вихідна пропускна здатність має перетворюватися в чергу, яка може поглинати більше даних. Але проблема повної черги все одно залишиться.

4.5 Дисципліни черг

Комбінацію планувальника та політики відкидання часто називають дисципліною черги (qdisc). Існує кілька доступних qdisc, найпоширенішим є FIFO, який реалізує одну чергу з tail-drop. Існує два класи qdisc:

- дисципліна безкласової черги;
- дисципліна класових черг.

Безкласові qdisc не реалізують багато логіки в плануванні.

FIFO qdisc реалізує одну чергу за допомогою простого tail-drop. Перші пакети, які стоять у черзі, є першими пакетами, які будуть надіслані. Коли черга заповнена, вхідні пакети відкидаються.

Стохастична справедлива черга є більш просунутим qdisc. Вона має планувальник, який впорядковує пакети в черги на основі інформації про з’єднання (наприклад, порт відправника та IP-адресу та порт одержувача та IP-адресу). Черга, в яку надсилається конкретний пакет, залежить від виводу функції хешування.

Кожна черга є базовою чергою FIFO з tail-drop. Коли черга заповнена, нові вхідні пакети відкидаються.

Вихідні пакети мультиплексуються в потік за допомогою round robin, який вибирає кожну чергу послідовно. Для кожної черги перший пакет видаляється з черги і мультиплексується у вихідний потік даних.

5 УПРАВЛІННЯ ЧЕРГОЮ

Керування чергою – це простий спосіб обмежити та зменшити потік трафіку. Існує багато способів обмежити потоки, починаючи від черг із відкидання хвоста і закінчуючи активним керуванням чергою. Використовуваний метод залежить від апаратного забезпечення, розгорнутого в мережі. На початку створення мереж філософія полягала в тому, що проміжне обладнання повинно виконувати якомога менше обробки, оскільки доступні ресурси були обмежені, наприклад, потужність обробки та ємність пам’яті. З роками потужність обробки й пам’ять стали дешевими, а маршрутизатори та проміжні мережеві пристрої стали потужнішими, отже, розміри черг зросли та затримують дані на вузьких місцях. TCP завжди збільшує черги, і проблема полягає в тому, як обмежити їх найбільш ефективно.

5.1 Активне управління чергами

Черга tail-drop – це добре відомий менеджер черг, який працює на низькопродуктивних системах з обмеженою пам’яттю. Кожен пристрій, який використовує його, може стати точкою затримки в мережі. Вихідна черга від пристроїв не повинна бути причиною надмірної затримки. Коли черга викликає затримку на кілька секунд, продуктивність буде погіршуватися. Занадто великі черги змушують алгоритм уникнення перевантажень TCP думати, що на зв’язку немає перевантажень, тому вікно продовжує зростати, і більше даних надсилається, оскільки ACK продовжують надходити.

Активне керування чергами є протидією швидкозростаючим чергам. Оскільки сучасні маршрутизатори та проміжні мережеві пристрої мають більшу обчислювальну потужність, ніж будь-коли раніше, вони можуть запускати складніші алгоритми, щоб обмежити довжину черги.

Найперші версії активного менеджера черг (AQM) мали багато погано задокументованих параметрів, які потрібно було статично налаштувати, щоб AQM працював правильно. Якщо конфігурація параметрів працювала ідеально для деяких випадків, інші випадки могли призвести до поганої роботи AQM, що ускладнило роботу для тих, хто хоче правильно налаштувати AQM для кількох випадків використання.

5.2 RED

У спробі виправити перевантажені мережі та оптимізувати продуктивність було створено AQM випадкового раннього виявлення (RED). Алгоритм RED був розроблений для роботи на проміжних маршрутизаторах, на відміну від алгоритмів уникнення перевантажень, які реалізуються в кінцевих точках і реагують, коли канал стає перевантаженим.

RED використовує середню довжину черги для визначення ймовірності відкидання пакетів.

Імовірність відкидання збільшується, коли довжина черги досягає заданого значення. Коли відбувається відкидання, черга працює як черга з відкиданням, скидаючи пакети з кінця черги. Завдяки дисципліні drop-tail, RED поділяє проблеми з іншими чергами drop-tail. З’єднання з низьким RTT мають перевагу перед з’єднаннями з вищим RTT, оскільки вони отримають більшу частину черги, ефективно придушуючи з’єднання з високим RTT.

Оскільки черга заповнюється, ймовірність відкидання зростає. Коли пакет отримано, алгоритм визначає, чи слід скинути пакет чи ні, використовуючи ймовірність відкидання та поріг відкидання. Поріг відкидання визначає, чи зростає ймовірність відкидання. Черга, яка заповнена на 50%, коли поріг відкидання встановлений на 0, має ймовірність відкидання 50%. Це означає, що половина отриманих пакетів відкидається, а відправник змушений передавати їх ще раз, коли закінчується таймер передачі, або коли отримано підтвердження про те, що деякі пакети відсутні.

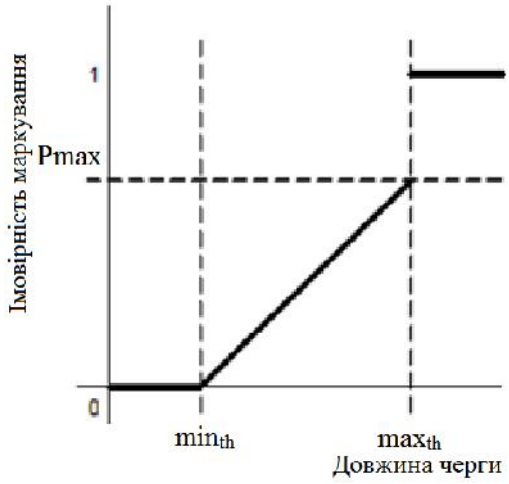

Рисунок 5.1 – Імовірність відкидання пакета алгоритмом RED

Коли RED відкидає пакет, він або справді скидає пакет, або надсилає явне сповіщення про перевантаження (ECN). ECN надсилається відправнику від одержувача, щоб змусити його знизити швидкість. ECN пропонується як заміну скидання пакетів.

RED має багато параметрів, і кожен з них потрібно налаштувати вручну. Це створює труднощі з використанням алгоритму і вимагає від користувачів знати кожну деталь про канали зв’язку, які підключені до пристрою, на якому працює RED. Кожного разу, коли деякі параметри змінюються, будь то RTT, пропускна спроможність або довжина черги, менеджер черг необхідно переналаштувати.

5.3 Компоненти AQM

5.3.1 Індикатор заторів

Кожен AQM має індикатор заторів. Функція індикатора заторів полягає у виявленні можливих заторів. Різні AQM мають різні показники заторів.

AQM на основі швидкості використовують наявну швидкість потоку як індикатор заторів.

Якщо швидкість відправки пакетів на маршрутизаторі падає нижче певного рівня або швидкість надходження збільшується, то черга зростає, якщо $rate_{out} < rate_{in}$. Індикатор перевантаження для AQM на основі швидкості може підтримувати швидкість прибуття на певному відсотку від пропускної здатності каналу.

AQM на основі черги використовують довжину черги як визначальний фактор, коли відбуваються відкидання. Коли довжина черги близька до визначеної цілі, ймовірність відкидання вища, ніж для меншої довжини черги. Ранні весії AQM були засновані на черзі. Індикатори перевантаженості на основі черги спричинять велику ймовірність відкидання, коли черга майже заповнена, навіть коли швидкість надходження нижча за швидкість відправки пакетів маршрутизатором. Це викликає велику кількість відкидання, навіть якщо довжина черги зменшується, що призводить до неоптимального використання пропускної здатності каналу.

У протилежному випадку AQM, заснований на черзі, зменшує частину розміру черги, коли довжина черги мала, навіть якщо вхідна швидкість вища за швидкість відправлення, що призводить до перевантаження.

Індикатори перевантаження на основі затримок використовують загальну затримку черги при визначенні ймовірності відкидання. Вони неявно контролюють довжину черги, відстежуючи кількість часу, який пакет провів у черзі. Деякі AQM використовують заздалегідь визначену кількість часу, щоб контролювати відкидання пакетів. Наприклад, деякі AQM використовують 5 мс як індикатор перевантаження.

5.3.2 Функція контролю заторів

Функція контролю заторів вирішує, що потрібно робити, коли змінюється індикатор затору. Він працює як функція відображення, де значення індикатора перевантаження є вхідним, а вихідним є ймовірність відкидання.

Маркування пакетів, які потрібно скинути, тісно пов’язане з функцією контролю перевантаження. Пакети, які потрібно скинути, визначаються значенням ймовірності. Незважаючи на те, що алгоритм відкидання часто ігнорується (звичайно завжди скидати тільки щойно надійшовший пакет), різні алгоритми можуть мати різні результати для однієї і тієї ж ймовірності відкидання, що спричиняє різницю в довжині черги, втрати та затримки.

5.3.3 Механізм зворотного зв’язку в AQM

Механізм зворотного зв’язку відповідає за надсилання зворотного повідомлення, коли пакет не пересилається його відправнику, коли черга збільшується за задану точку.

