Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра електронних обчислювальних машин
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА

Пояснювальна записка

Рівень вищої освіти другий (магістерський)

Моделі та методи зниження часу передачі даних в комп'ютерних мережах

(тема)

Виконав:
студент II курсу, групи КСМм-21-1
Андрос А.М.
(прізвище, ініціали)

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

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

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

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

Допускається до захисту

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

2022 р.

Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління

Кафедра електронних обчислювальних машин

Рівень вищої освіти другий (магістерський)

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

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

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

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

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Андросу Артуру Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Моделі та методи зниження часу передачі даних в комп'ютерних мережах

затверджена наказом по університету від "7" листопада 2022 р. № 1453 Ст

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

3. Вхідні дані до роботи 1) моделі та методи для керування мережевими інформаційними потоками; 2) сучасні вимоги до мережних показників; 3) перелік використаних програмних та апаратних засобів: ОС Windows 10, OpNet 14, NS-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 – 18 шт.

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

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

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз стану проблеми та сучасних методів її	08.11.22 – 10.11.22	
	вирішення		
2	Огляд технологій управління перевантаженням	11.11.22 – 14.11.22	
3	Розробка моделі управління мережним	15.11.22 – 21.11.22	
	трафіком		
4	Вибір програмних та апаратних засобів	22.11.22 – 28.11.22	
	реалізації		
5	Тестування запропонованого метода	29.11.22 – 05.12.22	
6	Оформлення пояснювальної записки	06.12.22 – 11.12.22	

Дата видачі завдання 7 листопада 2022 р.

Студент _____
(підпис)

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

РЕФЕРАТ

Пояснювальна записка кваліфікаційної роботи: 80 с., 26 рис., 1 дод., 15 джерел.

КАНАЛ ЗВ’ЯЗКУ, МОДЕЛЮВАННЯ, ПЕРЕВАНТАЖЕННЯ, ПІДТВЕРДЖЕННЯ, ПОВІДОМЛЕННЯ, ПРОПУСКНА СПРОМОЖНІСТЬ, СКИДАННЯ ПАКЕТІВ, ТАЙМ-АУТ, ТОПОЛОГІЯ, ТРАНСПОРТНИЙ РІВЕНЬ.

Контроль перевантаження залишається важливою темою для сучасних Інтернет-протоколів. Затори, як правило, шкідливі для користувачів, програм і мереж. Дослідники запропонували кілька механізмів для покращення контролю заторів. Ці механізми включають TCP Tahoe, Reno, Vegas, SACK і NewReno. Досліджуються поточні протоколи контролю перевантажень, враховуючи пропускну здатність, втрати, затримку та справедливість, які забезпечуються TCP.

ABSTRACT

Master’s thesis: 80 pages, 26 figures, 1 appendices, 15 sources.

ACKNOWLEDGMENT, BANDTHROUGH, COMMUNICATION CHANNEL, NOTIFICATION, OVERLOAD, PACKET DROP, SIMULATION, SPIKE, TIMEOUT, TOPOLOGY, TRANSPORT LAYER.

Congestion control remains an important topic for modern Internet protocols. Congestion is generally harmful to users, applications, and networks. Researchers have proposed several mechanisms to improve congestion control. These mechanisms include TCP Tahoe, Reno, Vegas, SACK, and NewReno. Current congestion control protocols are investigated, considering the throughput, loss, delay, and fairness provided by TCP.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

АПВ – алгоритм перевантажувального вікна

АПП – алгоритм повторної передачі

АТПП – алгоритм тайм-ауту повторної передачі

АКП – алгоритми класифікатора помилок

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

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

Cedge – грань затору (англ., Congestion edge)

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

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

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

OSI – взаємозв'язок відкритих систем (англ., Open System Interconnection)

PACK – часткове підтвердження (англ., Partial acknowledgment)

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

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

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

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

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

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

ВСТУП

TCP – це надійний наскрізний протокол, орієнтований на з’єднання. Він містить у собі механізми для забезпечення надійності, вимагаючи від одержувача підтвердження сегментів, які він отримує. Мережа не є досконалою, і невеликий відсоток пакетів втрачається на шляху або через помилки мережі, або через те, що мережа перевантажена, і маршрутизатори відкидають пакети.

Втрати пакетів в каналах мережі є мінімальними, і більшість втрат пакетів пов’язані з переповненням буферів на маршрутизаторах. Таким чином, для TCP стає все більш важливим реагувати на втрату пакетів і вживати заходів для зменшення перевантаження. TCP забезпечує надійність, запускаючи таймер кожного разу, коли він надсилає сегмент. Якщо він не отримує підтвердження від одержувача протягом інтервалу «тайм-ауту», він повторно передає сегмент.

TCP або протокол керування передачею представляє одну з переважаючих «мов» набору протоколів Інтернету, доповнюючи протокол Інтернету (IP), і тому весь набір зазвичай називають TCP/IP. TCP забезпечує надійність передачі даних у всіх наскрізних службах потоку даних в Інтернеті.

Цей протокол використовується основними інтернет-додатками, такими як електронна пошта, передача файлів, віддалене адміністрування та Інтернет. Інші програми, яким не потрібна надійна служба потоку даних, можуть використовувати протокол дейтаграм користувача (UDP), який надає службу дейтаграм, яка наполягає на зниженні затримки, а не на надійності.

1 МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завдання визначення доступної смуги пропускання потоку TCP дуже стомлююче та складне. Складність виникає через ефекти контролю перевантаження як динаміки мережі, так і TCP. Контроль перевантаження – це механізм, який використовується для визначення оптимальної смуги пропускання, у якій пакети мають надсилатися відправником TCP. Розуміння поведінки TCP і підходів, які використовуються для підвищення продуктивності TCP, фактично все ще залишаються серйозною проблемою.

Магістерська кваліфікаційна робота передбачає розробку алгоритмів роботи класифікатора помилок для протоколу TCP для подолання проблеми погіршення продуктивності TCP у гетерогенних мережах, де можуть співіснувати перевантаження та помилки передачі.

Виконання кваліфікаційної магістерської роботи передбачає:

- проведення детального аналізу літературних джерел, присвячених проблемі збільшення пропускної здатності мереж;
- аналіз існуючих механізмів роботи протоколу TCP стосовно боротьби з перевантаженнями;
- розробку методів, які дозволяють поліпшити роботу механізмів перевантажувального вікна, механізму повторної передачі та механізму тайм-ауту.
- проведення аналізу результатів моделювання.

2 ОГЛЯД СУЧАСНОГО СТАНУ ПРОБЛЕМИ

2.1 Перевантаження

За останнє десятиліття використання Інтернет-сервісів різко зросло, і останнім часом Інтернет-програми еволюціонували від стандартної функції пошуку документів до вдосконалених мультимедійних послуг. Однією з головних перешкод на шляху подальшого успіху Інтернету є перевантаження Інтернету, яке у багатьох випадках призводить до неприйнятно тривалого часу відповіді, особливо для додатків у реальному часі.

Мережа може бути перевантаженою з однієї з двох причин. По-перше, джерело високошвидкісного комп’ютера може генерувати трафік швидше, ніж може передати мережа. По-друге, якщо багато джерел із високою швидкістю надсилання одночасно розміщують дані в каналі з обмеженою пропускною здатністю. Коли більше пакетів надходить занадто швидко, ніж хост-одержувач або маршрутизатор для обробки, вони тимчасово зберігаються в пам’яті.

Якщо продовжує надходити такий трафік, приймаючий хост або маршрутизатор зрештою вичерпає свою пам’ять і повинен відкидати додаткові пакети, які надходять.

Коли пакет стикається з перевантаженням, існує велика ймовірність того, що пакет буде відкинуто, і відкинутий пакет втратить дорогоцінну пропускну здатність мережі на шляху від свого відправника до передчасної смерті.

Таким чином, стабільність мережі Інтернет залежить від контролю перевантаження в мережі. Це спонукає до розробки схем контролю перевантажень, які дозволяють користувачеві отримувати користь від мережі або шляхом резервування ресурсів для запобігання перевантаженням, або шляхом реакції на збільшення навантажень мережі.

Контроль перевантажень – це розподілені механізми, які запобігають перевантаженням до їх виникнення або навіть усувають перевантаження. Механізми контролю перевантажень мають дві основні мети: перша – уникнути перевантаження мережі та усунути перевантаження, якщо неможливо уникнути перевантаження, а друга – забезпечити справедливе обслуговування з'єднань. Загалом існує два способи реалізації контролю перевантаження:

- механізми контролю перевантажень за допомогою мережі (підходи, що застосовуються маршрутизаторами);
- наскрізні механізми контролю перевантаження (підходи, використані протоколом керування передачею (TCP)).

Протоколи контролю перевантажень використовують пізніший метод і в основному досягаються на транспортному рівні. Основна ідея контролю перевантаження за допомогою TCP полягає в тому, що відправник TCP завжди намагається заповнити канал між хостами-відправниками та адресатами, а також адаптує швидкість передачі в мережі. Ця адаптація запобігає перевищенню джерелом пропускної здатності мережі, щоб уникнути перевантаження в маршрутизаторі, з'єднаннях або на розширеному хості. Було проведено багато досліджень, щоб досягти стабільної, ефективної та чесної роботи контролю перевантажень TCP. Набір наскрізних механізмів був широко визнаний для підтримки стабільності Інтернету, серед них TCPTahoe [1], Reno[2-3], NewReno[4], sack[5] і Vegas[6].

2.2 Протокол керування передачею

Протокол керування передачею (TCP) є найбільш домінуючим протоколом, який використовується для надійної доставки даних через Інтернет [7]. TCP часто описують як байтовий потік, орієнтований на підключення та надійний протокол транспортного рівня доставки. Він гарантує надійну доставку даних, що надходять від однієї програми до іншої,

хоча базові рівні (IP) пропонують лише ненадійну доставку даних[8-9]. Ця надійність досягається шляхом призначення порядкового номера кожному переданому пакету та очікування позитивного підтвердження (Ack) від одержувача [10-11]. Ці підтвердження зберігають наступні дані, які очікується отримати одержувачем.

Порядковий номер використовується для зміни порядку пакетів, які можуть надійти до одержувача не в порядку, бути дубльованими або пошкодженими. Справді, приймач обробляє пошкоджені пакети, відкидаючи їх, а відправник зрештою закінчує тайм-аут і повторно передає пошкоджені або втрачені пакети. Тоді роль TCP полягає у забезпеченні наскрізних механізмів для забезпечення надійної доставки даних. Спочатку TCP реалізує механізм керування потоком на основі вікна [12]. Грубо кажучи, протокол на основі вікна означає, що поточний розмір вікна визначає сувора верхня межа кількості непідтверджених пакетів, які можуть передаватись між заданою парою відправник-одержувач.

Рання реалізація TCP відповідає простій моделі Go-Back-N, використовуючи кумулятивне позитивне підтвердження та вимагаючи закінчення таймера повторної передачі для повторного надсилання даних, втрачених під час транспортування, без будь-яких потужних методів контролю перевантаження. У моделі Go-Back-N відправник безперервно передає дані до розміру вікна, не чекаючи Ack. Якщо пакети отримані в порядку, без помилок, адресат підтверджує їх, надсилаючи Ack. Але якщо тайм-аут передавача закінчився, він повторно надсилає всі пакети.

2.3 Контроль перевантажень TCP

Стара реалізація TCP запускає з’єднання з відправником, вставляючи в мережу кілька сегментів, до розміру вікна, оголошеного отримувачем. Хоча це нормально, коли два хости знаходяться в одній локальній мережі, можуть виникнути проблеми через наявність проміжних маршрутизаторів і

повільніших з’єднань між відправником і одержувачем. Деякі проміжні маршрутизатори не можуть з цим впоратися, пакети стають у черзі, пакети відкидаються, в результаті повторної передачі продуктивність мережі погіршується.

Сучасні реалізації TCP містять низку алгоритмів для контролю перевантаження мережі, зберігаючи хорошу пропускну здатність. Алгоритми контролю перевантажень TCP запобігають переповненню пропускної здатності мережі відправником. Окрім оголошеного вікна приймача, awnd, контроль перевантаження TCP представляє дві змінні для з’єднання: вікно перевантаження, cwnd, і поріг повільного запуску, ssthresh.

Розмір вікна відправника, w, визначається як w=min(cwnd, awnd), а не дорівнює awnd. Перевантажувальне вікно – це контроль потоку, накладений відправником, тоді як оголошене вікно одержувача – це контроль потоку, накладений одержувачем. Перший ґрунтується на оцінці відправника передбачуваного перевантаження мережі; останнє пов'язане з обсягом доступного буферного простору в приймачі для цього з'єднання [13].

У той час як awnd використовується, щоб запобігти перевантаженню відправником ресурсів одержувача, мета cwnd полягає в тому, щоб запобігти надсиланню відправником більшої кількості даних, ніж мережа може прийняти в поточних умовах навантаження.

Як механізм контролю перевантаження, відправник TCP динамічно збільшує/зменшує розмір свого вікна відповідно до ступеня перевантаження мережі. Таким чином, ідея контролю перевантаження полягає в адаптивній модифікації cwnd для відображення поточного навантаження мережі. На практиці це робиться шляхом виявлення втрачених пакетів, які в основному можуть бути виявлені або через механізм тайм-ауту, або через повторні підтвердження (DACK) [10]. Сучасні реалізації TCP містять чотири переплетені алгоритми як базовий стандарт Інтернету: повільний запуск, уникнення перевантажень, швидка повторна передача та швидке відновлення.

2.3.1 Повільний старт

Повільний запуск працює, спостерігаючи, що швидкість, з якою нові пакети повинні бути введені в мережу, є швидкістю, з якою підтвердження повертаються іншою стороною. Коли встановлюється нове з'єднання з хостом в іншій мережі, перевантажувальне вікно (cwnd) ініціалізується одним сегментом (cwnd = 1). Потім відправник починає з передачі одного сегмента та очікує його ACK. Коли цей ACK отримано, перевантажувальне вікно збільшується з одного до двох, і можна надіслати два сегменти.

Коли кожен із цих двох сегментів підтверджується, перевантажувальне вікно збільшується до чотирьох. Тобто після кожного отриманого ACK значення cwnd оновлюється до cwnd=cwnd+1, що передбачає подвоєння cwnd для кожного RTT.

Це забезпечує експоненціальне зростання cwnd за час проходження в обидві сторони (RTT), хоча воно не зовсім експоненціальне, оскільки приймач може затримувати свої ACK.

Це продовжується до виникнення заторів або досягнення ssthresh (порогове значення повільного запуску). У цей момент з'єднання переходить у фазу уникнення перевантаження. Концептуально, ssthresh вказує правильний розмір вікна залежно від поточного навантаження на мережу, розрахованого на основі кількох перших переданих пакетів.