Зворотній зв’язок можна реалізувати шляхом відкидання пакетів, і в цьому випадку алгоритм TCP виявляє втрату пакетів (потрійні підтвердження ACK або тайм-аут) або надсилаючи явне сповіщення про перевантаження (ECN). Коли ECN надсилається, пакет пересилається одержувачу, а відправник отримує сповіщення, яке говорить йому знизити швидкість передачі та зменшити вікно перевантаження.

Імовірність відкидання пакета.

Імовірність AQM пакета є центральним компонентом механізму зворотного зв’язку AQM. Імовірність визначає, які пакети будуть втрачені і з якою швидкістю. Імовірність відкидання пакета розраховується по-різному від AQM до AQM.

AQM на основі швидкості використовують вхідну та вихідну швидкість для обчислення ймовірності AQM, тоді як AQM на основі черги визначають ймовірність AQM шляхом вимірювання довжини черги щодо заданої точки. В AQM на основі затримки ймовірність відкидання визначається шляхом обчислення часу, який потрібен для проходження черги пакетом. Якщо розрахований та/або виміряний час перевищує задану точку, ймовірність підвищується.

Відкидання пакета.

Відкидання пакетів – це звичайний механізм зворотного зв’язку, який використовується в комп’ютерних мережах. Коли черга переповнена, пакети відкидаються, і відправник відповідає на втрату.

Явне сповіщення про перевантаження.

Явне сповіщення про перевантаження (ECN) є альтернативою відкидання пакетів. ECN — це позначений пакет, який відповідає відправникові, що він викликає перевантаження, у спробі зменшити розмір вікна. Завдяки позначеному пакету, алгоритму TCP не потрібно чекати втрати пакетів, а розмір вікна має зменшуватися швидше без передачі непотрібних пакетів (які, ймовірно, все одно будуть відкинуті).

Недоліком ECN є те, що він реалізований не у всіх алгоритмах TCP, тому деякі версії можуть ігнорувати ECN. Незважаючи на те, що ECN має допомагати перевантаженим каналам, він виявився менш ефективним, ніж відкидання пакетів, якщо використовується в мережах з високою перевантаженістю через кількість пакетів, які все ще знаходяться в мережі.

Під час перевантаженого стану мережа не може впоратися зі збільшенням пакетів у передачі, і найкращим рішенням може бути відкидання пакетів, щоб усунути перевантаження.

5.3.4 Типи AQM

Існує кілька типів AQM, залежно від того, які критерії AQM використовує для виявлення потенційного переповнення черги. Типи AQM:

- на основі черги;
- на основі швидкісті надходження пакетів;
- на основі навантаження.

AQM на основі черги використовує довжину черги як критерій перевантаження. AQM починає скидати пакети, коли довжина черги зростає до її максимальної довжини; у міру зростання ймовірність скидання зростає.

AQM на основі швидкісті надходження пакетів використовують швидкість вхідного потоку, щоб визначити, коли дані повинні бути відкинуті.

AQM на основі навантаження використовують навантаження пристрою як індикатор перевантаження. Імовірність скидання визначається як $P(drop)=rate_{in}/rate_{out}$, якщо P(drop) перевищує 1 пакет, що стоїть у черзі, то ймовірність скидання має збільшитися.

5.4 Контроль заторів

На кінцевих точках підключення реалізовані стандартні алгоритми контролю перевантажень. З’єднання, що використовують контроль перевантаження кінцевої точки, є реактивними, тому контролер реагує після того, як канал перевантажується.

Час реакції відповідає довжині черги на вузькому місці. Якщо максимальна довжина черги становить 1 МБ, а вузьке місце має пропускну здатність 10 Мбіт/с, контроль перевантаження буде використовувати до 8 секунд для реакції.

Оскільки AQM працює на проміжних пристроях, він може виявити можливу чергу заздалегідь, будучи активним. Якщо черга ось-ось досягне рівня, при якому продуктивність мережі постраждає, AQM може сигналізувати відправнику знизити швидкість. Сигнал може бути відправлений різними способами, або шляхом відкидання пакетів, або шляхом відправки ECN, якщо він підтримується алгоритмом контролю перевантажень.

Якщо AQM має кілька черг для активних з’єднань, він може легко обмежити деякі черги, коли вони стають великими, а інші залишаються недоторканими.

AQM скидають пакети з заданою ймовірністю. Якщо канал стає перевантаженим, ймовірність падіння збільшується, якщо канал має більшу ємність, ймовірність падіння зменшується.

5.5 CoDel

CoDel — це AQM на основі черги, розроблений з наступними цілями:

- немає параметрів - немає регуляторів, які б регулювати оператори, користувачі чи розробники;
- трактує «хорошу чергу» і «погану чергу» по-різному, тобто зберігає низьку затримку, дозволяючи деякі сплески трафіку;
- контролює затримку, будучи нечутливим (або майже так) до затримок приймання-передачі, швидкості з’єднання та навантаження трафіку, ціль полягає в тому, щоб «не нашкодити» мережному трафіку, контролюючи затримку;
- адаптується до динамічно змінних швидкостей зв’язку без негативного впливу на використання каналу;
- є простим та ефективним (може легко охоплювати спектр від точок доступу на базі Linux і домашніх маршрутизаторів низького класу до високоякісних комерційних маршрутизаторів).

Вимірювання довжини черги є неявним, оскільки алгоритм CoDel вимірює затримку черги.

Якщо затримка черги досягає точки, яка вища за поточну встановлену точку, пакет скидається.

CoDel розрізняє «хороші черги» і «погані черги». «Добра черга» – це черга, яка зникає після одного RTT, коли відбувається відкидання, «погана черга» – це черга, яка залишається постійною навіть після того, як відбулося відкидання пакета.

Як і інші AQM, CoDel має три основні компоненти для контролю довжини черги:

- оцінювач – з’ясовує поточну ситуацію;
- цільова точка – вказує на потрібне значення;
- цикл керування – спосіб переміщення до цільової точки.

5.5.1 Оцінювач

Щоб дізнатися поточну довжину черги в маршрутизаторі, необхідна оцінка. Оцінювач працює, відстежуючи час, який пакет провів у черзі. Якщо черги немає, вхідний пакет надсилається негайно, а час перебування в черзі дорівнює 0. Якщо черга є, час перебування пакета в черзі збільшується.

Щоб визначити, чи формується черга, обчислюється локальний мінімум. Цей локальний мінімум – це мінімальна кількість часу, протягом якого пакет перебуває в черзі. Мінімум оновлюється через часті проміжки часу, щоб гарантувати, що значення не застаріє.

5.5.2 Цільова точка

Це точка, коли AQM виконує дію та скидає пакет, щоб обмежити довжину черги.

Високе використання пропускної здатності є результатом невеликої черги. Якщо черга повністю порожня, використання пропускної здатності зменшується, оскільки більше немає даних для надсилання. Тому дуже маленька черга – це добре.

5.5.3 Цикл керування

Цикл управління – це алгоритм, який визначає, коли пакети вилучаються з черги. Більша частина класичної теорії керування стосується керування лінійними, інваріантними в часі системами з одним входом і одним виходом (SISO). Контури керування для цих систем зазвичай надходять від (добре зрозумілих) пропорційно-інтегрально-похідних (PID) регуляторів. Справа в тому, що черги не є системою SISO, оскільки вони працюють не з одним, а з кількома з’єднаннями. Таким чином, AQM є системою з кількома вводами і багатьма виходами (MIMO), яка обробляє як

вхід, так і вихід різних з’єднань. Оскільки MIMO працює з кількома потоками, відкидання одного пакета з потоку буде недостатнім, оскільки реагує лише потік, який скинув пакет. У системі MIMO буде потрібно скинути один пакет з кожного потоку, який повинен реагувати.

Оскільки різні типи з’єднань у системі MIMO варіюються від невідповідних потоків UDP до потоків TCP, що реагують, AQM потребує політики відкидання, яка працює для обох типів.

Коли виникає постійна «погана» черга, скидання пакету з потоку видалить частину черги в RTT через зменшення вдвічі вікна перевантаження, яке виникає після скидання пакета. У зв’язку з цим найкращим є скидання менше одного пакета на RTT із збільшенням ймовірності скидання пакета. Ця політика зниження градієнта реагує на навантаження на вузькому місці. Імовірність відкидання збільшується до тих пір, поки постійна «погана» черга не зменшиться перед скиданням, коли виникає нова «погана» черга. Інтервал між двома скиданнями зменшується обернено пропорційно квадратному кореню з кількості скидань з моменту переходу в стан скидання. Таким чином, скидання пакетів починається повільно, але з часом скидання пакетів стає більш агресивним. Це основна ідея CoDel AQM.

Оскільки найкраща швидкість для початку відкидання – трохи більше одного пакета на RTT, початкова швидкість відкидання контролера може бути безпосередньо отримана з інтервалу оцінювача.

5.6 PIE

На відміну від CoDel, PIE є AQM, який використовує швидкість вилучення пакетів з черги для визначення ймовірності скидання для вхідних пакетів. Як показано на рисунку 5.2, PIE має такі компоненти:

- випадкове скидання;
- розрахунок ймовірності скидання;
- оцінка швидкості відправлення.

5.6.1 Випадкове скидання

Для кожного пакету, що надходить, PIE випадково приймає рішення про відкидання, намагаючись обмежити довжину черги. Коли PIE вирішує скинути пакет, він скидає пакет з кінця черги.

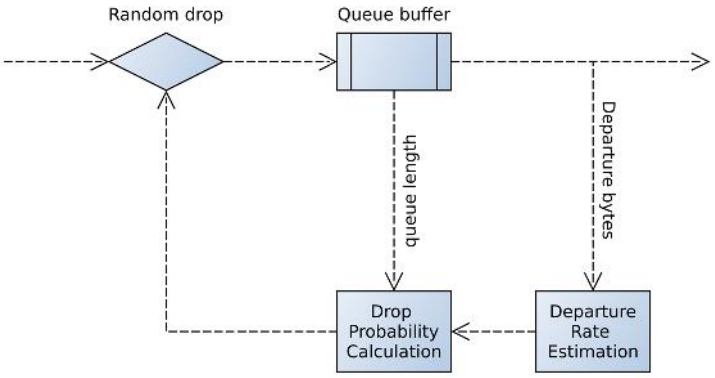

Рисунок 5.2 – Структура PIE

5.6.2 Розрахунок ймовірності скидання

PIE скидає пакети з певною ймовірністю. Ця ймовірність визначається рівнянням (5.1).

$$est\_del = qlen / depart\_rate, \tag{5.1}$$

$$p = p + \alpha \times (est\_del - target\_del) + \beta \times \left(est\_del - est\_del\_old\right), \tag{5.2}$$

$$est\_del\_old = est\_del. \tag{5.3}$$

У рівняннях 5.1–5.3 qlen – це поточна довжина черги, а швидкість відправлення визначається depart_rate, яка отримується з блоку «Оцінка швидкості відправлення».

est_del і est_del_old є змінними, які представляють поточну оцінку та стару оцінку затримки черги, тоді як цільове значення затримки представлено target_del.

5.6.3 Оцінка швидкості відправлення

Вичерпання черги в мережі часто змінюється через різні умови, як-от інші черги, які використовують той самий канал, або коливається пропускна здатність каналу. Коливання швидкості є поширеною проблемою в бездротових мережах.

PIE вимірює швидкість відправлення таким чином:

- якщо в черзі достатньо даних, AQM знаходиться в циклі вимірювання;

- у циклі вимірювання оновлюється поточний підрахунок відправлення;

- якщо кількість відправлень з черги перевищує заданий поріг, розраховується швидкість відправлення.

Швидкість відправлення PIE рекомендується встановити на 16 КБ, припускаючи типовий розмір пакету від 1 КБ до 1,5 КБ. Цей поріг достатньо довгий, щоб розрахувати середню швидкість відправлення, але також досить швидкий, щоб відображати раптові зміни швидкості відправлення.

5.7 FQ CoDeL

FQ CoDel – це AQM, який змішує властивості CoDel з дисципліною FairQueue, яка ставить пакети в різні черги на основі результату алгоритму хешування. Кожна з черг є чергою CoDel, яка намагається обмежити довжину черги, обмежуючи максимальну кількість часу, протягом якого будь-який пакет використовує цю чергу. Якщо пакет використовує більше встановленого ліміту, пакет відкидається або у відповідь надсилається ECN.

Будь-який пакет, який відкидається в кінці черги, може спричинити глобальну синхронізацію, при якій швидкість з’єднань одночасно знизиться, що призведе до можливої втрати зв’язку. Після цього кожне з’єднання намагається збільшити своє вікно, таким чином буфер переповнюється через кількість даних, отриманих одночасно. Якщо синхронізацію не буде вирішено, мережа буде змінюватися від пустої до повної.

FQ CoDel розрізняє «нові» та «старі» черги. Нові черги - це черги, які не мають постійних черг, тоді як старі черги мають. Якщо черга стоїть, вона класифікується як стара, тобто в тому стані, в якому вона знаходиться доти, доки стояча черга не зникне. Коли постійна черга скасовується, вона переміщується зі старого списку до нового списку. Для кожної ітерації певна кількість пакетів переміщується з черги та відправляється.

Якщо в черзі все ще є пакети (що призводить до затримки, що перевищує RTT), черга класифікується як стара.

У кожній черзі FQ реалізовано CoDel, що робить алгоритм ієрархічним. Кожні 100 мс (як описано в розділі 3.10 на стор. 23) перевіряється мінімальне значення, і пакети, що перебувають у черзі понад 5 мс, відкидаються.

5.8 Висновки

У буферах деяких пристроїв з’являються черги. Буфери реалізовані як захід проти агресивних з’єднань. Зі збільшенням дешевої пам’яті буфери пристроїв стають більше. Більші буфери можуть підтримувати більшу чергу, що викликає додаткову затримку для кожного з’єднання, що використовує буфер. Якщо у вузлі, який інтенсивно використовується, існує занадто великий буфер, буфер може підтримувати постійну чергу, яка викликає затримку на кілька мілісекунд або навіть секунд для інших з’єднань. Оскільки збільшення затримки не контролюється TCP, черзі дозволяється зростати. Алгоритм перевантаження TCP зменшує вікно перевантаження TCP, коли виявляє перевантаження.

«Хороша» черга – це черга, яка зникає після одного RTT. «Погана» черга – це черга, яка залишається в буфері після RTT і є результатом погано налаштованого буфера. «Стояча черга» - це інша назва «поганої» черги. Існують рішення – активні менеджери черг (AQM) для вирішення проблеми, але їх часто важко налаштувати, і їх потрібно впровадити та ввімкнути на всіх вузьких вузлах Інтернету, що забирає багато часу.

Перший AQM, RED, було важко налаштувати. Нещодавно були запропоновані інші AQM, які мають менші параметри конфігурації і в основному є в форматі «plug-and-play». Незважаючи на те, що конфігурація зведена до мінімуму, проблема їх розподілу на вузли все ще залишається.

6 УПРАВЛІННЯ ЧЕРГОЮ

Можливим рішенням відсутності активних алгоритмів керування чергою на маршрутизаторах є очищення черги. Очисник черги призначений для роботи на будь-якому мережевому пристрої, який пересилає дані з однієї мережі в іншу.

Якщо виявлено, що черга зростає, програма намагається очистити чергу, відкидаючі непридатні дані, що створють перевантаження буферу. Коли з’єднання відповідають на таку дію, вони зменшують розмір свого вікна перевантаження, що зменшує кількість пакетів в транзиті. Внаслідок цього буфер вузького місця спорожняється перед тим, як нові пакети буде поставлено в чергу на вузькому місці.

На рисунку 6.1 показано мережу з кількома точками, де може утворюватися черга, що викликає затримку. Пристрої в багатьох випадках неправильно налаштовані і мають занадто великі буфери, крім того, різниця в пропускній здатності між вузлами робить проблему ще більшою, оскільки розмір буферів не відповідає поточній пропускній здатності. Через цю невідповідність створюються великі черги, а також страждає швидкість реагування та затримка в мережі.

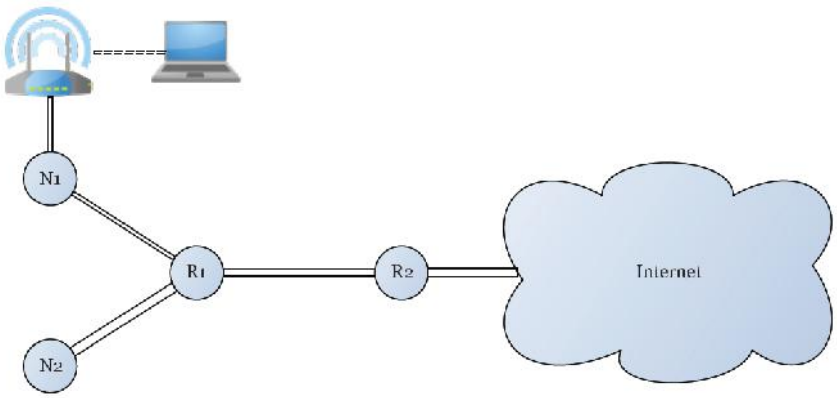

Рисунок 6.1 – Мережа з кількома вузькими місцями

Лістінг 6.1 на наступній сторінці надає просту алгоритмічну версію того, як працює очисник. Для кожного пакета, який він захоплює, очисник перевіряє, чи є пакет TCP-пакетом чи ні. Якщо це не так, пакет ігнорується.

Якщо пакет є пакетом TCP, обчислюється хеш параметрів потоку.

Хеш заснований на IP-адресах джерела та призначення пакетів і номерах портів. Коли хеш створюється, правильний потік знаходить шляхом зіставлення хеша з хешами відомих потоків. Якщо пакет існує в списку хешів пакетів потоку, він ігнорується, інакше його хеш додається до списку.

Якщо захоплений пакет має прапор ACK, встановлений у заголовку пакета, виконується пошук за допомогою інвертованого хеш-ключа. Інвертований хеш обчислюється на основі тих самих показників, що й звичайний хеш, тільки метрики змінюються місцями. Це дозволяє шукати ACK у списку пакетів зворотного потоку (ACK надсилаються від одержувача до відправника). Якщо знайдено пакет, що відповідає ACK, обчислюються як пропускна здатність, так і RTT. Поточний RTT перевіряється за встановленим порогом виміряного базового RTT. Якщо $RTT > threshold \times baseRTT$, то розпочинається заповнення UDP пакетами.

На основі того, як працюють активні менеджери черг (AQM), засіб для очищення черги має набір цілей, які відповідають цілям AQM:

- зменшити час RTT;
- забезпечіти справедливість;
- підвищення пропускної спроможності.