Попередня реалізація повільного старту виконується лише в тому випадку, якщо інший кінець мережі завжди виконує повільний старт [14]. Повільний старт можна описати так:

Лістинг 2.1 – Реалізація повільного старту

```
Initial:  cwnd = 1;
For (each packet Acked)
    cwnd++;
Until (congestion event, or, cwnd > ssthresh)
```

2.3.2 Уникнення заторів

Щойно розмір перевантажувального вікна (cwnd) перетинає ssthresh, TCP переходить у фазу уникнення перевантаження, щоб гарантувати, що cwnd не збільшується експоненціально завжди за допомогою системи адитивного збільшення мультиплікативного зменшення (AIMD).

Для кожного отриманого підтвердження cwnd збільшується на 1/cwnd (додаткове збільшення), що означає лінійне, а не експоненціальне зростання. Це лінійне збільшення триватиме, доки не буде виявлено втрату пакета.

Лістинг 2.2 – Реалізація фази уникнення заторів

```
/* slow start is over */
/*cwnd > ssthresh */
Every Ack:
    cwnd = cwnd + (1/cwnd)
    Until (timeout or 3 Dacks)
```

Коли виникає перевантаження, відправник TCP повинен уповільнити швидкість передачі пакетів у мережу. Якщо перевантаження вказує закінчення часу очікування, половина поточного cwnd або принаймні два сегменти зберігаються в ssthresh:

$$ssthresh=max(cwnd/2, 2 \text{ сегмент}).$$

cwnd встановлюється в один сегмент і викликає фазу повільного запуску. Крім того, якщо перевантаження вказується 3 дублікатами підтвердження (3 DACK), ssthresh і cwnd встановлюються на половину поточного (мультиплікативне зменшення) і викликають фазу швидкої повторної передачі та швидкого відновлення для виправлення поточного стану мережі.

2.3.3 Швидка повторна передача

Метою механізму швидкої повторної передачі є прискорення процесу повторної передачі, дозволяючи відправнику повторно передати пакет, як тільки він матиме достатньо доказів того, що пакет було втрачено. Швидка повторна передача дозволяє уникнути очікування TCP для повторного надсилання втрачених сегментів.

Це означає, що замість того, щоб чекати закінчення таймера повторної передачі, відправник може повторно передати пакет відразу після отримання трьох дублікатів ACK.

Оскільки відправник TCP не знає, чи дублікат ACK спричинений втраченим сегментом чи просто зміною порядку сегментів, він очікує отримання невеликої кількості дублікатів ACK. Передбачається, що якщо відбувається лише перевпорядкування сегментів, буде лише один або два повторюваних ACK перед обробкою переупорядкованого сегмента, який потім створить новий ACK.

Якщо поспіль отримано три або більше дублікатів ACK, це є серйозною ознакою того, що сегмент було втрачено. Потім TCP виконує повторну передачу того, що є відсутнім сегментом, не чекаючи закінчення таймера повторної передачі.

2.3.4 Швидке відновлення

Алгоритм швидкого відновлення – це вдосконалення, яке забезпечує високу пропускну здатність за помірного перевантаження, особливо для великих вікон. Після того, як швидка повторна передача надсилає відсутній сегмент, виконується уникнення перевантаження, але не повільний старт. Причина невиконання повільного запуску в цьому випадку полягає в тому, що отримання дублікатів ACK повідомляє TCP більше, ніж просто те, що втрачено пакет.

Лістинг 2.3 – Реалізація швидкого відновлення

```
ssthresh = Cwnd/2;
Cwnd = ssthresh +3;
Each DACK received;
Cwnd ++;
Send new packet if allow;
After new Ack:
Cwnd = ssthresh;
Return to congestion avoidance
```

Оскільки одержувач може згенерувати дублікат ACK лише тоді, коли отримано сегмент, цей сегмент покинув мережу та знаходиться в буфері одержувача. Тобто дані все ще передаються між двома абонентами, і TCP не хоче різко зменшувати потік, переходячи на повільний запуск.

3 ПРОТОКОЛ TCP

3.1 Покращення контролю над перевантаженням TCP

Протягом багатьох років було зроблено багато вдосконалень для контролю перевантаження TCP багатьма дослідженнями, що призвело до додавання нових алгоритмів.

3.1.1 TCP Tahoe

TCP Tahoe (4.3 BSD Tahoe) є першою реалізацією, яка обробляє контроль перевантаження. Tahoe TCP додає алгоритми контролю перевантажень для вдосконалення попередніх реалізацій TCP. Ці алгоритми: повільний запуск, уникнення перевантажень і швидка повторна передача. Удосконалення включають модифікацію засобу оцінки часу зворотного проходження, який використовується для встановлення значень часу очікування повторної передачі. За допомогою швидкої повторної передачі після отримання невеликої кількості дублікатів підтвердження (DACK) до того самого сегмента TCP відправник даних робить висновок, що пакет було втрачено, і повторно передає пакет, не чекаючи закінчення часу повторної передачі. Потім відправник встановлює ssthresh на половину поточного cwnd і встановлює cwnd на одиницю та запускає алгоритм повільного запуску.

3.1.2 TCP Reno

TCP Reno зберігає вдосконалення, включені в Tahoe, але змінює операцію Fast Retransmit і включає Fast Recovery. Функція Fast Recovery працює, припускаючи, що кожен отриманий DACK представляє один пакет, який залишив канал (шлях зв’язку). Таким чином, під час швидкого

відновлення TCP-відправник може зробити інтелектуальну оцінку обсягу непідтверженних даних. TCP вводить алгоритм швидкого відновлення після отримання порогового значення DACK, 3 DACK. Зазвичай поріг становить три. Після отримання порогового значення 3 DACK відправник передає один пакет і зменшує його cwnd і ssthresh на половину поточного cwnd, продовжує швидке відновлення до отримання нового ACK. Потім TCP викликає фазу уникнення перевантажень Замість повільного запуску який виконується відправником TCP Tahoe, відправник Reno використовує додатковий вхідний DACK для синхронізації наступних вихідних пакетів.

3.1.3 TCP NewReno

TCP NewReno розширює TCP Reno, щоб враховувати кількість пакетів, відкинутих з одного вікна. Зміна стосується поведінки відправника під час швидкого відновлення, коли отримано частковий ACK, який підтверджує деякі, але не всі пакети. У Reno часткові ACK виключають TCP зі швидкого відновлення, зменшуючи придатне вікно назад до розміру вікна перевантаження. У NewReno часткові ACK не виключають TCP із Fast Recovery. Натомість часткові ACK, отримані під час швидкого відновлення, розглядаються як ознака того, що пакет, який слідує безпосередньо за підтвердженим пакетом у просторі послідовності, було втрачено та має бути передано повторно.

Таким чином, TCP NewReno може відновити кілька втрачених пакетів з одного вікна даних без тайм-ауту повторної передачі.

NewReno залишається в режимі Fast Recovery, доки не буде підтверджено всі дані, які не передані адресату. Іншими словами, TCP NewReno розрізняє часткове підтвердження та повне підтвердження, де пізніший ACK підтверджує всі пакети, які були нерозглянутими на початку періоду швидкого відновлення, а перший підтверджує деякі, але не всі невиконані дані.

Лістинг 3.1 – Реалізація Fast Recovery

```
If (Ack=partial Ack)
  Stay in fast recovery;
Retransmit one packet per RTT;
  If (Ack=full Ack)
Exit fast recovery;
```

3.1.4 TCP SACK

TCP SACK (TCP із вибірковим підтвердженням (SACK)) є консервативним розширенням контролю перевантаження Reno, яке підтримує опцію SACK на обох кінцях. У TCP SACK, якщо пакет відкидається, у вікні отримання створюється діра, коли приймач отримує пакет відразу після діри, а потім приймач надсилає DACK для пакета перед діркою, як це робить TCP Reno.

SACK додає до поля Ack опцію, що містить пару порядкових номерів, що описують блоки даних, отримані не за порядком, з максимальним розміром 40 байт.

Таким чином, один Ack може містити щонайбільше 4 блоки, а через збільшення використання параметра timestamp кількість блоків SACK досягає 3.

Перший блок повідомляє про останню отриману впорядковану послідовність пакетів. Додавання SACK не змінює реалізацію базових алгоритмів контролю перевантажень. Основна відмінність, коли кілька пакетів випадає з одного вікна, полягає в тому, що відправник може вибірково повторно надсилати втрачені пакети залежно від поля опції SACK у Ack.

SACK додає змінний конвеєр до алгоритму швидкого відновлення, що представляє розрахункову кількість пакетів, які є в транзиті, відправник лише повторно передає або надсилає новий пакет, коли значення менше Cwnd.

Канал змінюється таким чином: pipe++ під час передачі пакету або pipe-- під час отримання DACK. Успішна розробка SACK дає йому спеціальний механізм для часткового Ack, який зменшує канал на два, а не на один.

Рішенням для запобігання небажаної повторної передачі є зменшення кількісті даних, які використовуються для представлення блоку SACK, до 32-бітного порядкового номера для першого блоку правого краю, але решта країв: 1-представлено зміщенням від першого, або 2-представлено як зміщення від попереднього призводять до збільшення кількості блоків у варіанті SACK.

3.1.5 TCP Vegas

TCP Vegas розширює механізми ретрансляції Reno. Він динамічно збільшує/зменшує розмір перевантажувального вікна на основі доступної пропускної здатності в мережі. TCP Vegas використовує більш складну схему оцінки пропускної здатності, використовуючи різницю між очікуваною та фактичною швидкістю потоку. Коли мережа перевантажена, різниця між очікуваною та фактичною швидкістю потоку використовується для налаштування розміру вікна. Очікувані та фактичні значення можна сформулювати так:

Лістинг 3.2 – Налаштування розміру вікна

```
Expected = W/ RTTb;
Actual = window size (W)/ RTT;
Diff = [Expected- Actual] * RTTb;
```

RTTb (базовий RTT) – це мінімальний час проходження пакетів в мережі в обидві сторони (туди й назад).

Cwnd постійно оновлюється до:

$$Y_n = \begin{cases} \text{Cwnd+1}; & \text{if Diff} < \alpha \\ \text{Cwnd-1}; & \text{if Diff} > \beta \\ \text{Cwnd}; & \text{else} \end{cases} \quad (3.1)$$

Де α і β на два пороги більше нуля. Оскільки перевантаження мережі у зворотному шляху не повинно впливати на потік у прямому шляху. Покращення TCP Vegas зроблено, щоб подолати зменшення розміру Cwnd. Зменшення викликано збільшенням часу зворотної черги, що, у свою чергу, впливає на фактичну пропускну здатність і збільшення Diff. Раніше було запропоновано розділити виміряний час RTT на час затримки вперед, час прямої черги, час зворотної затримки та час зворотної черги, використовуючи параметр позначки часу tcp, щоб отримати час зворотної черги та оцінити фактичний наступним чином:

Факт= window size/(RTT-$QD_{backward}$); $QD_{backward}$ – затримка в черзі.

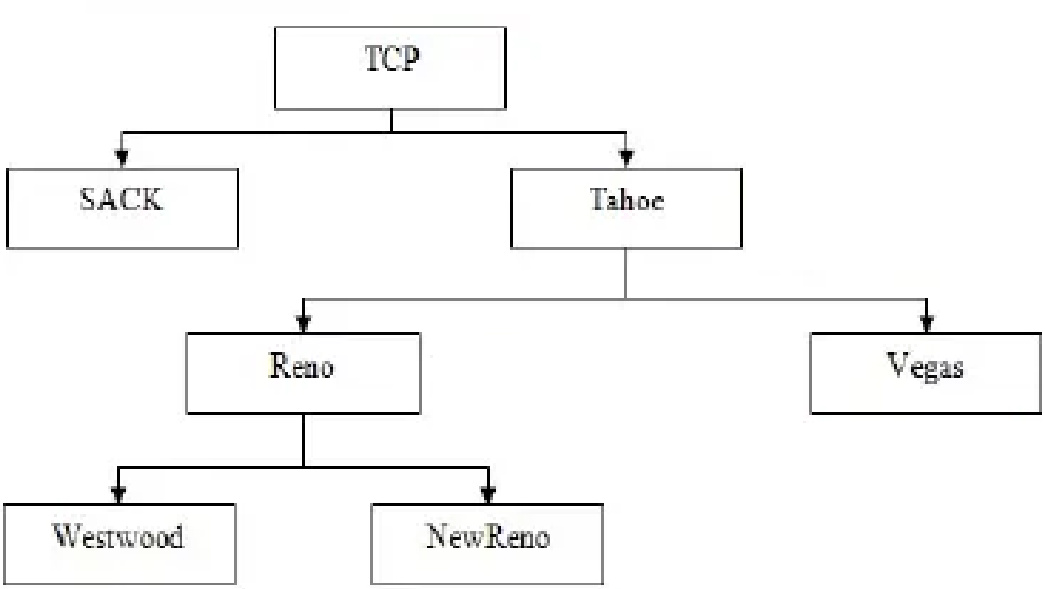

Рисунок 3.1 – Різновиди TCP

3.2 Пропускна здатність

На рисунку 3.2 показано пропускну здатність для кожного варіанту TCP. Використовується сукупна або кумулятивна пропускна здатність, яка розраховується як сукупність отриманих даних за час. Пропускна здатність визначається в кілобітах на секунду (Кбіт/с), де пропускна здатність обчислюється за кожну довжину інтервалу часу Time Interval length (TIL) і ділиться на час.

Результати наведені з використанням TIL=1 секунда (сек). Як показано на рисунку, для всіх варіантів TCP у фазі повільного запуску пропускна здатність зростала експоненціально, а в алгоритмі повільного запуску подвоювалось перевантажувальне вікно з кожним RTT.

Оскільки вхід у фазу уникнення перевантажень залежить від значення ssthresh, яке визначається з перших пакетів потоку в передачі. Можна побачити, що за допомогою TCP Tahoe відправник може повторно передати пакет, який було отримано правильно, що зменшує його пропускну здатність. При вході у фазу уникнення перевантаження розмір перевантажувального вікна лінійно збільшується до втрати пакетів, що сприймаэться як ознака перевантаження. Додавання алгоритму швидкої повторної передачі покращує продуктивність TCP порівняно з його попередньою реалізацією.