Справедливість досягається шляхом планування, оскільки спосіб адміністрування черги є єдиним способом досягнення справедливості.

Для чутливих потоків, таких як TCP, черга FIFO, яка використовує політику скидання хвоста, надає перевагу потокам з низьким RTT незалежно від того, запущена чи ні.

З’єднання, затримані чергою, не мають покращення або незначного зниження пропускної спроможності в той час як RTT зменшується. Сильне зниження пропускної здатності свідчить про наявність інших потоків.

Лістинг 6.1 – Псевдокод процедури очисника черги

```
program QueueFlusher
begin
  function insert_data_packet(hash,packet)
  begin
    flow=get_flow(hash);
    if packet_exist(flow,packet) return;
    add_packet(flow,packet);
  end
  function check_ack(hash,packet)
  begin
    if not ack return;
    flow=get_flow(hash);
    data=get_data_packet(flow,packet);
    packet.rtt=calculate_rtt(data,packet);
    if packet.rtt>X*basertt flood( );
  end
  while next packet
    if packet.type not tcp continue;
    hash=calculate_hash(packet);
    insert_data_packet(hash,packet);
    check_ack(hash.inv,packet)
  loop
end
```

Очисник черги має бути незалежним як від того, де він розміщений у мережі, так і від того, які протоколи він підтримує. Очисник черги призначений для роботи з чутливими з’єднаннями, зокрема TCP. Очисник черги не відстежує потоки UDP, але будь-який потік UDP, що реагує, повинен реагувати на скидання пакетів, викликане очістником черги.

Щоб мати можливість відстежувати потоки та виявляти затори, йому потрібні такі вимоги:

- вимірювати час RTT;
- вимірювати базовий RTT;
- мати дискретні засоби керування перевантаженнями;
- виконувати очищення черг.

Вимірювання часу RTT потрібне для виявлення черг у мережі з комутацією пакетів, оскільки RTT змінюється відповідно до довжини черги у вузькому місці.

Базовий RTT такий же важливий, як і RTT, оскільки він описує найнижчий виміряний RTT в мережі. Базовий RTT є мірою затримки в мережі, а низький базовий RTT вказує на низький час відгуку в мережі. Оскільки базовий RTT залежить від поточного трафіку, базовий RTT для потоку може бути високим, навіть якщо базовий RTT для цього з’єднання є низьким порівняно з базовим RTT. Це вказує на наявність інших потоків, що використовують той самий шлях. Оскільки очисник не знає цього напевно, він повинен розглядати базовий RTT для поточного потоку так, ніби це поточний базовий RTT в мережі.

Очисник для очищення черги повинен працювати для потоків TCP, використовуючи різні алгоритми уникнення перевантажень.

Можливість очищення черг є важливою вимогою. Якщо чергу не можна очистити, очисник має повторити спробу через короткий проміжок часу.

6.2 Збір даних

6.2.1 Бібліотеки

Існує кілька методів захоплення пакетів, і, мабуть, найвідомішим є libpcap. Бібліотеки збору даних забезпечують простий спосіб захоплення пакетів, оскільки вони забезпечують зрозумілий інтерфейс прикладного програмування (API) для використання програмістами. Без бібліотек програміст повинен використовувати специфічні для ядра API для захоплення пакетів, створюючи таким чином програму, яку важко перенести на інші системи.

Libpcap – це бібліотека, спеціально розроблена для захоплення пакетів із мережі, оскільки вона призначена для забезпечення можливості захоплення пакетів операційним системам на платформі *nix (Unix, GNU/Linux, OS X, FreeBSD, BSD, Solaris...). Є також версія для пристроїв під керуванням

Microsoft Windows, яка називається WinPCap. Такі програми, як Wireshark і TCP Dump, використовують libpcap для збору даних, що робить бібліотеку добре відомою та ретельно перевіреною.

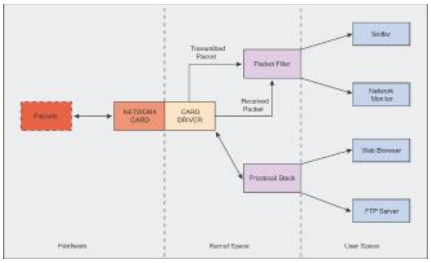

Рисунок 6.2 – Елементи, залучені до процесу захоплення libpcap

Libpcap працює шляхом підключення до ядра, яке повідомляє ядру надсилати копію кожного захопленого пакета до libpcap. Перехоплений пакет може бути пакетом даних, який надсилається, приймається або проходить через пристрій, як показано на рисунку 6.2.

Перехоплені пакети отримують простий заголовок, що містить інформацію про час захоплення пакета та довжину всього пакета, включаючи заголовки. Однією з великих переваг libpcap є можливість писати запити, завдяки чому бібліотека відповідає за повернення правильних пакетів і дозволяє розробнику зосередитися на завданні програмування, не звертаючи уваги на тип пакетів, які повертає бібліотека.

6.2.2 Реалізація

Програма для очищення черги використовує libpcap для захоплення пакетів, оскільки це бібліотека, яка підтримується на більшості платформ і полегшує перенесення програми на інші системи.

Оскільки пакети TCP – це єдині пакети, які цікавлять очищувача черги, він виконує перевірку типу пакета. Якщо захоплений пакет є пакетом TCP, він обробляється програмою, якщо ні, пакет ігнорується. Якщо відомо, що

пакет є пакетом TCP, очищувач звіряє деталі пакета зі своїм внутрішнім списком з’єднань, з яких він захопив пакети. Якщо з’єднання знайдено, пакет додається до списку пакетів, захоплених цим з’єднанням, інакше він додає новий елемент з’єднання та додає пакет до цього з’єднання.

Індекс списку – це хешована функція, яка використовує IP-адресу призначення, IP-адресу джерела, порт призначення та вихідний порт як вхідні параметри для хешу.

Для кожного захопленого пакета створюються два хеш-ключа для пакета один «оригінальний» і один «перевернутий» на основі деталей потоку: IP-адреси відправника та одержувача та використовуваних портів.

Якщо поточний пакет визначено як ACK, ACK шукається у списку надісланих пакетів для з’єднання за допомогою перевернутого ключа, щоб здійснити пошук у зворотному потоці. Якщо знайдено пакет, що відповідає ACK, RTT обчислюється і всі попередні пакети (включаючи той, що відповідає ACK) видаляються, як показано на рисунках 6.3 та 6.4.

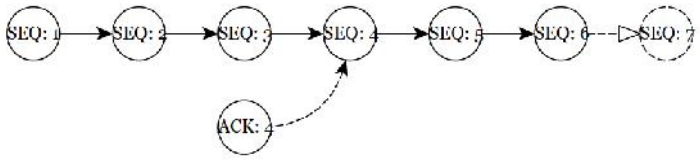

Рисунок 6.3 – Список пакетів перед видаленням

Рисунок 6.4 – Список пакетів після видалення

Незалежно від того, є пакет ACK чи ні, пакет додається до списку захоплених пакетів для потоку відправник-одержувач за допомогою обчисленого ключа. Для кожного захопленого ACK інформація про пакет реєструється для використання під час створення графіків. Якщо захоплений

ACK не має відповідного пакета, вважається, що пакет уже підтверджено, і він видаляється зі списку пакетів. Таким чином, ACK є помилковим і ігнорується.

Лістинг 6.2 – Псевдокод контуру керування

```
const NUM_SAMPLES=10
function controlLoop()
begin
  while(packet=getPacket(nic))!=NULL do processData(packet);
    if (flow=processAck(packet))!=NULL then
      for i=0…NUM_SAMPLES-1 do
        maxThroughput=max(maxThroughput,flow-
>samples[i].throughput);
        maxRtt=max(maxRtt,flow>samples[i].rtt);
        minRtt=min(minRtt,flow>samples[i].rtt);
  loop
# Якщо minRtt перевищує поріг і перевищує 50 мс,
# збільшити лічильник зразків
  if (sec(minRtt)>threshold*sec(flow->baseRtt) and
      millisec(minRtt)> 50) or
      sec(flow ->ackRecvd-flow->lastFlood)>30 then
    flow ->nsamples2++;
  elseif flow ->nsamples2>0 then flow ->nsamples2--;
  endif
  if flow ->nsamples2>2 then
    root->nsamples2=0;
    flood=true;

args={flowRoot=flow,maxThroughput=maxThroughput,maxRtt=maxRtt}
    flood(args)
  endif
endif
loop
end
```

6.3 Розрахунок RTT

Час RTT розраховується при отриманні ACK. Оскільки TCP не обов’язково надсилає ACK для кожного отриманого пакету, очисник черги повинен враховувати це при прийнятті ACK та обчисленні RTT. Коли отримано ACK, відповідний пакет знаходить шляхом узгодження номера

ACK із сумою порядкового номера пакета та довжини пакета, оскільки номер ACK є наступним сегментом байтів, який очікує отримати одержувач, і обчислюється RTT. Для кожного отриманого ACK всі пакети включно до пакета, що відповідає ACK, видаляються, використовуючи тільки наймолодший відповідний пакет при обчисленні RTT.

Наприклад, якщо пакети 1, 2, . . . , 6 надсилаються, ACK для пакета 3 видалить пакети 1. . . 3 і RTT обчислюється шляхом віднімання позначки часу ACK з позначки часу пакета 3, таким чином створюється зразок RTT.

Під час розрахунку RTT використовується експоненціально зважене ковзне середнє (EWMA), яке створює ковзне середнє на основі кожної вибірки. Якщо пакет 3 надсилається кілька наступних разів, що вказує на скинутий пакет, представлення пакета оновлюється новою міткою часу. Це гарантує, що RTT буде правильно розраховано, коли надходить ACK.

6.3.1 Експоненціально зважене ковзне середнє

Експоненціально зважене ковзне середнє відоме зі статистики як метод для обчислення середнього значення на основі кількох значень. Оскільки воно залежить лише від двох змінних і деяких констант, експоненціально зважене ковзне середнє (EWMA) не потребує багато пам’яті для обчислення.

EWMA визначається як у рівнянні 6.1, де α позначає вагу змінних. Сума α+(1−α) завжди дорівнює 1, коли α∈[0,1], тому змінна α позначає швидкість, з якою середнє збільшується або зменшується на основі даних.

$$Y_n = \begin{cases} \mathrm{X}_1 \\ \alpha \cdot Y_{n-1} + (1-\alpha) \cdot \mathrm{X}_n \quad if \;\; n > 1 \end{cases} \tag{6.1}$$

TCP використовує α=7/8, що дає значно більший пріоритет старшим значенням RTT, ніж новішим. Оскільки значення α відоме, засіб очищення черги використовує те саме значення EWMA для обчислення поточного RTT.

6.3.2 Базовий RTT

Для того, щоб очисник знав, коли йому слід спробувати очистити чергу, базовий RTT необхідно розрахувати та контролювати, оскільки умови мережі диктують поведінку RTT.

Якщо з’єднання має базовий RTT 30 мс і він раптово змінюється на 50 мс, очістник може пошкодити пропускну здатність і спричинити додаткові проблеми, але не вирішити їх. Це підкреслює необхідність оновлення базового RTT через встановлені інтервали.

Базовий RTT – це мінімальний RTT, записаний за останній відрізок часу. Для кожного зразка RTT, який нижчий за поточний базовий RTT, він оновлюється.

Як згадувалося раніше, базовий RTT в мережі може змінюватися, але це рідкісна подія. У зв’язку зі зміною шляху в мережі та рідкісним зростанням базового RTT, базовий RTT потрібно оновлювати лише час від часу.

6.4 Розрахунок пропускної здатності

Пропускна здатність обчислюється шляхом підсумовування обсягу даних, надісланих протягом інтервалу, і ділення його на довжину інтервалу в секундах. Розрахунок показано в рівнянні 6.2, де n – кількість пакетів, $payload_p$ – корисне навантаження пакета p у байтах, а interval – встановлений інтервал.

Реалізація пропускної здатності така, як показано в лістінгу 6.3, де корисні навантаження пакетів підсумовуються і діляться на поточний RTT.

Причина поділу кількості байтів на інтервал полягає в тому, що потік і інтервал захоплення не збігаються повністю. З інтервалом в 1 секунду пропускна здатність розраховується, коли різниця між поточною міткою часу та міткою часу на початку інтервалу >= 1, тому вона має оновлюватися, коли

інтервал наближається до 1 секунди, наскільки це можливо. Розрахунок пропускної здатності виконується, коли очисник отримує ACK. Коли канал використовується іншими з'єднаннями, пропускна здатність зменшується, оскільки протягом певного інтервалу надсилається менше пакетів. Причина розрахунку з 1500 байтами полягає в тому, що це максимальний розмір корисного навантаження для кадру Ethernet.

$$\text{throughput} = \sum_{p=1}^{n} \frac{\text{length}(\text{payload}_p) \cdot 8}{\text{interval}} \quad . \tag{6.2}$$

Лістинг 6.3 – Розрахунок пропускної здатності

```
function calc_throughput(flow,item,time_)
begin
  flow->numBytes+=item->totlen;
  double t=sec(clock_diff(time_,root->throughputTime))
  if (t>=1) {
root->throughput=2* root->numBytes*8/t;
root->numBytes=item->totlen;
root->throughputTime=time_;
}
end
```

Лістинг 6.4 – Розрахунок RTT

```
function calc_rtt (rtt,rttSample)
  begin
    rtt=(1-7/8)*rtt+7/8*rttSample;
  end
```

6.5 Очищення

Очищення – це операція, коли програма думає, що виявила чергу, і намагається очистити її шляхом «змивання», щоб збільшити продуктивність мережі, не маючи при цьому великого впливу на пропускну здатність. Коли фіксуються ACK, записується поточний RTT кожного ACK. Коли надходить

ACK, поточний RTT і пропускна здатність додаються до внутрішнього буфера розміром 10. Цей внутрішній буфер є буфером записаних вибірок, і його розмір називається розміром вибірки.

Для кожного записаного зразка лічильник збільшується на 1. Якщо пакет захоплюється під час флешування, RTT і пропускна здатність не додаються до буфера вибірки, а лічильник встановлюється на 0. Коли буфер заповнюється (тобто лічильник повернувся до 0, використовуючи цикловий буфер), збережені значення аналізуються.

Затримка пропускної здатності (bandwidth delay product, BDP) визначає, що посилання може транспортувати задану кількість байтів у будь-який час.

Лістинг 6.5 – Псевдокод процедури флудера

```
function flood (args)
  begin
    currentRoot=args->flowRoot;
    throughput=currentRoot->maxThroughput;
    rtt=currentRoot ->floodRtt||2* currentRoot ->baseRtt;
    currentRoot->lastFloodRtt=currentRoot->lastFloodRtt
                              ||currentRoot->baseRtt;
    multiplier=1.2;
    currentRoot->lastFloodMax=max(currentRoot->lastFloodMax,
                                  item->maxRtt);
    rtt=multiplier*rtt;
//baseRtt<=rtt<=maxRtt
    currentRoot->rttFlood=max(rtt, 2*currentRoot->baseRtt);
    currentRoot->rttFlood= min(rtt,2* currentRoot->maxRtt);
    numberOfPackages=throughput*rtt/(8*1500);
    for p=0 . . numberOfPackages -1 do
       send(args->recipient);
    loop
end
```

За умови, що BDP є постійним, збільшення пропускної здатності призведе до зменшення RTT. Однак, якщо RTT збільшується без зменшення пропускної здатності, то в мережі виникає черга, оскільки пакети накопичуються швидше, ніж вони транспортуються з вузького місця, таким чином надаючи більший RTT, який збільшується з кожним надходженням

ACK. Якщо буфер вибірки (розміром 10) заповнений і найнижчий RTT у буфері вибірки вищий за threshold×base RTT , а найнижчий RTT перевищує 50 мс, лічильник збільшується.

Коли лічильник збільшується більш ніж 2 рази поспіль, очистка активується. Вхідними параметрами є найвища виміряна пропускна здатність і призначення пакетів.

Коли очистка запускається, алгоритм намагається визначити потрібну кількість даних для надсилання, використовуючи найвищу виміряну пропускну здатність і починаючи зі значення RTT 2×baseRTT, щоб обчислити BDP, який використовується як міра кількості пакетів UDP для відправки.

Для кожного наступного змиву значення RTT підвищується на 20%. тобто попереднє значення RTT множиться на 1,2. Верхня межа для значення RTT встановлена 2×maxRTT. Цього має бути достатньо для того, щоб флешер успішно «змив» чергу, навіть якщо деякі пакети скидаються перед проблемним буфером.

6.6 Розміщення очисника черги

Як показано на рисунках 6.5-6.7, в мережі є кілька місць, де можуть виникати черги.

Якщо в мережі N2 є програма, яка передає багато даних або іншим чином створює чергу, черга з’явиться в R1, оскільки це найповільніша точка. На цьому етапі черга вплине на потік даних у мережу N1 та з неї.

Оскільки AQM не реалізовано або не ввімкнено на багатьох апаратних засобах, очищення черги може бути тимчасовим рішенням. Якщо очисник працює на N1 або N2, як показано на рисунках 4.6 і 4.7, слід розглядати чергу в R1, оскільки це вузьке місце, яке ставить пакети в черги та затримує кожен потік, що проходить через нього. Очистка черги зображена ромбом у з’єднанні між каналом та вузлом.

На рисунках 4.5 і 4.6 черга в R1 не обробляється, тому що канал N1-R1 повільніше, ніж N2-R1, це вузьке місце зв’язку знаходиться в N1. Незважаючи на те, що черга в N1 очищена, можлива черга на R1 не обробляється через те, що очисник не виявляє чергу через своє положення в мережі.

Однак, якщо обидва канали N1-R1 і N2-R1 мають однакову пропускну здатність і базовий RTT, вузьке місце буде на R1, і очисник, що працює на N1 або N2, з однаковою ймовірністю виявить і очистить чергу.