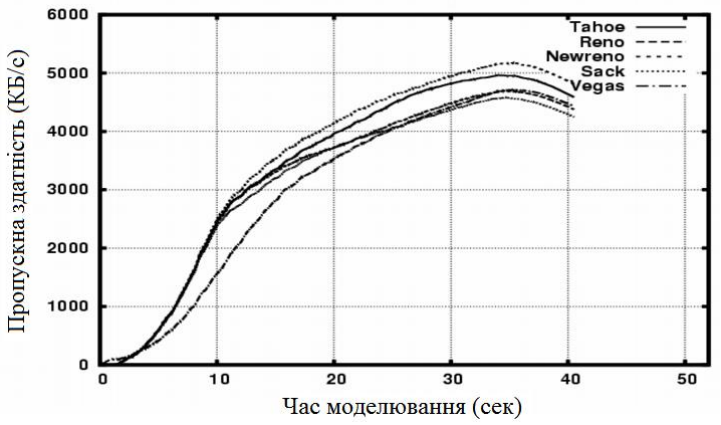

Рисунок 3.2 – Залежність пропускної здатності від часу моделювання

У Tahoe TCP з’єднання завжди починає збільшувати швидкість передачі повільно після втрати пакета. Однак, якщо розмір вікна великий і втрати пакетів трапляються рідко, було б краще, щоб з’єднання продовжилося з фази уникнення перевантажень, оскільки збільшення розміру вікна від 1 до значення ssthresh займе деякий час.

Метою алгоритму швидкого відновлення в Reno TCP є досягнення цієї поведінки. Отже, можна побачити, алгоритм Reno Fast Recovery оптимізований для випадку, коли один пакет скидається з вікна даних. У Reno можуть виникати проблеми з продуктивністю, коли з вікна даних скидається кілька пакетів.

З кількома втратами пакетів TCP Reno по тайм-ауту активує фазу повільного запуску. Як показано на рисунку, пропускна здатність Reno буде меншою, ніж у Tahoe, і це тому, що з кількома втратами пакетів з одного вікна Reno зменшить свою cwnd у більше разів, ніж Tahoe.

TCP NewReno покращено порівняно з Reno, коли кілька втрат пакетів утворюють одне вікно. NewReno збільшує загальну пропускну здатність, ніж Reno. Як показано на рисунку TCP NewReno має найвищу пропускну здатність.

Слід зазначити, що TCP NewReno повторно передає один втрачений пакет за час RTT, доки всі втрачені пакети з цього вікна не будуть повторно передані.

TCP SACK намагається усунути проблему втрати кількох пакетів з одного вікна. TCP SACK додає вибіркове підтвердження та вибіркову повторну передачу, а відправник TCP може виявляти втрату кількох пакетів в одному вікні та вибірково повторно передавати втрачені пакети, навіть якщо додавання опції SACK до підтвердження збільшує навантаження на мережу та потребує покращення у відправнику та одержувачі TCP, щоб увімкнути опцію SACK. На рисунку пропускна здатність TCP Sack вище, ніж у Reno. TCP Vegas намагається збільшити вікно через доступну пропускну здатність. Важливо відзначити, що TCP Vegas намагається уникнути заторів.

Але на рисунку TCP Vegas отримує найнижче значення пропускної здатності, це пояснюється тим, що коли фактична пропускна здатність з'єднання наближається до значення очікуваної максимальної пропускної здатності, воно може не ефективно використовувати буферний простір проміжних маршрутизаторів і, отже, має збільшити швидкість потоку. З іншого боку, коли фактична пропускна здатність набагато менша за очікувану, мережа, швидше за все, буде перевантажена, і, отже, відправник повинен зменшити швидкість потоку (cwnd).

3.3 Втрати пакетів

У мережі більшість втрат пакетів відбувається через затримку передачі, яка має місце в мережевому шляху, і час очікування в чергах маршрутизаторів між двома абонентами. Однак затримка в черзі маршрутизатора є найбільшим фактором, який впливає на загальну затримку. Зі збільшенням затримки зростає ймовірність скидання пакетів. На рисунку 3.3 показано кумулятивну суму всіх відкинутих пакетів, нанесену на графік із часом, де кількість відкинутих пакетів підсумовується кумулятивно, а сума визначається для кожної довжини часового інтервалу (одна секунда на графіку).

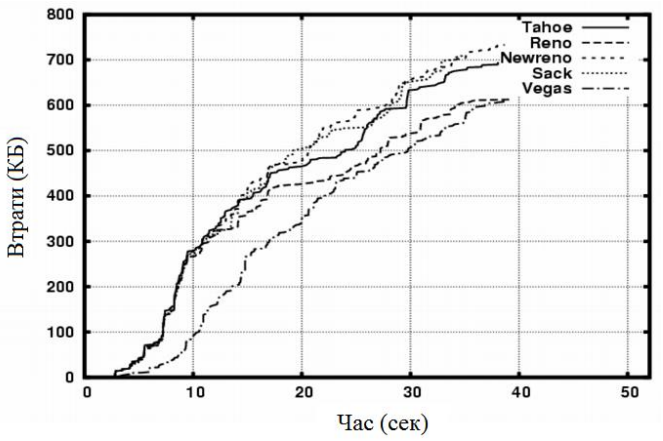

Рисунок 3.3 – Залежність втрат від часу моделювання

На рисунку показано залежність кількості скинутих пакетів від часу моделювання. TCP Tahoe має високу швидкість скидання пакетів.

Існує приблизно рівне значення втрат для всіх варіантів, крім Вегаса, на початку моделювання до часу моделювання 10 секунд. Причина такої поведінки лежить в повільному старту та алгоритмах уникнення перевантажень. З TCP Reno загальна кумулятивна сума втрат пакетів меньша приблизно на 35,7% від Tahoe тому, що швидке відновлення зменшує непотрібну повторну передачу пакетів.

У Vegas TCP загальна сума скинутих пакетів становить приблизно 69,0% від Tahoe, тобто збільшення втрат для Vegas, ніж для SACK і NewReno. У TCP SACK загальна кумулятивна сума відкинутих пакетів дорівнює NewReno, але зміна втрат відрізняється від Reno, де він може відновити втрату кількох пакетів в одному вікні швидше, ніж інший варіант. За допомогою TCP NewReno кількість скинутих пакетів трохи збільшилася порівняно з пакетом SACK, де під час швидкого відновлення NewReno надсилає новий пакет із отриманим дублікатом ACK, щоб заповнити канал, що призводить до збільшення ймовірності скидання пакетів.

На рисунку 3.4 показано залежність затримки від часу для всіх варіантів, зазначається, що Vegas має меншу затримку, ніж усі варіанти, на відміну від Tahoe і Reno, які мають високу затримку.

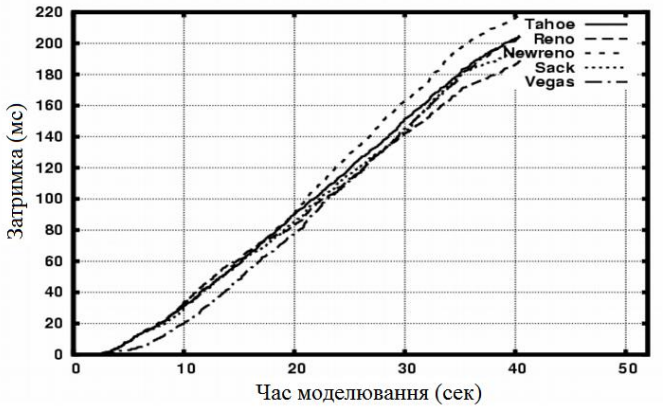

Рисунок 3.4 – Залежність затримки від часу моделювання

Поведінка перевантажувального вікна показана на рисунку 3.5, зміна перевантажувального вікна ілюструє поведінку алгоритмів у кожному варіанті.

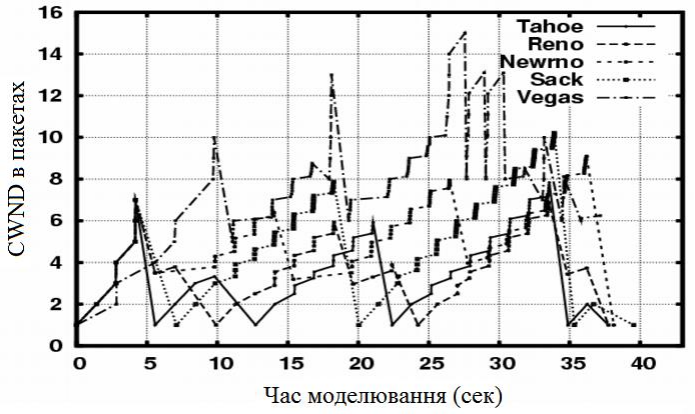

Рисунок 3.5 – Залежність розміру перевантажувального вікна від часу

3.4 Справедливість

Збереження справедливості між кількома з’єднаннями є важливим фактором для мережі. Справедливість досягається, якщо пропускна здатність каналу рівномірно розподілена між спільними з’єднаннями. Має бути наявність справедливої смуги пропускання, спільної між кожним варіантом TCP і UDP (протокол дейтаграм користувача), який не пропонує механізм контролю перевантажень. Пропускна здатність використовується для показу справедливого спільного використання смуги та поведінки кожного варіанту TCP при наявності UDP.

На рисунку 3.6 показано топологію мережі, яка зазвичай використовується при розгляданні справедливості потоків. Результати моделювання поведінки для кожного варіанту TCP і UDP зображено на рисунку 3.7, де показано, що протокол TCP страждає від несправедливого розподілу, коли є в наявності протокол UDP без контролю перевантаження, але кожен варіант має різний рівень спільного використання.

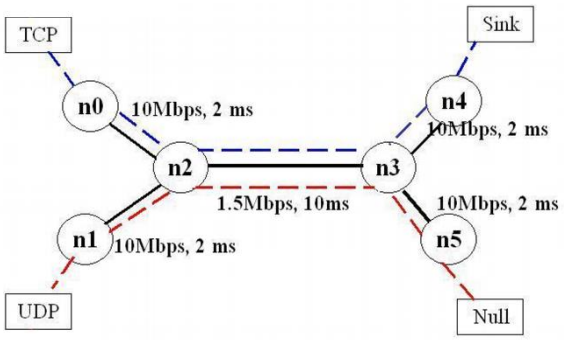

Рисунок 3.6 – Топологія мережі для дослідження справедливості

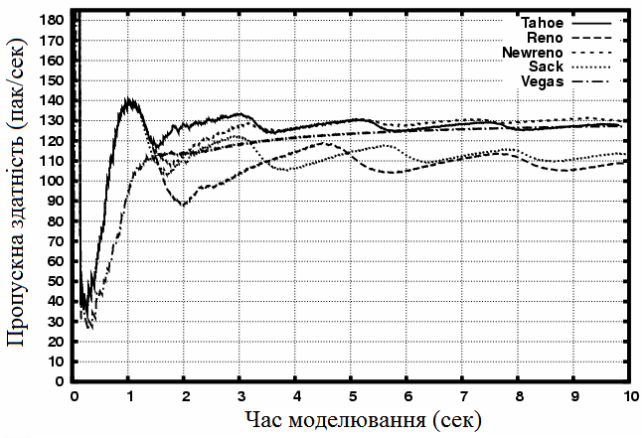

Рисунок 3.7 – Справедливість розподілу пропускної здатності

TCP Tahoe і NewReno (через SACK) мають найвищий рівень спільного доступу, на відміну від Vegas, де наявність UDP погано впливає на його продуктивність.

Vegas виявляє стан мережі та змінює розмір свого вікна, зменшуючи або збільшуючи, що викликає проблему, якщо Vegas ділиться пропускною здатністю з іншим протоколом.

Зі збільшенням навантаження на мережу Vegas починає знижувати швидкість передачі, але цього не робить UDP протокол, що призведе до несправедливого розподілу пропускної здатності.

4 ПОКРАЩЕННЯ РЕАКЦІЇ КЛАСИФІКАТОРА ПОМИЛОК TCP НА ВТРАТИ ПАКЕТІВ

Багато наскрізних пропозицій щодо покращення продуктивності TCP для переривань передачі, зокрема дискримінаторів (класифікаторів) помилок, ґрунтувалися на ідеї, що якщо можна правильно розрізняти помилки, тоді реакція на переривання передачі може бути простою. Отже, основною метою є розробити точний класифікатор помилок.

Це підвищить продуктивність TCP у разі переривань передачі, зменшивши величіну скорочення швидкості надсилання TCP. Також це не збільшить рівень перевантаження через розбіжності класифікатора помилок між типами помилок.

Останній пункт є основною причиною, чому потрібно мати точний класифікатор помилок. Оскільки, якщо класифікатор помилок не точно ідентифікує помилку як помилку передачі, хоча насправді це помилка перевантаження, він не зменшить швидкість надсилання, що призведе до збільшення перевантаження мережі. Однак, наскільки відомо, не існує ідеального наскрізного класифікатора помилок, який міг би розрізняти помилки зі 100% точністю.

Однак, якщо можна підтримувати процес, який не збільшуватиме рівень перевантаження в мережі, тоді навіть класифікатор помилок із середньою/низькою точністю зможе підвищити продуктивність TCP і одночасно уникнути непотрібного перевантаження мережі.

Коли відбувається втрата пакету, це впливає на багато механізмів TCP. Зокрема три основні механізми: механізм оновлення вікна перевантаження, механізм повторної передачі та механізм тайм-ауту.

Ідея полягає в тому, що не потрібно повністю ігнорувати помилки, якщо вони були класифіковані як помилки передачі, натомість треба зменшити швидкість передачі TCP на основі кількості відкинутих пакетів на

вікно. Крім того, у разі багаторазових переривань передачі треба повторно передати всі пакети, випущені з того самого вікна. Крім того, потрібно проводити обчислення часу очікування, яке враховує поточні умови мережі.

Ці алгоритми мають бути реалізовані в TCP-відправнику, і необхідно, щоб вони використовувалися розрізнювачами помилок TCP у разі передачі пакету.

4.1 Робота перевантажувального вікна

Тривіальною дією, коли відбуваються переривання передачі, є повторне надсилання втраченого пакета та уникнення зменшення перевантажувального вікна. Цей підхід є основою більшості наскрізних рішень на основі відправника. Він базується на наступному міркуванні: загальна формула для обчислення пропускної здатності TCP за допомогою RTT та розміру вікна:

$$\text{Throughput}_i = \frac{W_i}{RTT_i} \tag{4.1}$$

де W_i – розмір вікна в час проходження туди й назад RTTi. Отже, середню пропускну здатність можна отримати таким чином:

$$\text{AvgThroughput} = \frac{\text{AvgW}}{\text{AvgRTT}} \tag{4.2}$$

Отже, якщо можна підтримувати середній розмір вікна якомога більшим під час переривань передачі, пропускна здатність має бути вищою, ніж звичайний TCP (де розмір вікна скорочується з кожним скиданням пакету), припускаючи, що отримано AvgRTT (тобто AvgRTT не збільшується зі збільшенням перевантажувального вікна).

Цей висновок було зроблено на основі припущення, що AvgRTT не залежить від AvgW. Іноді краще скоротити перевантажувальне вікно навіть для втрати пакетів під час передачі.

У високошвидкісних мережах TCP потрібен великий розмір вікна, щоб використовувати доступну пропускну здатність каналу. Однак, якщо підключення страждає від помилок передачі, канальний рівень буде зайнятий повторним надсиланням пошкоджених пакетів, і, отже, його пропускна здатність зменшиться.

У той же час TCP залишатиме перевантажувальне вікно, оскільки помилки є помилками передачі. Це призведе до збільшення затримки, оскільки канальний рівень буде змушений буферизувати пакети, доки вони не будуть повторно передані належним чином, або навіть відкидати їх, якщо розмір буфера недостатній або після тайм-ауту. Тому на практиці бажано контролювати збільшення перевантажувального вікна у разі помилок передачі з з деяких наступних причин.

Контроль збільшення перевантажувального вікна навіть для помилок передачі може запобігти небажано великій кількості відкидань пакетів під час тимчасових фаз перевантаження.

Якщо перевантаження відбувається під час великого перевантажувального вікна, велика кількість пакетів може бути відкинута, що змушує TCP вводити серію подій тайм-ауту. Ці події тайм-ауту змушують TCP очікувати в режимі очікування, а також експоненціально збільшуватимуться з кожним наступним тайм-аутом.

Неконтрольоване збільшення перевантажувального вікна може призвести до збільшення навантаження на мережу, що призведе до збільшення RTT.

Будь-яке збільшення RTT має в основному два побічні ефекти на пропускну здатність TCP, тоб то швидкість збільшення RTT може перевищувати швидкість збільшення розміру перевантажувального вікна, і це, як правило, скасовує будь-який приріст пропускної здатності.

Іншим ефектом збільшення RTT з’єднання є те, що воно збільшує таймер очікування повторної передачі TCP і, отже, збільшує період очікування TCP після помилок.

У багатьох мережах канальний рівень відповідає за буферизацію та повторну передачу втрачених пакетів, спричинених збоєм зв’язку. Однак, якщо відправник кінцевої точки продовжує надсилати дані на високій швидкості, незважаючи на втрати передачі даних, канальний рівень буде змушений буферизувати великі обсяги даних або навіть відкидати деякі пакети, що в кінцевому підсумку призведе до збільшення RTT і, як наслідок до перевантаження мережі.

Усі ці фактори в кінцевому підсумку призведуть до збільшення затримки кожного пакета та, отже, до збільшення середнього часу зворотного зв’язку для всього з’єднання (AvgRTT). З цього можна зробити висновок, що якщо класифікатор помилок не скорочує перевантажувальне вікно у випадку перепадів передачі, RTT може збільшитися таким чином, що може скасувати будь-яку вигоду, отриману від збільшення розміру перевантажувального вікна.

Основна ідея полягає в тому, щоб зменшити розмір перевантажувального вікна TCP cwnd на кількість скинутих пакетів в останньому вікні у разі помилок передачі, щоб запобігти зростанню навантаження на мережу, а отже, збільшенню AvgRTT з’єднання. І оскільки скорочується перевантажувальне вікно як для перевантаження, так і для помилок передачі, це допоможе запобігти зростанню перевантаження у випадку, коли класифікатор помилок не вірно діагностував скидання пакетів як помилки передачі.

Нарешті, будь-яке скидання пакетів в мережі, навіть помилки передачі в каналах, вказують на те, що канал не може впоратися з такою кількістю додаткових пакетів у даний момент часу. З цієї причини вкрай бажано зменшити поточне перевантажувальне вікно принаймні на певну кількість пакетів.

4.2 Алгоритм

Запропонований алгоритм дії перевантажувального вікна (АПВ), і він працює наступним чином, у разі відкидання пакетів TCP скорочує перевантажувальне вікно після отримання трьох дублікатів підтвердження.

Пропонується відкласти рішення про зменшення перевантажувального вікна, доки TCP не отримає всі повторювані підтвердження для поточного вікна (тобто пакети, що надходять під час втрати пакету). Подвійне підтвердження зазвичай вказує на скидання пакета, але також вказує на те, що один пакет покинув мережу (отриманий іншою стороною). Використовуючи цю інформацію, можна оцінити, скільки пакетів було відкинуто на перевантажувальне вікно:

$$droppedpackets=Windowsize-(No.ACKs+No.DACKs).$$

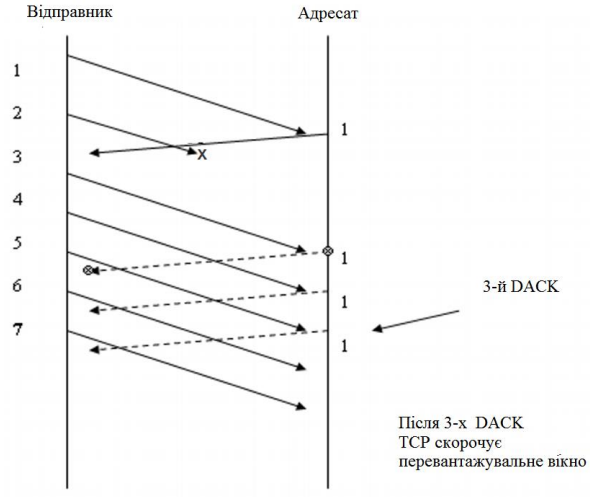

Рисунок 4.1 – Подвійне підтвердження TCP

Щоб переконатися, що отримано всі дублікати підтвердження, TCP має надіслати новий пакет після отримання кількох дублікатів підтвердження, і оскільки ця передача пакету відбулася після надсилання попереднього вікна, його підтвердження буде останнім, яке буде отримано, тому, коли TCP

отримає кількість дублікатів підтвердження, а потім підтвердження для пакета закриття, який він знає, що отримав усі дублікати підтвердження для поточного вікна.

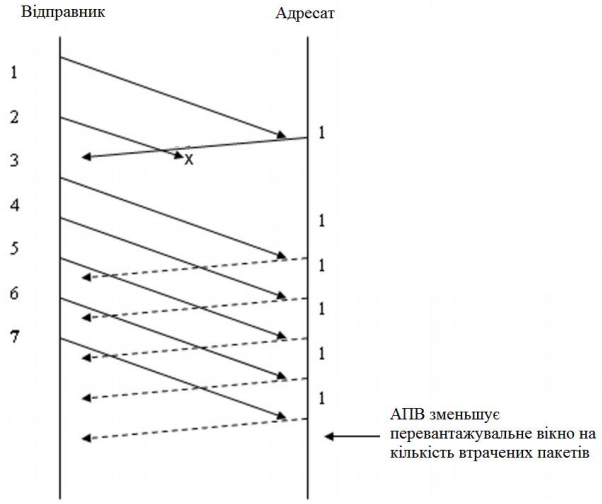

Рисунок 4.2 – Дія подвійного підтвердження АПВ

Нехай називається цей пакет закриваючим пакетом, оскільки він закриває попереднє вікно. Крім того, можна використати повторну передачу першого втраченого пакета як завершальний пакет, і таким чином TCP може прискорити процес відновлення. Крім того, для простоти припустимо, що підтвердження закриття буде проходити тим самим шляхом, що й попередні підтвердження.

Отже, у випадку помилок передачі, замість того, щоб скорочувати перевантажувальне вікно наполовину (як TCP) або не скорочувати його взагалі, пропонується скорочувати його лише за кількістю відкинутих пакетів. Таким чином TCP скорочує перевантажувальне вікно відповідно до кількості скинутих пакетів.

Перевага цього методу полягає в тому, що він покращує продуктивність (особливо для малих частот помилок) і уникає підвищення рівня перевантаженості, змушуючи TCP скорочувати швидкість надсилання навіть для помилки передачі.

Алгоритм представлено в лістингу 4.1. В алгоритмі TCP перевіряє, чи є поточне підтвердження дублікатом підтвердження, і якщо так, він перевіряє, чи це третій дублікат підтвердження в рядку. Потім він зберігає порядковий номер останнього надісланого пакета в останньому надісланому 3Dack і повторно надсилає втрачений пакет.

Потім, коли одержувач підтверджує повторно переданий пакет, TCP перевіряє, чи він підтвердив усі пакети до останнього надісланого 3Dack і якщо це не так (тобто останній надісланий 3Dack > current_ACK), тоді TCP обчислює кількість скинутих пакетів і відповідно скорочує перевантажувальне вікно.

Однак якщо отримане підтвердження стосується всіх надісланих пакетів (тобто останній надісланий 3Dack == current_ACK), тоді TCP нічого не робить, оскільки було лише одне скидання пакету, який було повторно передано та безпечно отримано. Крім того, якщо повторна передача не вдалася і мається подія тайм-ауту, TCP скорочує перевантажувальне вікно до одиниці.

АПВ слід використовувати лише у випадку помилок передачі. Однак, якщо детектор помилок неправильно використовував АПВ також для помилок перевантаження, ситуація в мережі може погіршитися. Щоб вирішити цю проблему, визначимо інше порогове значення, так званий порог перепадів передачі (tthresh).

Він використовується для запису розміру перевантажувального вікна (cwnd), коли відбудеться перше скидання пакету. Він визначає область між початком фази уникнення перевантаження (тобто від ssthresh) і першим скиданням. Оскільки це перше скидання, позначимо cwnd size up для tthresh безпечної зони.

Інформація, яку надає tthresh, полягає в тому, що перед цією точкою немає скидань пакетів, тому, ймовірно, немає заторів перед цією точкою, і що після цієї точки відбулися скидання, отже, існує ймовірність виникнення перевантажень.

Лістинг 4.1 – Псевдокод АПВ

```
Ініціалізація: prev_ack=-1; last_sent_3Dack=-1
З кожним отриманим підтвердженням Ack_i:
current_ack=Ack_i
if (current_ack==prev_ack) then              // подвійний ack
  dackcount=dackcount+1
  if dackcount==3 then                       // скидання пакету
    last_sent_3Dack=P_max
    Передати пакет з seqNo=current_ack+1     // зберегти cwnd
  end if
end if
if (current_ack>prev_ack) then               // не більше DACKs
  prev_ack=current_ack
  if (last_sent_3Dack>current_ack) then
    flight_size=last_sent_3Dack-current ack  //розмір
    num_drops=flight_size-dackcount
    cwnd=cwnd-num_drops                      //розмір вікна
  end if
end if
if timeout==true then                        //тайм-аут
  ssthresh=max(2,cwnd/2)
  cwnd=1
end if
```

Змінні:

- ccurrent_ack: порядковий номер поточного підтвердження;
- prev_ack: порядковий номер попереднього підтвердження;
- dackcount: змінна для відстеження кількості дублікатів підтвердження TCP, отриманих на даний момент (ця змінна встановлюється на 0 кожного разу, коли надходить нове підтвердження);
- last_sent_3Dack: змінна для збереження порядкового номера останнього надісланого пакету після отримання 3 DACK;

- P_{max}: останній надісланий пакет;
- flight_size: кількість пакетів, надісланих, але ще не підтверджених;
- num_drops: кількість пакетів, викинутих із цього вікна;
- cwnd: розмір перевантажувального вікна;
- ssthresh: поріг повільного запуску;
- RTO: Таймер тайм-ауту повторної передачі.

Тепер, якщо класифікатор помилок вказує, що визначена помилка є помилкою передачі, тоді перед тим, як класифікатор помилок вирішить, як скоротити перевантажувальне вікно, АПВ виконує ще одну перевірку (тобто це дворівнева перевірка, одна класифікатором помилок, а друга АПВ) за допомогою порівняння перевантажувального вікна, коли відбувається скидання, з tthresh.

Якщо перевантажувальне вікно більше, ніж tthresh, існує велика ймовірність того, що класифікатор помилок не відповідає помилці передачі, тому класифікатор помилок реагує консервативним способом, розглядаючи скидання пакету як скидання перевантаження та скорочує перевантажувальне вікно наполовину. (як це робить звичайний TCP).

Мета – максимально підвищити точність діагностики помилок перевантаження, щоб уникнути шкоди мережі. Однак, якщо помилка виникає, коли перевантажувальне вікно менше tthresh, можна безпечно вважати помилку помилкою передачі.

Алгоритм АПВ з tthresh представлений в листингу 4.2. Після першого скидання пакету значення cwnd зберігається в tthresh. Пізніше, коли відбудеться ще одне скидання, класифікатор помилок перевірить, чи cwnd ≤ tthresh, і якщо так, скидання пакету, ймовірно, є помилкою передачі.

В іншому випадку скидання пакету вважається помилкою перевантаження. Крім того, TCP має перераховувати tthresh після кожної події тайм-ауту, оскільки TCP ініціалізує перевантажувальне вікно та почне створювати нове вікно. Це робиться в алгоритмі за допомогою першого відкидання, яке буде встановлено в одиницю після кожного тайм-ауту.

Лістинг 4.2 – АПВ+tthresh

```
Ініціалізація: prev_ack=-1; last_sent_3Dack=-1; first_drop=1
З кожним отриманим підтвердженням Ack_i:
current_ack=Ack_i
if (current_ack==prev_ack) then              //подвійний ack
  dackcount=dackcount+1
  if dackcount==3 then                       //скидання пакету
    last_sent_3Dack=Pmax
    Передати пакет з seqNo = current ack+1   // зберегти cwnd
    if first_drop then
      tthresh=cwnd
      first_drop=0                           //встановлюєм в 0
    end if
  end if
end if
if (current_ack>prev_ack) then               //більше
  prev_ack=current_ack
  if (last_sent_3Dack>current_ack) then //більше поточного
    flight_size=last_sent_3Dack-current_ack
    num_drops=flight_size-dackcount
    if cwnd<tthresh then                     //менше порога
      cwnd=cwnd-num_drops
      else
      cwnd=cwnd/2
      ssthresh=cwnd                          //поріг
    end if
  end if
end if
if timeout==true then                        //тайм-аут
  ssthresh=max(2,cwnd/2)
  cwnd= 1
  first_drop=1                               //ініціалізація
first_drop після кожного timeout
end if
```

first drop – це прапорець, який вказує, що відбувся перший розрив цього з’єднання.

Тайм-аут додається до наведених вище алгоритмів лише для того, щоб показати, що відбувається у випадку тайм-ауту, однак зазвичай тайм-аут буде в окремій функції та викликатиметься лише після закінчення таймера. Також слід зазначити, що у разі помилок передачі не змінюється ssthresh, а лише змінюється cwnd, оскільки скидання пакетів не є помилкою перевантаження, і, ймовірно, пропускна здатність каналу (позначена ssthreh) суттево не змінилася.

У решті цієї роботи буде використовуватися остаточна версія алгоритму (АПВ+tthresh) і називатиметься АПВ для простоти.

Нарешті, однією з важливих цілей АПВ є збільшення розміру перевантажувального вікна TCP шляхом зменшення швидкості скорочення перевантажувального вікна у разі помилок передачі. Однак TCP відновлює лише перший скинутий пакет, а решту залишає для відновлення через тайм-аути.

У цьому випадку алгоритм перевантажувального вікна АПВ зазнає негативного впливу, оскільки TCP скидає розмір перевантажувального вікна до одного сегмента після кожного тайм-ауту, тому будь-яке скорочення АПВ буде скасовано.