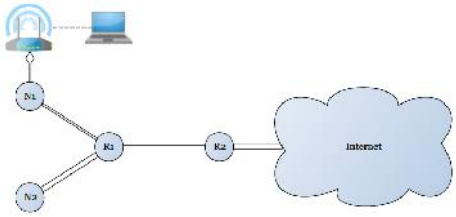

Рисунок 6.5 – Мережа з декількома вузькими місцями та очисником, що працює на маршрутизаторі клієнта

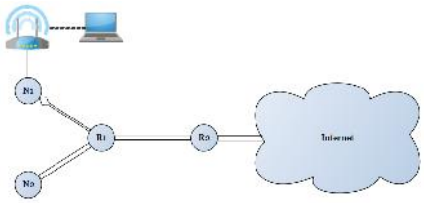

Рисунок 6.6 – Мережа з кількома вузькими місцями та очисником, що працює на N1

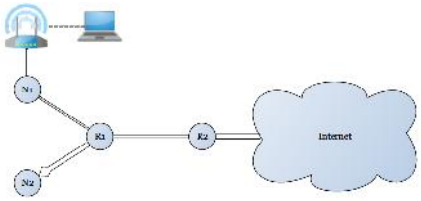

Рисунок 6.7 – Мережа з кількома вузькими місцями та очисником, що працює на N2

6.7 Робота очисника

На рисунках 6.8 та 6.9 показано графіки змін RTT та пропускної здатності очищувача під час запусків, коли він є пасивним та активним.

На графіках, які показують пропускну здатність, є стрибки, які трохи вищі за пропускну здатність мережі. Причиною цих стрибків може бути те, що деякі додаткові пакети були підраховані протягом цього певного інтервалу. Для кожного ACK, де знайдено відповідний пакет даних, обчислюється пропускна здатність. Коли різниця між початком інтервалу та останньою часовою міткою ACK більше або дорівнює інтервалу, починається новий інтервал і пропускна здатність обчислюється на основі суми розмірів пакетів ACK протягом останнього інтервалу. Пропускна спроможність встановлюється як: throughput=bits/time, де біти – це сума розмірів пакетів, що підтверджуються, а час – це різниця в часі між першим і останнім пакетом у інтервалі.

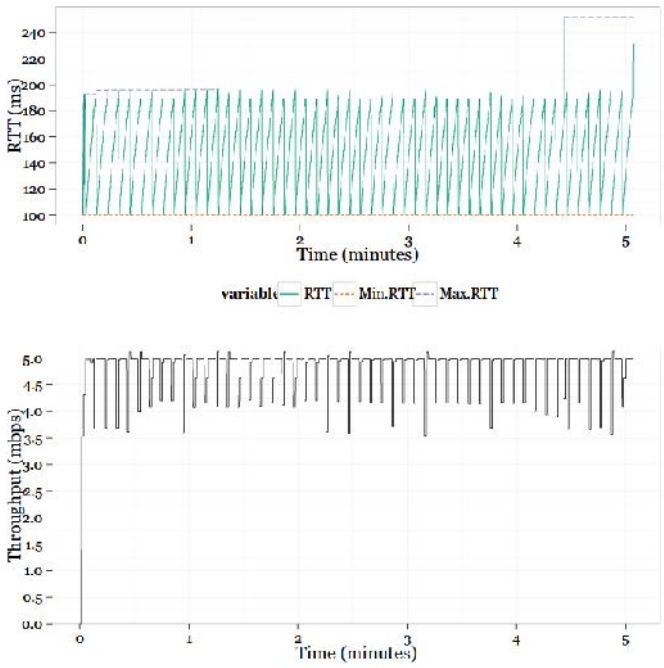

Рисунок 6.8 – Графіки пасивного очисника. З’єднання має буфер на 50 пакетів з базовим RTT 100 мс

Тестування показало, що частота стрибків пропускної здатності підвищується при зниженні базового RTT. Скачки можуть бути наслідком тремтіння, оскільки деякі пакети можуть прийти до одержувача раніше, ніж очікувалося. Якщо така подія станеться, очисник може розрахувати вищу пропускну здатність через більшу кількість ACK, отриманих за короткий період.

Джіттер є природним ефектом у мережах з комутацією пакетів через зміну RTT. Коли пакети стоять у черзі, RTT підвищується. Це спричиняє зміни в міжпакетній затримці та відображається як тремтіння на одержувачі. Потім тремтіння передається на ACK і викликає невеликі варіації в обчисленні пропускної здатності очістника. Тремтіння пропускної здатності показано на рисунках 6.8 та 6.9.

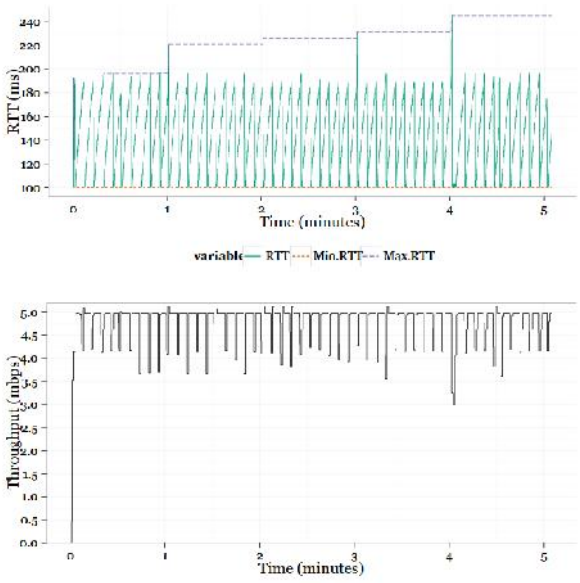

Рисунок 6.9 – Графіки активного очисникка. З’єднання має буфер на 50 пакетів з базовим RTT 100 мс

Джіттер – це зміна затримки між кожним отриманим пакетом. Незважаючи на те, що пакети надсилаються з постійною затримкою, затримка між пакетами може змінитися, коли пакети надходять до адресата. Зміни міжпакетної затримки відомі як тремтіння. Джіттер найбільше впливає

на трафік в режимі реального часу, наприклад на аудіопотоки між учасниками онлайн-чату. Під час передачі великих файлів тремтіння можна не помітити, його ефект найбільш помітний для трафіку, який вимагає високої швидкості реакції мережі.

На рисунку 6.9 показано ефект очистки із занадто великою кількістю пакетів. На 4-хвилинній точці RTT зростає, і слідує короткий період низького RTT.

На жодному з графіків це не часте явище, що показує, що це не обов’язково ефект, який викликає проблеми. Трохи вища пропускна здатність не є проблемою для очисника. Ні скачки RTT, ні стрибки пропускної здатності не повинні негативно впливати на очисник, якщо RTT раптом не має стрибків, які вдвічі перевищують нормальний рівень.

Незважаючи на те, що тремтіння пропускної здатності виникає частіше на з’єднаннях з нижчим базовим RTT, варіації занадто малі, щоб зробити велику різницю.

7 ТЕСТУВАННЯ

7.1 Common open research emulator (CORE)

Існує кілька вимог до того, яким має бути проведено тестування. Компонування тестової моделі:

- клієнтська машина для надсилання даних;
- сервер для отримання даних;
- принаймні один маршрутизатор для маршрутизації даних між клієнтом і сервером.

Має бути принаймні одне вузьке місце, щоб уповільнити потік і створити чергу.

Існує кілька можливих конфігурацій, які відповідають вимогам, і мінімальна топологія, така як та, що показана на рисунку 7.1, є достатньою для тестування та перевірки функцій очищення черги.

Відображена топологія відповідає всім встановленим вимогам, оскільки містить вузьке місце, сервер, клієнт і маршрутизатор.

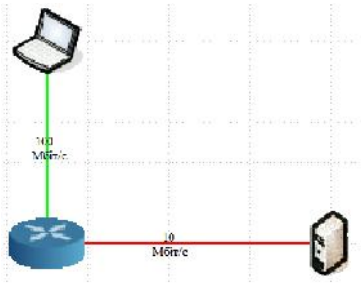

Рисунок 7.1 – Мінімальна топологія

Common open research emulator (CORE) – це набір інструментів для створення тестових моделей, розподілених на одній чи кількох комп’ютерах. Складається з набору інструментів, графічного інтерфейсу користувача (GUI)

для легкого керування віртуальними машинами та шляхами, демона, який спілкується з хостом і встановлює топологію мережі, описану за допомогою графічного інтерфейсу.

Графічний інтерфейс CORE дозволяє легко створювати велику мережу, що містить канали з різною пропускною здатністю. Якщо CORE встановлено на кількох підключених машинах, емульована мережа може охоплювати кожен окремий хост. Це може бути корисно, якщо в емульованій мережі потрібно багато вузлів. Однією з переваг використання CORE є можливість легко працювати з різними технологіями, оскільки CORE здатний емулювати як провідні, так і безпровідні мережі з відключенням і повторним підключенням вузлів. Це корисно для вивчення того, як топологія мережі працює з рухомими вузлами (наприклад, мережа GSM). Також можна підключити деякі вузли до симулятора мережі NS-3.

7.2 Моделювання

Щоб мати можливість контролювати доступну пропускну здатність каналу та затримку, є деякі доступні інструменти. Для Linux дисципліна черги є стандартним способом обмеження пропускної здатності. Деякі qdisc достатньо розвинені, щоб визначати пріоритети різним видам трафіку та затримувати інші. FreeBSD застосував інший підхід, реалізувавши IPFW, брандмауер з функціональними можливостями для регулювання вхідних і вихідних з’єднань.