З цієї причини буде запропоновано алгоритм для відновлення кількох відкинутих пакетів на вікно даних, який має на меті зменшити вплив подій тайм-ауту на АПВ.

4.3 Продуктивність АПВ

Важливою метою АПВ є покращення продуктивності TCP шляхом збереження більшого перевантажувального вікна у разі помилок передачі. Щоб виміряти покращення, необхідно вимірювати середній розмір перевантажувального вікна протягом усього терміну з’єднання.

Припустимо, що в з’єднанні присутні лише помилки передачі, тому використовується проста топологія, представлена на рисунку 4.3. Подібні топології використовуються для тестування модифікацій TCP лише за наявності помилок передачі.

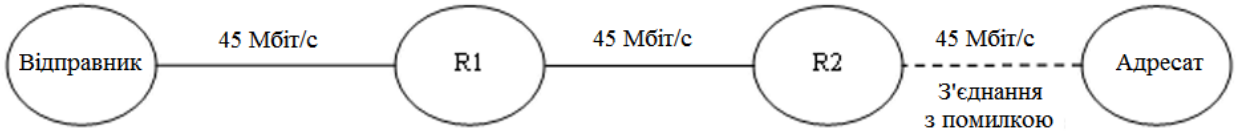

Рисунок 4.3 – Проста топологія мережі

Кожна смуга пропускання bw фіксована на рівні 45 Мбіт/с (з’єднання Т3). Загальна затримка поширення каналу становить 12 мс, тому кожна затримка каналу займає 2 мс. Помилки передачі генеруються в останньому каналі зв’язку з використанням моделі двох станів для імітації фаз помилок і фаз без помилок, а частота (коефіцієнт) помилок коливається від 0,001 (0,1%) до 0,1 (10%) зі збільшенням на 1% кожного разу. Той самий діапазон помилок використовується для всіх експериментів, за винятком алгоритму тайм-ауту повторної передачі (АТПП), де було збільшено рівень помилок до 20%, щоб показати покращення за серйозних помилок. У всіх експериментах повторюється експеримент достатню кількість разів з різним початковим значенням для генератора випадкових чисел. 95% довірчі інтервали дуже малі і не видно на графіках, тому середнє значення тільки на графіках.

Запускається експеримент, щоб виміряти перевантажувальне вікноTCP, а потім додається АПВ до TCP (щоб замінити стандартну реакцію TCP на помилки, яка полягає в скороченні перевантажувального вікна наполовину) і повторюється експеримент із зміненим TCP.

Графіки на рисунку 4.4 показують, що після додавання АПВ TCP отримав більше середнє перевантажувальне вікно. Це досягається за допомогою АПВ, що запобігає непотрібним скороченням

перевантажувального вікна та обмежує скорочення кількістю втрачених пакетів у разі переривання передачі. На рисунку використовується логарифмічно-лінійний масштаб, оскільки значення розміру перевантажувального вікна займають великий діапазон, тому використовується логарифмічно-лінійний (або напівлогарифмічний) масштаб, щоб було легко побачити низькі та високі значення на осі у.

Збільшення розміру перевантажувального вікна збільшує швидкість надсилання TCP. Крім того, це зменшує ймовірність тайм-аутів, оскільки з більшим вікном TCP отримує більше повторюваних підтверджень після відкидання.

Ці дублікати підтвердження викликають повторну передачу втрачених пакетів і збільшують перевантажувальне вікно під час фази швидкого відновлення.

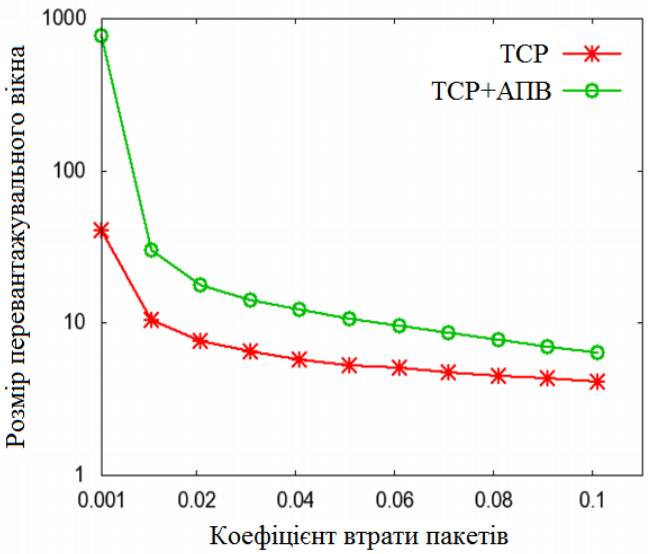

Рисунок 4.4 – Порівняння TCP та АПВ

Розмір перевантажувального вікна представлено в напівлогарифмічному масштабі (пакети). Однак через те, що збільшення частоти помилок збільшить тривалість тайм-ауту, перевантажувальне вікно не матиме шансу збільшитися після подіЇ тайм-ауту.

Більше того, зі збільшенням частоти помилок багато пакетів буде відкинуто більше одного разу, і через багаторазове відкидання пакетів виникатимуть більш тривалі тайм-аути повторної передачі. Це ще одна проблема, яка знижує продуктивність АПВ, тобто багаторазове скидання пакетів. Оскільки TCP повторно надсилає лише один скинутий пакет на вікно, решта буде відновлена через тайм-аут. Це збільшить кількість подій тайм-ауту, що негативно вплине на продуктивність АПВ. У наступному розділі буде розв’язана ця проблема за допомогою алгоритму повторної передачі (АПП).

Обмеження: tthresh пропонується як друга лінія захисту від створення непотрібних перевантажень, спричинених класифікаторами помилок, які можуть діагностувати помилки перевантаження як помилки передачі й, отже, не реагувати на перевантаження. tthresh намагається запобігти цьому, відстежуючи скидання пакетів та записуючи розмір перевантажувального вікна (cwnd), коли відбувається перше скидання, і розглядає це як ознаку того, що саме тут відбуваються перевантаження.

Отже, якщо пізніше відбудеться ще одне скидання, коли перевантажувальне вікно перевищує цей розмір, то це, ймовірно, скидання перевантаження.

Однак цей метод може зменшити приріст продуктивності TCP від використання класифікатора помилок, якщо помилки передачі сталися під час cwnd>tthresh. Це пояснюється тим, що якщо cwnd>tthresh, АПВ використовує стандартну дію TCP, скорочуючи перевантажувальне вікно до половини.

Однак використання tthresh захищає мережу від непотрібних перевантажень, спричинених помилками класифікаторів. Більше того, у найгіршому випадку, якщо всі помилки, у яких є помилками передачі та одночасно відбуваються, коли cwnd>tthresh і, отже, обробляються АПВ як помилки перевантаження, продуктивність буде найгіршою як у стандартного TCP (тобто скорочується перевантажувальне вікно наполовину для кожної

втрати пакету). Крім того, tthresh перераховується після кожної події тайм-ауту, оскільки тайм-аут вказує на те, що значення tthresh, ймовірно, не підходить для запобігання перевантаженням, оскільки тайм-аут зазвичай вказує на серйозне перевантаження.

Крім того, оскільки АПВ скорочує лише cwnd, а не ssthresh, це допоможе швидко відновити cwnd, оскільки cwnd експоненціально збільшується, коли воно менше за ssthresh («повільний старт»). Це швидше відновлення допоможе збалансувати зниження продуктивності, спричинене tthresh.

4.4 Відновлення після кількох втрат пакетів

TCP надсилає лише перший пакет, скинутий із вікна даних, а решту залишає для відновлення за допомогою механізму тайм-ауту повторної передачі після періоду очікування.

Однак численні скидання перевантаження можуть виникнути через різку поведінку TCP (наприклад, експоненціальне зростання під час повільного запуску) або через серйозне перевантаження.

Якщо з одного вікна відкидається більше ніж один пакет, TCP повторно надсилає перший відкинутий пакет, а потім припиняє надсилання пакетів і залишається в режимі очікування, доки не настане тайм-аут. Цей тайм-аут ініціює повторне надсилання решти скинутих пакетів.

Однак це викликає втрату часу TCP завдяки двом факторам: по-перше, коли він залишається бездіяльним без надсилання даних, доки не настане тайм-аут, а по-друге, після будь-якого тайм-ауту повторної передачі TCP зменшує перевантажувальне вікно до мінімуму (зазвичай один сегмент).

Щоб вирішити цю проблему для зниження перевантаженості рекомендується змінити механізм швидкої повторної передачі в TCP, щоб змусити TCP повторно надсилати всі пакети, повторно надсилаючи один пакет кожного RTT. Цей метод дозволяє TCP залишатися в режимі повторної

передачі, доки всі втрачені пакети не будуть повторно передані. Однак проблема полягає в тому, що TCP повторно надсилає лише один пакет на RTT, оскільки він повторно надсилає пакет для кожного часткового підтвердження, яке він отримує (часткове підтвердження – це нове підтвердження, яке не підтверджує всі неопрацьовані пакети). Отже, якщо N пакетів скинуто з вікна, TCP знадобиться n*RTT для відновлення цього вікна.

4.4.1 Алгоритм

Використовуючи концепції, як у АПВ, у цьому алгоритмі TCP використовуватиме набір дублікатів підтвердження, отриманий після першого скидання пакету, щоб оцінити кількість скинутих пакетів на вікно, і, припускаючи, що це число представляє послідовні скинуті пакети, повторно надсилає цю кількість пакетів, починаючи з першого скинутого пакета. TCP обчислить кількість відкинутих пакетів шляхом віднімання кількості дублікатів підтвердження від кількості фактичних пакетів, надісланих із цього вікна. Тому після першого часткового підтвердження надсилається кількість пакетів, яку було розраховано. Таким чином, якщо скинуті пакети були послідовними, можна було б відновити все вікно за один RTT без непотрібних повторних передач.

Однак, якщо в цьому вікні є більше скинутих пакетів або скинуті пакети були розкидані, пропонується інший механізм повторної передачі, який може надсилати їх один за одним як NewReno або повторно надсилати решту, як Bulk Repeat на основі на спрощеної оцінки частоти помилок мережі. На рисунку 4.5 показано реакцію TCP на сплеск падінь пакетів, а на рисунку 4.6 показана запропонована ідея.

Як можна побачити з рисунків, запропонована ідея відновить вікно за менший час, ніж TCP і NewReno, і з меншою кількістю непотрібних повторних передач, за умови, що на кожне вікно виникає одна серія помилок.

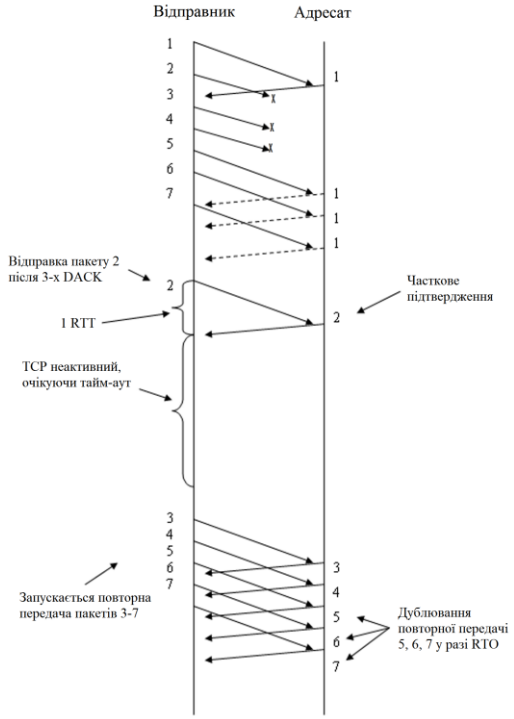

Рисунок 4.5 – TCP і сплеск відкидання пакетів

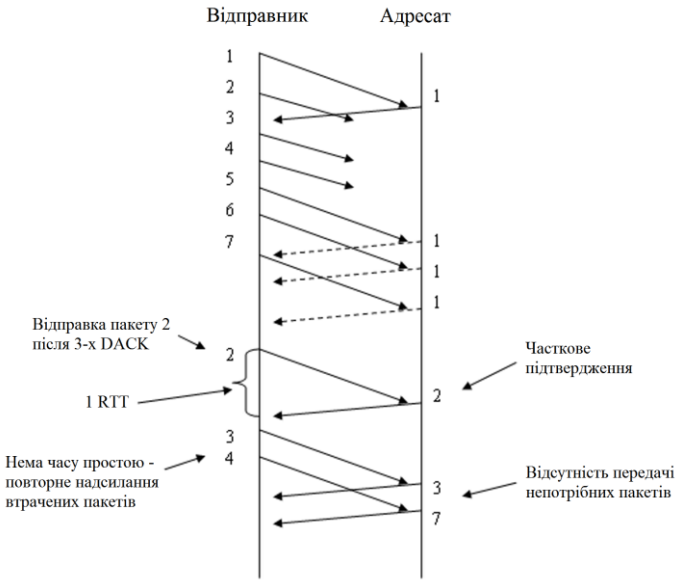

Рисунок 4.6 – Запропонована повторна передача з кількома втратами пакетів

Псевдокод алгоритму представлено в лістингу 3.3. Цей запропонований алгоритм має використовуватися класифікатором помилок для відновлення після кількох втрат пакетів на вікно.

Лістинг 4.3 – Псевдокод алгоритму АПП

```
Ініціалізація: prev_ack=-1; last_sent_3Dack=-1; first_burst=1
З кожним отриманим підтвердженням Ack_i:
current_ack=Ack_i
if (current_ack==prev_ack) then              //подвійний ack
  dackcount=dackcount+1
  if (dackcount==3&&first_burst) then        //скидання пакету
    last_sent_3Dack=Pmax
    передати пакет з seqNo=current_ack+1     //зберегти cwnd
  end if
end if
if (current_ack>prev_ack) then               //не більше DACK
  prev_ack=current_ack
  if ((last_sent_3Dack>current_ack) && first_burst) then
    розрахунок кількості скидань:
    flight_size=last_sent_3Dack-current_ack  //встановлюємо
    num_drops=flight_size-dackcount
    for (i=1;i≤(num_drops);i++) do           //перебір
      resend prev_ack+i
    end for
    first_burst=0                            //одна передача на
вікно
  else
    if (last_sent_3Dack≤current_ack) then
      first_burst=1                          //встановлюємо
    end if
  end if
end if
if timeout==true then                    //тайм-аут
  ssthresh=max(2,cwnd/2)
  cwnd=1
  last_sent_3Dack=-1                     //встановлюємо
   first_burst=1
end if
```

first_burst – прапорець, що вказує, чи це перша серія втрат пакетів у цьому вікні

Основна перевага полягає в тому, що він підвищує продуктивність TCP за рахунок збільшення швидкості повторного надсилання втрачених пакетів. Це дозволяє TCP повторно надсилати всі пакети, втрачені з того самого вікна, і це зменшує час очікування, особливо коли серія пакетів скидається. Зменшення кількості тайм-аутів є важливим для покращення продуктивності TCP, оскільки кожен тайм-аут скидає перевантажувальне вікно до мінімуму та змушує TCP бути в режимі очікування.

Однак у деяких випадках помилки передачі є постійними, як у випадку довгого розриву зв’язку, і навіть повторно передані пакети відкидаються, тому немає сенсу продовжувати повторну передачу втраченого пакета, тому потрібно зачекати (припинити надсилання) протягом часу, щоб знову запустити з’єднання. З цієї причини алгоритм повторно передає втрачені пакети лише один раз на вікно, а потім допускає тайм-аут повторної передачі між вікнами, якщо помилки є постійними.

Змінна first_dup в алгоритмі використовується, щоб дозволити повторне надсилання пакетів лише один раз на вікно. Незважаючи на те, що алгоритм спрямований на помилки передачі, він також може дати хороші результати з відкиданням пакетів при перевантаженнях.

4.4.2 Продуктивність АПП

Якщо відкидання не є послідовним, АПП повторно надсилає один пакет на RTT (як у NewReno). Це консервативний підхід, але він гарантує відсутність непотрібної повторної передачі. Щоб виміряти покращення за допомогою АПП, необхідно вимірювати кількість тайм-аутів протягом життя з’єднання для TCP до та після додавання АПП, оскільки дія АПП спрямована на зменшення кількості тайм-аутів, необхідних TCP, намагаючись повторно надіслати всі скинуті пакети.

На рисунку 4.7 показано кількість тайм-аутів для TCP і АПП. Як можна побачіти, кількість тайм-аутів у випадку АПП набагато менше, ніж у TCP. Однак навіть у разі використання АПП інколи очікується виникнення тайм-аутів, особливо з вищим рівнем помилок, коли той самий пакет може бути відкинутий більше одного разу, і відповідно, АПП повторно надсилає пакет лише один раз на вікно, щоб запобігти непотрібній повторній передачі у випадку, якщо з'єднання буде поганим на довгий час.

Коли використовується логарифмічна шкала числа RTO, можна побачити, що дві криві мають однакову форму на рисунку 4.8. Це вказує на те, що і TCP, і АПП страждають від ефекту мультиплікативного скорочення перевантажувального вікна, яке використовується в TCP (тобто скорочення перевантажувального вікна наполовину після кожної втрати пакету), що призведе до меншого розміру перевантажувального вікна, а отже, і до меншої кількості підтверджень, що призводить до більшої кількості тайм-аутів. Ця проблема вирішується за допомогою дії АПВ. Це показує, що АПВ та АПП доповнюють одне одного.

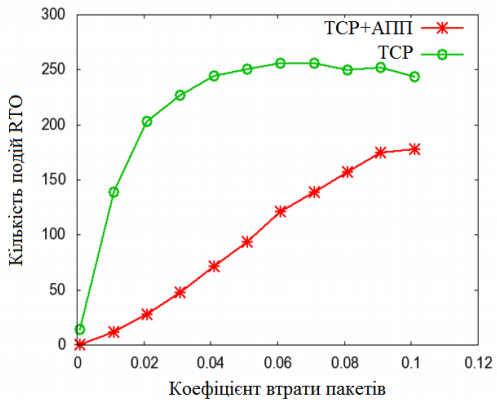

Рисунок 4.7 –Порівняння TCP та АПП

Слід зазначити, що кількість тайм-аутів на рисунку 4.8 для АПП починається зі значення менше одиниці (0,3). Це пояснюється тим, що запускається кожен експеримент кілька разів, а потім береться середнє

значення всіх циклів, тому в цьому нижчому рівні помилок АПП настільки ефективний у деяких циклах, що кількість RTO дорівнює 0, а в деяких – 1, тому в середньому 0,3.

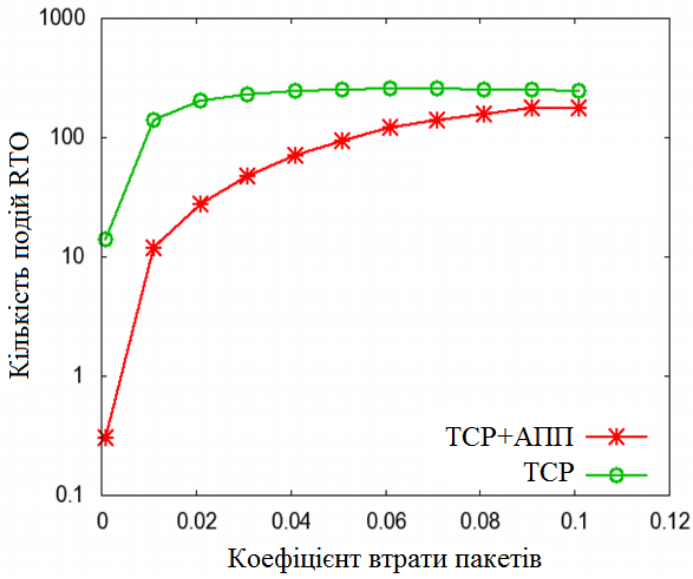

Рисунок 4.8 – Напівлогарифмічна шкала TCP та АПП

Крім того, ми можна помітити, що на рисунку 4.7 зі збільшенням частоти помилок кількість тайм-аутів починає встановлюватися приблизно на 250 (тобто без великого збільшення) у випадку TCP.

Це природно, оскільки зі збільшенням частоти помилок з'єднання швидко деградує через велику кількість скинутих пакетів, тому кількість надісланих пакетів зменшується, а отже, кількість тайм-аутів не надто збільшується.

Нарешті, той факт, що навіть коли TCP використовує АПП, кількість тайм-аутів збільшується, показує, що АПП не в змозі повністю приховати ефект відкидання кількох пакетів від TCP, що призводить до того, що TCP потрапляє в тайм-аут.

Однак АПП, безсумнівно, здатний зменшити цей вплив на TCP, особливо за нижчої частоти помилок, що призводить до зменшення кількості тайм-аутів.

4.5 Покращення механізму RTO

Коли в відбувається серія тайм-аутів, TCP експоненціально збільшує час очікування повторної передачі (RTO) (тобто збільшує час очікування після відкидання). Це збільшення називається відстрочкою тайм-ауту повторної передачі та є оцінкою часу, необхідного мережі для очищення своїх буферів після перевантажень.

Причина вибору експоненціального збільшення полягає в тому, щоб зробити відправника TCP більш обережним, поки перевантаження все ще існує. Однак, якщо немає перевантажень і падіння є результатом помилки передачі, це експоненціальне очікування може призвести до невиправдано тривалих періодів бездіяльності.

У випадку помилок передачі класифікатор помилок TCP повинен збільшувати час очікування з кожним тайм-аутом, але таким чином, щоб враховував стан мережі шляхом оцінки доступної пропускної здатності мережі та обчислення зворотного зв’язку. на основі цієї оцінки. Отриманий рівень відставання коливатиметься між експоненціальним відступом і відступом із фіксованим значенням на основі оціненої пропускної здатності.

Після кожної події тайм-ауту TCP збільшує RTO (backs-off) наступним чином:

$$RTO=RTO*2. \tag{4.3}$$

Отже, після n подій тайм-ауту можна розрахувати RTO наступним чином:

$$RTO=RTO*2^{n}, \tag{4.4}$$

де n –означає кількість невдалих спроб повторної передачі на даний момент (тобто кількість нових отриманих підтвержень немає).

Замість подвоєння RTO з кожною невдалою повторною передачею спочатку оцінюємо доступну пропускну здатність мережі, а потім замість використання n у рівнянні 4.4 як ступінь 2 будемо використовувати функцію f(n), яка відображає доступну пропускну здатність безпосередньо перед першим тайм-аутом. Отже, після n подій тайм-ауту нова політика очікування буде обчислена наступним чином:

$$RTO = RTO * 2^{f(n)}. \quad (4.5)$$

Щоб обчислити f(n), треба оцінити доступну смугу пропускання. Доступна пропускну здатність bw оцінюється шляхом обчислення швидкості отриманих підтвердження, де кожне підтвердження представляє один розмір сегмента, який був успішно доставлений.

$$bw = \frac{\text{розмір сегменту}}{T_i - T_{(i-1)}}, \quad (4.6)$$

де T_i – час отримання Ack_i, а $T_{(i-1)}$ – час отримання $Ack_{(i-1)}$.

Використовуючи це рівняння, можна оцінити доступну пропускну здатність bw як швидкість пакетів, які можуть успішно пройти через вузьке місце мережі протягом часу, що дорівнює $T_i - T_{(i-1)}$. Збільшення цієї швидкості означає збільшення доступної смуги пропускання і навпаки. Потім необхідно обчисллити середнє зважене значення avalb_bw доступних показників пропускної здатності bw, щоб відфільтрувати раптові зміни:

$$avalb\_bw = \beta \cdot avalb\_bw + (1-\beta) \cdot bw. \quad (4.7)$$

Оскільки оцінка пропускної здатності дуже пов’язана з оцінкою часу проходження мережі (RTT), використовуються значення між 0,8 і 0,9 для β, які є тими значеннями, які використовуються в TCP для обчислення RTT.

Тепер, використовуючи дані оцінювача пропускної здатності обчислюємо f(n) наступним чином:

$$f(n)=n\cdot\left(1-\frac{avalb\_bw}{max\_avalb\_bw}\right), \quad (4.8)$$

де max_avalb_bw – це максимальне значення доступної смуги пропускання, яке спостерігалося на даний момент.

Алгоритм розрахунку доступної смуги пропускання представлено на лістингу 4.4. Цей алгоритм слід викликати лише після того, як відправник TCP отримає принаймні два підтвердження, щоб мати можливість обчислити $T_{(i-1)}=T_i$.

Крім того, після обчислення першого значення доступної пропускної здатності bw необхідно обов’язково обчисллити середнє зважене значення avalb_bw=bw.

Лістинг 4.4 – Псевдокод алгоритму розрахунку доступної смуги пропускання

```
Ініціалізація: β=0.9; current_time=0; max_avalb_bw=0
з кожним Acki:
prev_time=current time              //T(i-1)=Ti встановити
current_time=get_current_time
bw=segment_size/(current_time-prev_time)     //смуга
avalb_bw=β*avalb_bw+(1-β)*bw
if avalb_bw>max_avalb_bw then                //виконується
  max_avalb_bw=avalb_bw
end if
```

Алгоритм розрахунку RTO (RTO back-of algorithm) представлено в лістингу 4.5.

Лістинг 4.5 – Псевдокод алгоритму розрахунку RTO

```
Ініціалізація: Fn=0; ORTO=CRTO
після кожного timeout:
n=n+1
Fn=n*(1-(avalb_bw/max_avalb_bw))              //функція
CRTO=ORTO*2Fn
```

Змінні:

- n – кількість подій тайм-ауту на даний момент;
- ORTO –тайм-аут, розрахований на основі показань;
- CRTO – поточний timeout.

Ідея полягає в тому, що у випадку помилок передачі відношення avalb_bw/max_avalb_bw буде використовуватися для оцінки рівня відстрочки тайм-ауту повторної передачі класифікатора помилок. Отже, якщо немає заторів, тоді avalb_bw буде дуже близьким до максимального avalb_bw. Це призведе до меншого F_n і, отже, до меншого значення часу очікування повторної передачі (CRTO).

Тепер, коли класифікатор помилок вирішив, що помилка є помилкою передачі, тоді замість використання тривалого тайм-ауту повторної передачі, як у TCP, він використовує тайм-аут, розрахований на основі доступної пропускної здатності, що робить його більш пов’язаним із поточними умовами мережі.

Це допоможе TCP, якщо немає перевантажень і є лише помилка передачі, оскільки це зменшить періоди очікування під час подій тайм-ауту та, отже, збільшить швидкість повторної передачі, що дасть пакетам більше шансів бути доставленими.

Крім того, скорочення часу очікування у випадку помилок, не пов’язаних із перевантаженням, дозволить TCP надсилати той самий обсяг даних за коротші періоди часу.

Крім того, оскільки перевантажувальне вікно після тайм-ауту зменшується до одного сегмента, збільшення швидкості повторної передачі не збільшить ризик перевантаження в мережі, оскільки вікно відправника обмежене лише одним пакетом.

Це зробить використання класифікатора помилок можливим навіть із низькою точністю розрізнення, оскільки TCP запускатиметься з невеликим перевантажувальним вікном після кожного RTO, тому навіть якщо класифікатор помилок не відрізнить помилку перевантаження від помилки передачі, вплив на мережу буде мінімальним.

Крім того, у разі перевантаження оцінена доступна пропускна здатність зменшиться, і це збільшить значення F_n в алгоритмі. Отже, навіть якщо класифікатор помилок не відрізнить помилку перевантаження від помилки передачі, відмова буде близькою до стандартного TCP (тобто експоненціальне збільшення RTO).

Нарешті, використання оцінки пропускної здатності для обчислення значення відставання базується на роботі, представленій у [13]. Однак важливою відмінністю між запропонованим методом і методом [13] є те, що він використовує адитивне збільшення RTO замість мультиплікативного збільшення.

Однак, хоча використання адитивного збільшення зменшує довжину RTO під час великих втрат пакетів, коли перевантаження та помилки передачі співіснують, рідкісне збільшення довжини RTO призведе до того, що класифікатор помилок спричинить перевантаження у випадку невідповідності помилок, оскільки збільшення RTO буде невеликим і мережа не встигне усунути перевантаження.

Однак у запропонованому алгоритмі тайм-ауту повторної передачі (АТПП), використання мультиплікативного збільшення довжини RTO запобіжить цьому, а також внесе мінімальні зміни в реалізацію TCP, оскільки TCP використовує мультиплікативне збільшення для обчислення рівня тайм-ауту. Інша відмінність полягає в тому, що в реалізації динамічно

обчислюється максимальна доступна пропускна здатність, тоді як у [13] максимальна пропускна здатність обмежена 128 Кбіт/с. Крім того, пропонується практичний метод інтеграції нового механізму в TCP, реалізувавши його в класифікаторі помилок, що дозволить використовувати його з низьким і високим рівнем помилок, а також за наявності помилок перевантаження.

На рисунку 4.9 представлено порівняння загального часу простою TCP через RTO до та після додавання АТПП. Використовуються ті самі параметри експерименту, що й для оцінки продуктивності АПВ, за винятком того, що збільшено діапазон частоти помилок до 20%, оскільки з вищою частотою помилок TCP матиме довші події тайм-ауту, оскільки пакети відкидаються більше одного разу.

Коли використовується АТПП, спостерігається помітне зменшення загальної тривалості подій RTO, особливо зі збільшенням частоти помилок (поліпшення досягає 10% порівняно з частотою помилок 20%). Це пов’язано з тим фактом, що політика відстрочки є найважливішою, коли відбувається багаторазове відкидання одного пакета, що призводить до кількох подій невдалої повторної передачі. Кожна невдала повторна передача експоненціально збільшує час очікування TCP.