І qdisc, і IPFW здатні відносно легко обмежити довжину вихідної черги, пропускну здатність і затримку. IPFW є єдиним, який з відносною легкістю також може обмежувати вхідні з’єднання без значних модифікацій, завантажуючи додаткові модулі ядра.

Завдяки можливості працювати з діапазонами IP, IPFW можна налаштувати на затримку трафіку з кількох мережевих інтерфейсів, налаштувавши, які діапазони IP слід використовувати.

Iperf – це кросплатформне програмне забезпечення для тестування продуктивності в мережі за допомогою потоків TCP або UDP. Оскільки iperf може діяти і як сервер, і як клієнт, він містить деякі цікаві властивості. Для TCP він має можливість вибору контролю перевантаження для певного потоку, що робить його хорошим вибором при тестуванні та вивченні різних засобів контролю перевантаження та керування чергою.

Під час налаштування iperf доступно кілька опцій, наприклад, можливість встановити таймер. Коли таймер встановлено, iperf намагається завершити передачу якомога ближче до встановленого часу. Якщо є дані, які не отримали підтвердження, передача триватиме до отримання підтвердження, тому деякі тести можуть тривати довше, ніж встановлений час.

Під час і в кінці передачі iperf реєструє поточне вимірювання пропускної здатності та кількість даних, переданих з початку. Крім того, він може мати кілька потоків, що йдуть від клієнта до сервера, як паралельних, так і зворотних (тобто сервер є відправником, а клієнт – одержувачем).

TCP Dump5 – це утиліта, яка може вивантажувати пакети, які вона захоплює, у файл. Файл складається з необроблених пакетних даних, які захопив TCP Dump. Оскільки TCP Dump використовує libpcap, можна визначити, які типи пакетів він має захоплювати. При використанні в цій атестаційній роботі TCP Dump налаштовано на дамп лише TCP-пакетів.

Synthetic packet (SPP) – це утиліта, яка використовується для обчислення RTT на основі захоплень із TCP Dump. Для розрахунків SPP потребує певної інформації про з’єднання, в основному IP-адреси відправника та одержувача, а також дампи TCP як для відправника, так і для одержувача. IP-адреси використовуються для фільтрації «правильних» пакетів і обчислення RTT підключення.

Починаючи з випуску ядра Linux версії 2.4 є спосіб отримати інформацію про TCP-з’єднання за допомогою його внутрішнього посилання на сокет.

Дані повертаються за допомогою структури tcp_info, яка містить інформацію про розміри вікон, максимальні розміри сегментів, надіслані та отримані підтвердження, RTT тощо.

Спеціальне програмне забезпечення, розроблене в цій атестаційній роботі як спосіб тестування мережі, називається TCP Flooder, оскільки воно надсилає дані так швидко, як це може виконати TCP. Таким чином мережа заповнюється і буфери заповнюються. Деякі функції спільні з iperf, наприклад, можливість працювати як сервер і клієнт, а також настроюваний алгоритм перевантаження. Він також підтримує протоколювання структури tcp_info у файл.

Для кожного надісланого пакета витягується інформацію з сокета і записуєтьсяї у файл. Програма наразі не може надіслати файл або зупинитися після певного часу (яким потрібно керувати ззовні). Часте ведення журналу викликає досить великі файли журналів, якщо TCP Flooder запускається протягом тривалого часу, оскільки він скидає всю інформацію у файл.

Журнали з TCP Flooder зручні кількома способами не лише для реєстрації розміру вікна перевантаження, а й для перевірки правильності обчисленого RTT із засобу очищення черги. RTT, розрахований за допомогою SPP, також додається для подальшого використання.

Розрахований RTT від SPP відрізняється від інших розрахунків RTT тим, що він має більшу частину з високим RTT. Це збільшення високого RTT, ймовірно, пов’язане з тим, що SPP вдається розрахувати більш правильний RTT. Графіки, що показують обчислення RTT від SPP, показують ефект промивання у вигляді великих стрибків на рисунку 7.2.

Технології, що використовуються для тестування, описані в попередніх розділах. Використовуючи хост Linux, на якому розміщується все інше програмне забезпечення, контейнери використовуються як клієнти, підключені до дискретної мережі з власним діапазоном IP, щоб гарантувати, що мережевий трафік не порушується шляхом проходження трафіку з інших мереж.

Контейнерна реалізація – це та, яка поєднується з CORE. Рисунок 7.2: Графіки, що показують розрахований RTT з використанням SPP на додаток до пропускної здатності, розрахованої з тих самих дампів TCP з активним очисником і базовим RTT 100 мс.

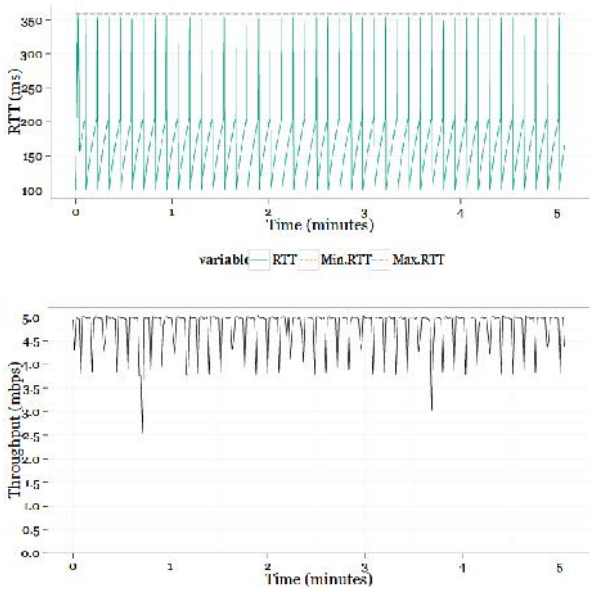

Рисунок 7.2 – Розрахований RTT з використанням SPP на додаток до пропускної здатності, розрахованої з тих самих дампів TCP з активним очисником

7.3 Результати моделювання

Програмне забезпечення, яке використовувалося для реєстрації активних з’єднань, було TCP Dump, а RTT розраховується за допомогою SPP на основі дампів, згенерованих TCP Dump. Крім того, графіки пропускної здатності генеруються тими ж дампами TCP, що й RTT.

Усі запуски виконуються відповідно до сценаріїв з різною мережевою базою RTT. Якщо не вказано інше, графіки показують виконання, коли буфер вузького місця здатний утримувати чергу з максимальною довжиною 100 пакетів (150 КБ). Розміри буфера вузького місця варіюються від 10 пакетів до

100 пакетів з кроком 10 пакетів. Зібрані дані отримані від запусків із такими розмірами буфера вузького місця: 10, 20, 30, 40, 50, 60, 70, 80, 90 і 100 пакетів. Кожен запуск, незалежно від того, чи він був зроблений з активним чи пасивним очисником, тривав 5 хвилин.

Конфігурації мережі, наведені в таблиці 6.1 на наступній сторінці, вибираються для того, щоб створити чергу вузького місця, яку очисник повинен спробувати очистити. Оскільки мережа є «чорним ящиком», засіб очищення черги знає лише, скільки пакетів надіслано і скільки отримано підтверджень. Такі значення, як RTT і пропускна здатність, вимірюються та обчислюються для кожного отриманого ACK.

Таблиця 7.1 – Сценарії

10 Мбіт/с Одностороння затримка	5 Мбіт/с Одностороння затримка	Загальний RTT
20 ms	30 ms	100 ms
10 ms	15 ms	50 ms
2 ms	3 ms	10 ms

Сценарії виконуються із зазначеними параметрами мережі та з різними порогами очистки, щоб визначити, який поріг має найкращі результати щодо RTT та пропускної здатності, а також як працює очисник у порівнянні з існуючими активними менеджерами черг. Функція порогів полягає в тому, щоб визначити, коли змивник повинен почати промивання.

Щоб побачити ефективність очисника черги порівняно з існуючими технологіями, деякі AQM необхідно було протестувати. Вибрані варіанти були CoDel і PIE. Це самонастроювані AQM, що означає, що вони, як правило, працюють нормально, якщо налаштовані зі значеннями за замовчуванням. У особливих випадках їх може знадобитися налаштувати по-різному, але вимірювання тут виконуються зі значеннями параметрів за замовчуванням, оскільки «промивник черги» також використовує адаптивний алгоритм.

Параметрами за замовчуванням CoDel є:

- обмеження черги 100 пакетів;
- цільова затримка черги 5 мс;
- інтервал оновлення 100 мс.

Параметрами за замовчуванням CoDel і PIE є:

- обмеження черги 100 пакетів;
- очікувана затримка черги 20 мс;
- частота оновлення ймовірності падіння 30 мс.

Запуски для вимірювання продуктивності як CoDel, так і PIE були зроблені за допомогою пасивного промивання для вимірювання RTT на додаток до дампу TCP, що виконується у відправника та одержувача. Активний очисник не повинен мати впливу, оскільки RTT тримається під контролем AQM.

Якби очищувач спробував очистити чергу, більшість пакетів було б відкинуто, оскільки активний AQM вступив би в дію.