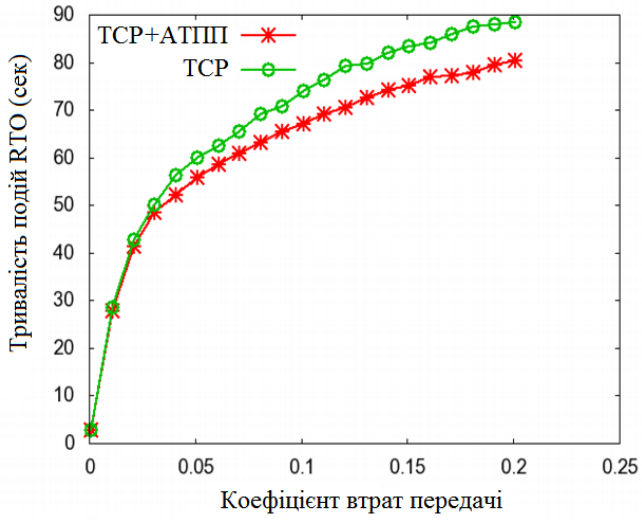

Рисунок 3.9 – Довжина RTO

Оскільки кількість подій скидання пакетів збільшується зі збільшенням частоти помилок, можна побачити, як АТПП зменшує вплив множинних втрат пакетів на TCP, оскільки час очікування більше не збільшується експоненціально з кожною втратою пакету, а натомість більше залежить від фактичного стану мережі та передбачуваної пропускної здатності.

На рисунку 4.10 показано затримки TCP і АТПП порівняно з розрахунковою смугою пропускання за високої швидкості переривання передачі (20%). TCP реагує на високу швидкість скидання пакетів, збільшуючи рівень затримки до найвищого значення 64 незалежно від умов мережі. Однак у випадку АТПП затримка приймає значення, пов'язане з доступною пропускною здатністю (пропускна здатність вимірюється в Мбіт/с), яка змінюється відповідно до змін у доступній пропускній здатності мережі (тобто вона зменшується зі збільшенням доступної пропускної здатності та навпаки).

Однак за низьких частот помилок немає ніяких покращень, як можна побачити на рисунку 4.10, оскільки за таких низьких частот помилок є менше подій тайм-ауту, і, отже, TCP не збільшує RTO. Однак алгоритм все ще може зменшити довжину RTO і, отже, час простою TCP зі збільшенням частоти помилок.

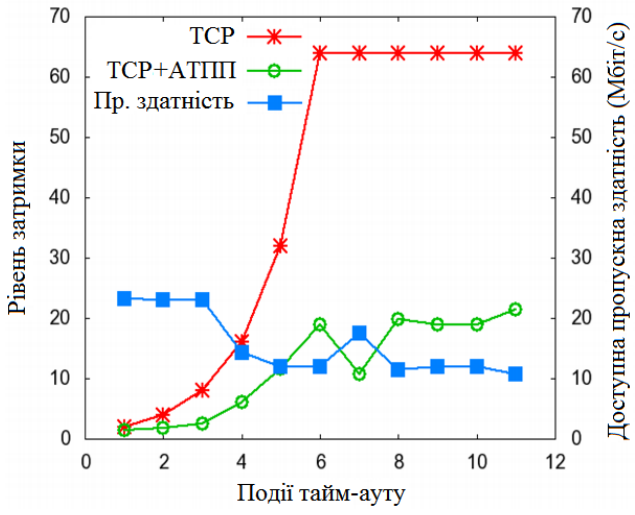

Рисунок 4.10 – Доступна пропускна здатність і рівень затримок

Якщо підтвердження пропущено, оцінка пропускної здатності, ймовірно, не працюватиме. Однак ця проблема виникла спочатку через дизайн TCP. TCP використовує підтвердження для обчислення RTT і RTO. Якщо підтвердження відкинуто, TCP не обчислить RTO належним чином, і оскільки запропонований алгоритм додано до TCP, він матиме ту саму проблему.

Однак припущення про те, що підтвердження будуть доставлені в більшості випадків, підтверджується тим фактом, що підтвердження зазвичай є дуже маленькими пакетами порівняно зі звичайними TCP-пакетами (зазвичай 4% від звичайного розміру сегмента TCP), тому є менша ймовірність їх втрати завдяки помилкам передачі або перевантаженням. Крім того, у разі втрати підтвердження, оцінка пропускної здатності АТПП зменшиться, тому продуктивність буде подібною до TCP, і це гарантуватиме відсутність шкоди для мережі, оскільки TCP експоненціально знижує швидкість передачі.

4.6 Функції класифікатора помилок

Раніше було запропонувано три алгоритми АПВ, АПП і АТПП. Будемо називати ці алгоритми в комбінації алгоритмами класифікатора помилок (АКП).

Хоча функціональні можливості кожного алгоритму є незалежними, ці алгоритми були розроблені для формування дії передачі для класифікатора помилок, щоб допомогти йому реагувати на помилки передачі. На рисунку 4.11 показано загальний дизайн класифікатора помилок, а на рисунку 4.12 показано зміни після додавання алгоритмів класифікатора помилок (АКП).

Є сенс в тому, щоб АПВ, АПП і АТПП працювали разом, перш ніж додавати їх до класифікатора помилок. Отже, надалі буде перевірена продуктивність алгоритмів класифікатора помилок (АКП), щоб побачити, як вони покращують продуктивність лише у випадку помилок передачі.

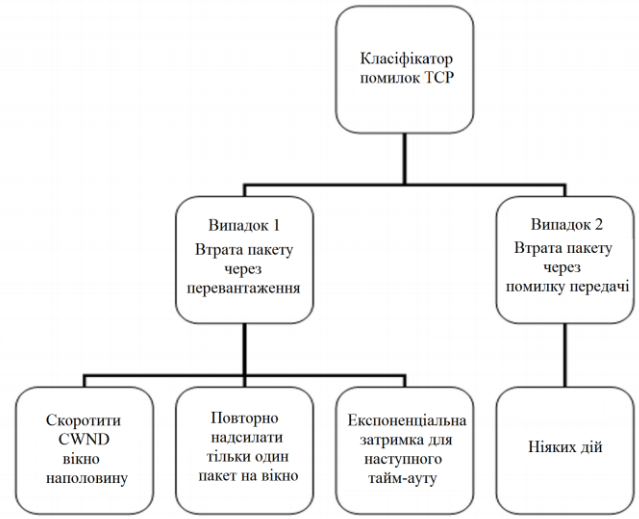

Рисунок 4.11 – Загальні функції класифікатора помилок

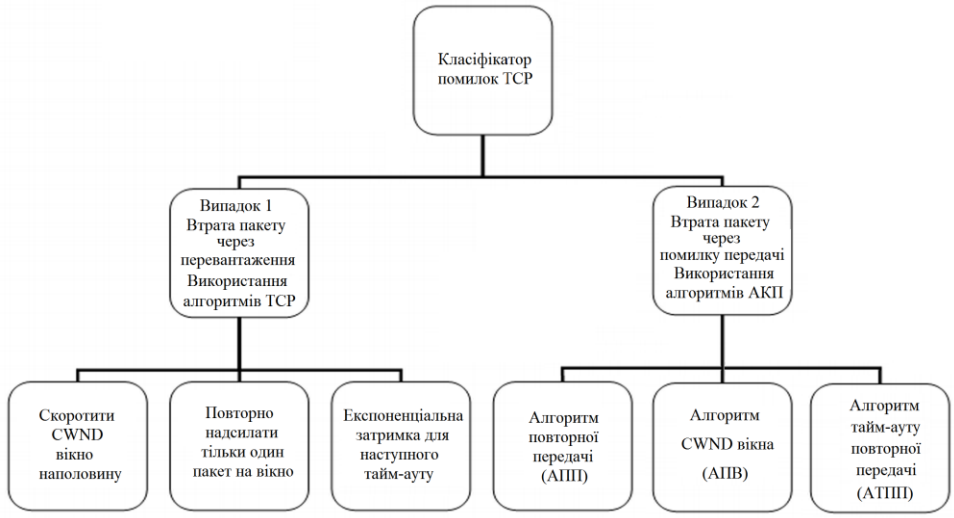

Рисунок 4.12 – Класифікатор помилок після додавання АКП

5 ІМІТАЦІЙНА МОДЕЛЬ

5.1 Умови моделювання

В якості середи моделювання було обрано ns3, який є одним із широко використовуваних симуляторів у дослідженнях комп’ютерних мереж. Крім того, оскільки робота тісно пов’язана з TCP, цікаво знати, що реалізація TCP у ns3 була розроблена групою комп’ютерних систем та інженерії Каліфорнійського університету, яка реалізувала деякі з перших і популярних версій TCP, як-от TCP Tahoe в UNIX 4.3BSD-Tahoe і TCP-Reno в UNIX 4.3BSD-Reno.

Використовується мережева топологія вузького місця, яка зазвичай використовується в оцінці TCP і представлена на рисунку 5.1. У цій топології є два види джерел: джерела TCP (TB) і джерела UDP (UB). У всіх експериментах перше джерело TCP (TB1) застосовуватиме модифікований протокол, який треба перевірити.

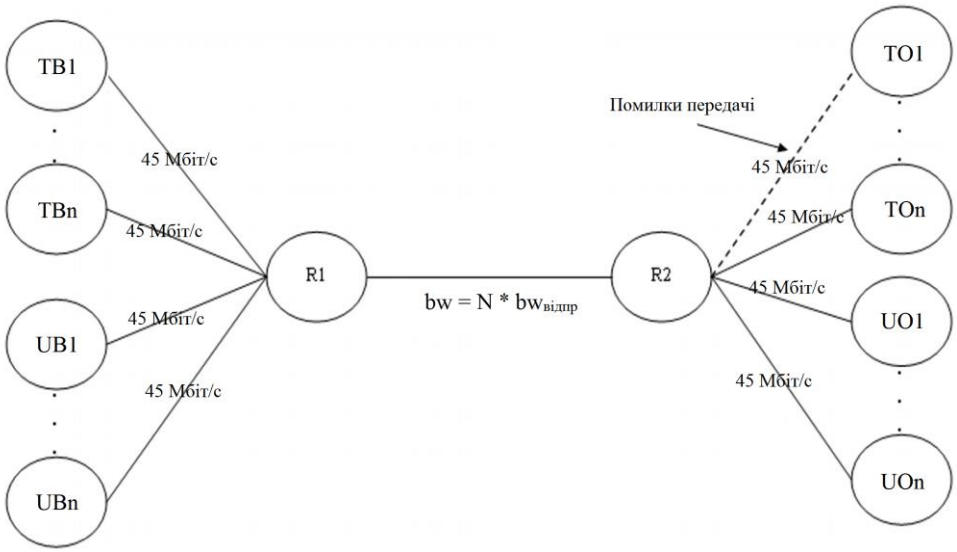

Рисунок 5.1 – Топологія мережі

Адресати для джерел TCP називаються TO, а адресати для джерел UDP називаються UO. Шлях до пунктів призначення проходить через два проміжні маршрутизатори R1 і R2. Маршрутизатори використовують черги з відкиданням хвоста. Розміри буферів дорівнюють добутку смуги пропускання на затримку зв'язку в вузькому місці (Bandwidth-Delay product = bandwidth у бітах на секунду, помножена на загальну затримку в секундах і поділена на розмір пакета).

Такий підхід дозволяє запобігти створенню неконтрольованих перевантажень на шляху підключення. Шлях до TO1 містить помилки передачі на останньому переході, які будуть використані для тестування запропонованих алгоритмів.

В експериментах змінюється пропускна здатність і затримка лише тоді, коли необхідно виміряти їхній вплив на продуктивність. Однак основна увага зосереджена на впливі помилок передачі, тому, коли змінюється частота помилок, пропускна здатність фіксується на типовому з'єднанні T3 (45 Мбіт/с), а максимальна загальна затримка розповсюдження фіксується на рівні 48 мс. Однак фактична загальна затримка може змінюватися через затримку в черзі, навіть якщо затримку розповсюдження було фіксовано.

Пропускна здатність вузького місця bw_{bn} і затримка, встановлюються відповідно до того, чи необхідно створити перевантаження чи ні. Якщо не треба зменшувати перевантаження, тоді bw_{bn} встановлюється на загальну пропускну здатність усіх джерел, так що $bw_{bn} = N * bw_s$, де N – кількість джерел.

Однак, якщо треба створити перевантаження, встановлюється значення bw_{bn} не більше ніж 80% сукупної пропускної здатності джерела та налаштовується $pdly_{bn}$, доки не отримаємо необхідний рівень перевантаження.

Створюється одне вузьке місце між R1 і R2 через те, що пропускна здатність і затримка з'єднань від R2 до вузлів призначення, приймають ті самі значення, що й пропускна здатність і затримка від джерел до R1.

Трафік для джерел TCP генерується за допомогою програм FTP у ns3. Розмір пакета TCP вибрано рівним 1 КБ (1024 байт), що близько до значення за замовчуванням у ns3 (1000 байт).

Моделювання помилки передачі використовує марковський ланцюг із двома станами, який є простою моделлю, яка широко використовується для моделювання наявності вибухів у помилках бездротового зв'язку [14]. Модель помилки складається з двох станів: хорошого стану (без помилок) і поганого стану (помилки). Система може бути в одному стані в будь-який час.

Тривалість у кожному стані є випадковим числом з експоненціальним розподілом. У більшості експериментів експеримент проводиться щонайменше на 100 секунд (100 000 мс), а збір даних починається через 5 секунд, щоб усунути будь-які початкові ефекти та дати мережі налаштуватися.

5.2 Показники продуктивності

Пропускна здатність/хороша пропускна здатність: пропускна здатність – це показник для вимірювання обсягу переданих даних за певний проміжок часу (наприклад, кількість байтів на секунду). Проте пропускна здатність включає як нові, так і повторно передані дані.

Хороша пропускна здатність – це підмножина пропускної здатності, яка враховує лише дані, які безпечно досягли пункту призначення, без урахування повторних передач, і це більш точний показник для вимірювання продуктивності, коли треба знати швидкість фактичних даних, які були доставлені (що що важливо для кінцевого користувача). Через це продуктивність використовується як основний показник ефективності в експериментах. Однак, оскільки пропускна здатність є більш поширеним терміном, то вона використовується для позначення хорошої пропускної здатності, якщо не вказано інше.

Крім того, у багатьох випадках використовується нормалізована хороша пропускна здатність (нормалізована продуктивність), яка є корисною пропускною здатністю для кожного потоку над максимальною досяжною продуктивністю та коливається від 0 до 1. Максимальна досяжна ефективна пропускна здатність – це пропускна здатність вузького місця.

Розмір перевантажувального вікна: перевантажувальне вікно – це унікальна метрика TCP, яка контролює кількість пакетів, які TCP може надіслати до отримання підтвердження. Це тісно пов’язане зі швидкістю надсилання TCP.

Кількість/тривалість подій тайм-ауту (RTO): коли відбуваються скидання пакетів та підтвердження перестають надходити до відправника TCP, відправник чекає настання тайм-ауту, щоб продовжити надсилання. Однак тривалість цього періоду очікування впливає на продуктивність відправника та мережі одночасно. Занадто довгі тайм-аути призведуть до зниження продуктивності TCP, а надто короткі тайм-аути можуть збільшити перевантаження в мережі.

Також важлива частота тайм-аутів, оскільки з кожною подією тайм-ауту перевантажувальне вікно TCP скидається до одного сегмента.

Коефіцієнт втрат від перевантаження: це кількість скинутих пакетів із вузького місця намаршрутизаторі на загальну кількість переданих пакетів.

Коефіцієнт втрат при передачі: це кількість викинутих пакетів через помилки, пов’язані з неперевантаженням, на загальну кількість переданих пакетів.

5.3 Оцінка ефективності запропонованих алгоритмів

Пропускна здатність вузького місця дорівнюватиме загальній пропускній здатності всіх відправників (тобто для N джерел $bw_{bn} = N*bw_s$), щоб уникнути перевантаження. Щоб показати ефект додавання запропонованих алгоритмів до TCP, треба порівняти його з пропускною

здатністю TCP, використовуючи ту саму топологію та експериментальні умови. Запускається експеримент кілька разів кожного разу з різним початковим значенням для генератора випадкових чисел, щоб сгенерувати різні шаблони трафіку та помилок. Частота помилок передачі коливається від 0,001 до приблизно 0,1 із кроком збільшення на 0,01 (тому використовуються точні частоти помилок: 0,001, 0,011, 0,021, 0,031, 0,041, 0,051, 0,061, 0,071, 0,081, 0,091 і 0,10 частота помилок). Цей діапазон охоплює низький, середній і високий рівень помилок передачі.

На рисунку 5.2 показано продуктивність TCP до і після додавання запропонованих алгоритмів у напівлогарифмічному масштабі. Хороша пропускна здатність нормалізується відповідно до справедливої частки пропускної здатності вузького місця кожного потоку.

Також порівнюються запропоновані алгоритми з двома варіаціями TCP, TCP-NewReno і TCP-Sack. Однак TCP-NewReno та TCP-Sack мають перевагу перед TCP у тому, що вони можуть відновлювати кілька втрачених пакетів з одного вікна. На рисунку 5.3 представлено продуктивність TCP-Sack. На рисунку 5.4 додається TCP-NewReno. Як можна побачити, у всіх випадках запропоновані алгоритми мають найвищу продуктивність.

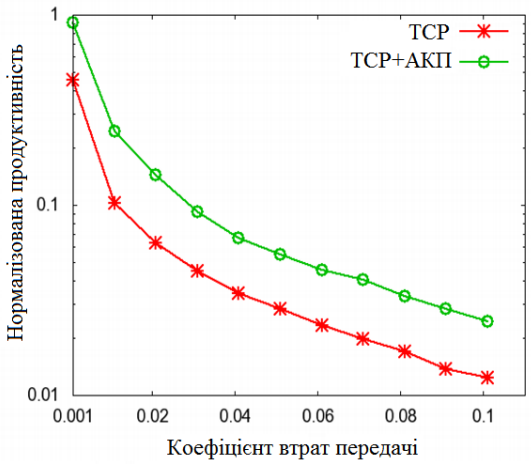

Рисунок 5.2 – Нормалізована продуктивність TCP і запропонованих алгоритмів (напівлогарифмічний масштаб)

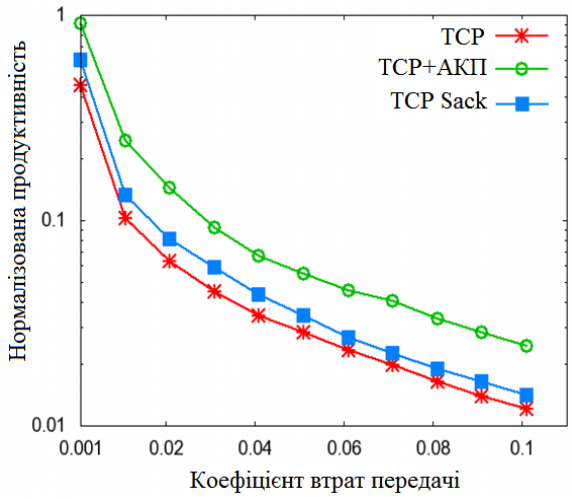

Рисунок 5.3 – Нормалізована продуктивність TCP, Sack і запропонованих алгоритмів

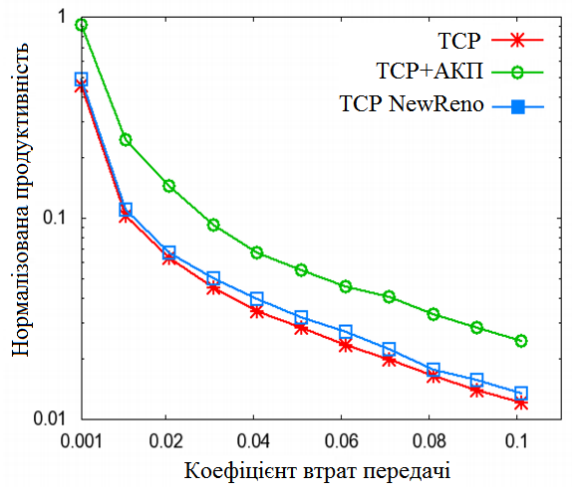

Рисунок 5.4 – Нормалізована корисна продуктивність TCP, запропонованих алгоритмів та NewReno

Однак можна помітити, що через різні методи, які використовуються NewReno і Sack для відновлення після втрати кількох пакетів, Sack має більші переваги порівняно з TCP. Причина, чому Sack має вищу продуктивність, ніж NewReno, полягає в тому, що NewReno може відновлювати один пакет за кожен RTT, тоді як Sack може відновлювати кілька вилучених пакетів за один RTT. Крім того, оскільки NewReno може

відновлювати один пакет за RTT він має дуже незначне покращення з низькими показниками помилок (понад 0,001 частота помилок NewReno вище, ніж TCP приблизно на 6%, а максимальне покращення становить 16% порівняно з частотою помилок 0,061).

Середнє покращення запропонованих алгоритмів порівняно з TCP становить 105%. Незважаючи на те, що це покращення здається значним, подібні покращення зазвичай відбуваються, коли TCP уникає скорочення перевантажувального вікна.

Основні причини, чому АКП також може перевершити Sack і NewReno, хоча вони можуть відновлювати кілька скинутих пакетів, полягають у тому, що окрім АПП, який дозволяє АКП відновлювати множинні скидання пакетів, АКП також застосовує АПВ, який зменшує швидкість скорочення перевантажувального вікна на 50% кожного разу, коли відкидання відбувається зі швидкістю, яка дорівнює кількості відкинутих пакетів з кожного вікна. Крім того, використання АТПП у АКП зменшує тривалість подій тайм-ауту.

Щоб детальніше дослідити вплив кожного компонента АКП (АПВ, АПП та АТПП), показано на рисунку 5.5, як АКП збільшує середній розмір cwnd порівняно з TCP.

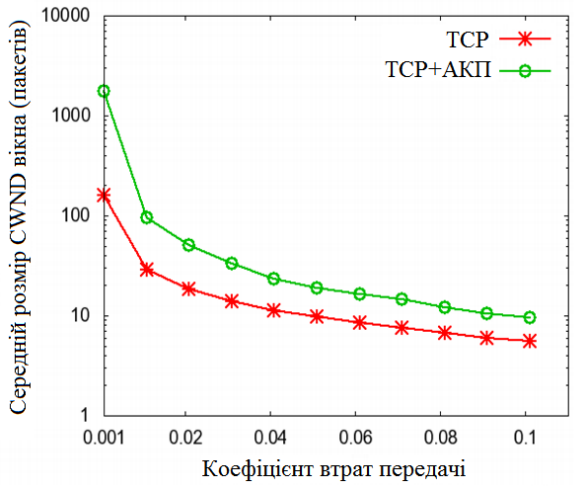

Рисунок 5.5 – Середній розмір перевантажувального вікна для TCP і АКП

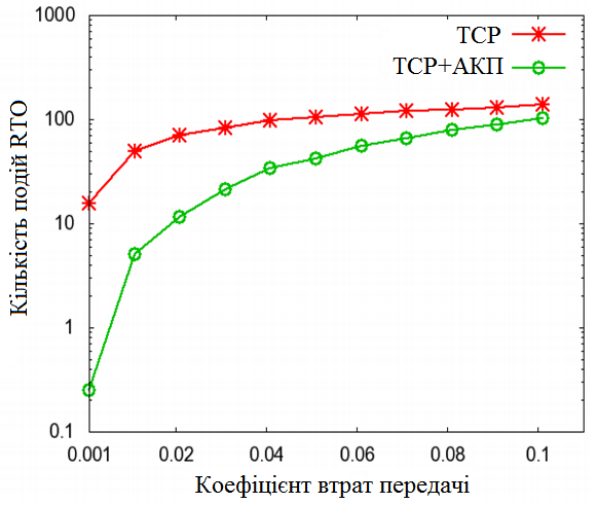

Рисунок 5.6 – Кількість подій тайм-ауту повторної передачі

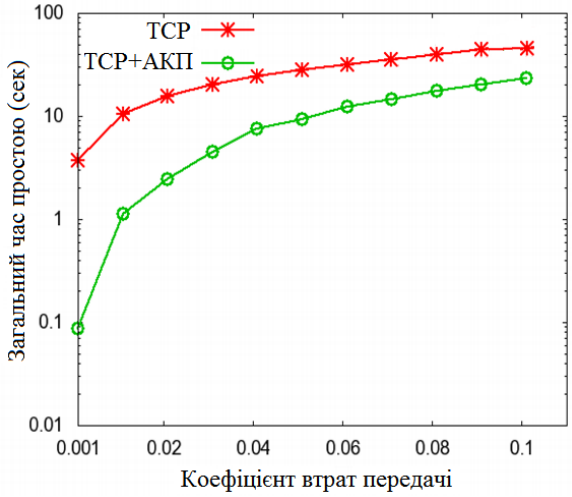

Рисунок 5.7 – Загальний час простою для TCP і АКП

Це збільшення в основному пов’язано з компонентом АПВ АКП. Також на рисунку 5.6 показано зменшення кількості RTO, викликане застосуванням АПП. Нарешті, АТПП зменшує довжину RTO і, отже, час простою TCP, як можна побачити на рисунку 5.7. Усі ці фактори вплинули на підвищення продуктивності представлені на рисунку 5.5.

ВИСНОВКИ

Було запропоновано рішення для подолання проблеми погіршення продуктивності TCP у гетерогенних мережах, де можуть співіснувати перевантаження та помилки передачі. Одним із можливих рішень є розробка класифікаторів помилок передачі, які можна додати до TCP, щоб покращити його механізм контролю перевантажень і таким чином підвищити продуктивність. Перевага використання такого підходу полягає в тому, що він вимагає змін лише в точці джерела пакетів, тому не потрібні зміни в мережі.

Поточні класифікатори помилок просто пригнічують TCP і не дозволяють йому скоротити перевантажувальне вікно у разі помилок передачі. Це призводить до збільшення рівня втрат від перевантаження мережі при використанні таких механізмів. Було виявлено, що поточні дії, які використовуються в класифікаторах помилок, є недостатніми та можуть призвести до збільшення рівня втрат через перевантаження мережі.

Було запропоновано поліпшення до механізмів перевантажувального вікна, механізму повторної передачі та механізму тайм-ауту.

Результати моделювання показують, що застосування запропонованого класифікатора помилок передачі дозволяє досягти вищої продуктивності, ніж у TCP. Крім того, він має набагато нижчий рівень втрат перевантаження, меншу наскрізну затримку та створює менший розмір черги, ніж класичний TCP.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Van Jacobson, Michael J. Karels, “Congestion Avoidance and Control”,Sigcomm '88 Symposium, vol.18 (4): pp.314–329. Stanford, CA. August,1988.

2. Navraj Chohan, “An Analysis of TCP through Simulation”, Technicalre port CS 276, 2006.

3. Jitendra Padhye, Victor Firoiu, Donald F. Towsley, and James F. Kurose,“Modeling TCP Reno Performance: A Simple Model and Its EmpiricalValidation”, IEEE/ACM Transactions on Networking, vol. 8, no. 2, pp:133-145,, April 2000.

4. S. Floyd, T. Henderson, “The NewReno Modification to TCP's FastRecovery Algorithm”, RFC 2582, April 1999

5. M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, “TCP SelectiveAcknowledgment Options”, RFC 2018, October 1996.

6. Lawrence S. Brakmo, Sean W. O’Malley, Larry L. Peterson,” TCP Vegas: New Techniques for Congestion Detection and Avoidance”, ACMSIGCO MM Computer Communication Review, vol. 24 , Issue 4, pp:24-35,,October 1994

7. Wilson Boulevard, Admiralty Way,” Transmission Control Protocol”, RFC793, September 1981.

8. W. R. Stevens, “TCP/IP Illustrated, Volume 1 the Protocols”, AddisonWesley Professional, Oct. 1993.

9. S. J. Golestani, and S. Bhattacharyya, ”A Class of End-to-End CongestionControl Algorithms for the Internet”, the Sixth International Conference on Network Protocols, pp:137-151 , ISSN:1092-1658 , 1998

10. Sonia Fahmy, Tapan P. Karwa, “Tcp congestion control: Overview andSurvey of Ongoing Research”, Technical report, PurdueUniversity, 2000.

11. Chadi Barakat, “TCP/IP Modeling and Validation”, IEEE Network, vol.15Issue 3, pp: 38-47, May 2001.

12. Sally Floyd and Kevin Fall, “Promoting the Use of End-to-End CongestionControl in the Internet”, IEEE/ACM Transactions On Networking, vol. 7,no. 4, pp: 458-472, AUG 1999.

13. L. Rizzo, “TCP retransmissions on very lossy networks,” 1996, http://info. iet.unipi.it/~luigi/tcp.ps (accessed October 2007).

14. B. Paolo, Packet Loss in Terrestrial Wireless and Hybrid Networks. PhD thesis, University of Pisa, 2007.

15. А.М. Андрос, С.О. Партика, О.А. Янковський, “Нелінійна динаміка трафіку із застосуванням AQM RED”, Десята міжнародна науково-технічна конференція «Проблеми інформатизації». – Черкаси-Баку-Бельсько-Бяла-Харків. 24 – 25 листопада 2022 р. – С. 34.