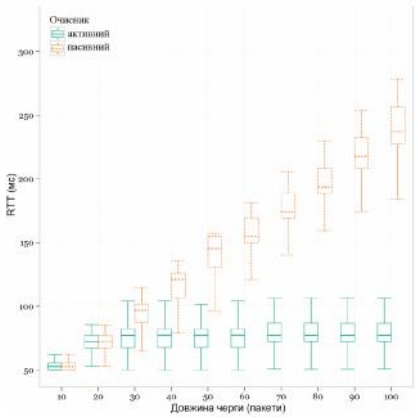

Рисунок 7.3 – Відмінності в RTT для різних довжин черги з використанням базового RTT 50 мс та порогового значення flooder 1,7

Як показано на рисунках 7.4-7.6, очисник виконує належну роботу порівняно з існуючими AQM. Очікується, що очістник не буде працювати так добре, як CoDel або PIE, оскільки він тимчасово збільшує навантаження

на канал, викликаючи стрибки в RTT. Існуючі AQM точно знають, що відбувається всередині пристрою, на якому вони працюють, тоді як очисник повинен вгадати, щоб отримати результат.

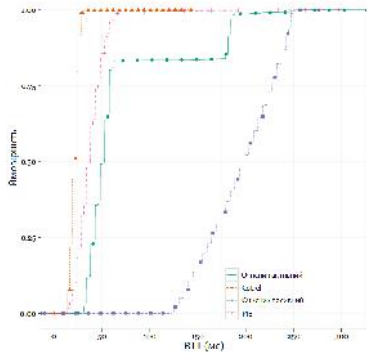

Рисунок 7.4 – RTT з базовим RTT 10 мс

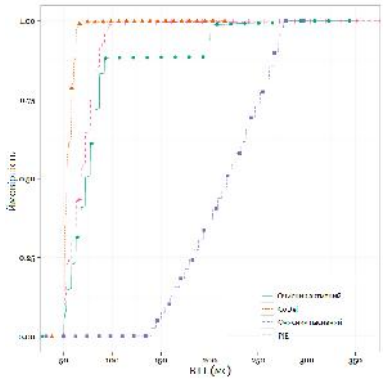

Рисунок 7.5 – RTT з базовим RTT 50 мс

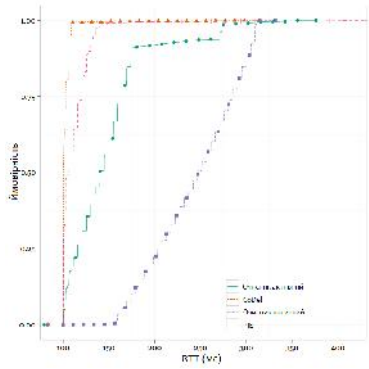

Рисунок 7.6 – RTT з базовим RTT 100 мс

Відображені відмінності RTT вказують на те, що очисник виконує свою роботу належним чином. У кращому випадку очисник знаходиться на одному рівні з PIE AQM.

Коли очисник неактивний, а черга на вузькому місці є чергою FIFO, розмір черги може постійно збільшуватися, перш ніж він переповниться та станеться падіння пакетів. Коли базовий RTT збільшується, різниця між очисником і AQM збільшується. Причин для збільшення може бути кілька, але головною причиною може бути те, що алгоритми TCP реагують повільніше через вищий базовий RTT.

Більш високий базовий RTT впливає, як на час реакції TCP, так і на час реакції очисника. Оскільки очисник вимірює RTT шляхом моніторингу ACK, і він розміщений в іншому місці, ніж вузьке місце, реакція очістника збільшується з базовим RTT. Алгоритми AQM працюють на вузькому місці й знають кожен параметр вузла.

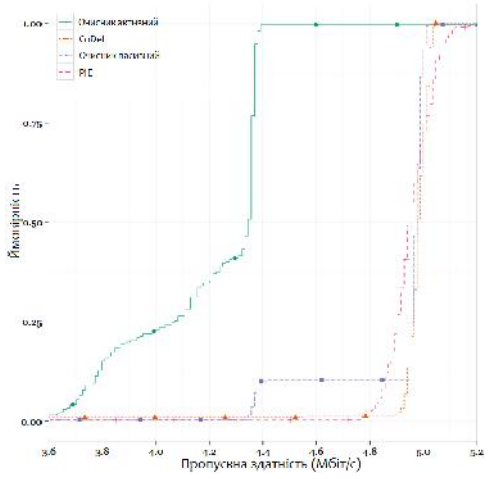

Рисунок 7.7 – Пропускна здатність при RTT 10 мс і L черги 100 пакетів

Наприклад, розмір буфера та час прибуття/відбуття. Вони можуть реагувати швидше, оскільки знають, коли черга заповнює буфер, або коли черга більша, ніж в ідеалі, тому що вони відокремлені від базового RTT з’єднань, що проходять через вузьке місце.

Очевидно, що очисник зменшує пропускну здатність. Цікаво побачити, як пропускна здатність порівнюється з пропускною здатністю, виміряною при використанні AQM. При вивченні графіків стає очевидним, що пропускна здатність, коли CoDel активний, насправді зменшується, коли базовий RTT збільшується.

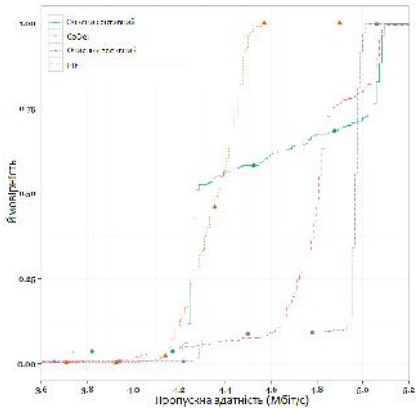

Рисунок 7.8 – Пропускна здатність при RTT 50 мс і L черги 100 пакетів

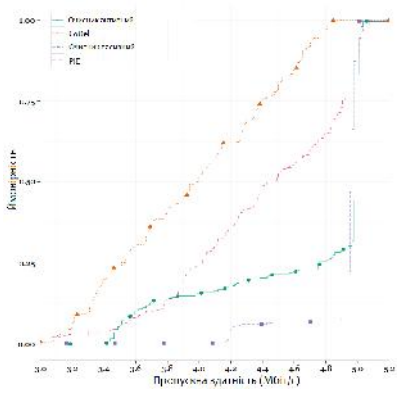

Рисунок 7.9 – Пропускна здатність при RTT 100 мс і L черги 100 пакетів

PIE AQM масштабується краще, оскільки його ймовірність відкидання розраховується шляхом вимірювання швидкості передачі.

Оскільки ймовірність відкидання розраховується на основі швидкості передачі, PIE дозволяє черзі бути такою, якою необхідно для оптимальної роботи.

ВИСНОВКИ

Надмірна буферизація в мережевому обладнанні, яку іноді називають «роздуттям буфера», може спричинити дуже великі затримки для пакетів, що проходять через Інтернет. Це погіршує показники роботи мережі, особливо коли є критичні вимоги для затримки, наприклад, з інтерактивними мультимедійними додатками.

Нещодавно ця проблема була вирішена за допомогою механізмів активного керування чергою, однак такі механізми необхідно встановити на всі пристрої, де виникає проблема. Іноді великі неконтрольовані буфери викликають затримку в пристрої, який знаходиться далеко від користувача, який страждає від проблеми, і, отже, не контролюється цим користувачем.

Представлено просте програмне рішення, яке може принаймні частково покращити ситуацію віддалено: «очісник черги», який відстежує час передачі пакетів RTT і, помічаючи значне та стійке зростання затримок, намагається контрольовано переповнити мережу UDP пакетами, які надсилаються в напрямку, де підозрюється проблема. Результати імітаційного моделювання показують, що можливо помітно покращити середню затримку в мережі за рахунок обмеженого зниження пропускної здатності.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. M. Hassan and R. Jain, High Performance TCP/IP Networking Concepts, Issues and Solutions. Prentice Hall, 2005.

2. H. Elaarag, “Improving TCP performance over mobile networks,” ACM Computing Surveys (CSUR), vol. 34, no. 3, pp. 357–374, 2002.

3. R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, “Hypertext transfer protocol-HTTP/1.1,” RFC 2616, 1999.

4. V. Jacobson, “Congestion avoidance and control,” in Symposium proceedings on Communications architectures and protocols, (Stanford, California, United States), pp. 314–329, ACM Press, 1988.

5. V. Jacobson, “Modified TCP congestion avoidance algorithm,” email sent to end2end-interest mailing list, 1990.

6. G. Xylomenos, G. C. Polyzos, P. Mahonen, and M. Saaranen, “TCP performance issues over wireless links,” IEEE Communications Magazine, vol. 39, no. 4, pp. 52–58, 2001.

7. H. Balakrishnan, S. Seshan, A. E., and R. H. Katz, “Improving TCP/IP performance over wireless networks,” In Proceedings 1st ACM international conference on Mobile Computing and Networking (Mobicom), 1995.

8. J. Postel, “Internet Protocol”, RFC 791, USC/Information Science Institute, September 1981.

9. M. W. Murhammer, O. Atakan, S. Bretz, L. R. Pugh, K. Suzuki and D. H. Wood, “TCP/IP Tutorial and Technical Overview”, IBM Corporation, International Technical Support Organization, Sixth Edition, October 1998.

10. В.В. Петрук, І.А. Волошин, О.А. Янковський, «Методи підвищення пропускної здатності комп’ютерних мереж», Проблеми інформатизації: Матеріали дев’ятої міжнародної науково-технічної конференції. – Черкаси, Харків, Баку, Бельско-Бяла. 2021. –с. 32.