ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ І. І. МЕЧНИКОВА
(повне найменування закладу вищої освіти)

Факультет математики, фізики та інформаційних технологій
(повне найменування факультету)

Кафедра механіки, автоматизації та інформаційних технологій
(повна назва кафедри)

Кваліфікаційна робота

на здобуття ступеня вищої освіти «бакалавр»

«Оптимізація синхронізації розподілених застосунків у комп'ютерних мережах»

«Optimize the synchronization of distributed applications across computer networks»

Виконав(ла): здобувач(ка) денної (очної) форми навчання
спеціальності 122 Комп'ютерні науки

Освітня програма Комп'ютерні науки

Максимова Юнна Артурівна
(прізвище, ім'я, по-батькові здобувача)

Керівник викладач Недєва О. А.
(науковий ступінь, вчене звання, прізвище, ініціали) (підпис)

Рецензент канд. фіз.-мат. наук, доцент Рачинська А. Л.
.
(науковий ступінь, вчене звання, прізвище, ініціали)

Рекомендовано до захисту:
Протокол засідання кафедри
механіки, автоматизації та інформаційних технологій
№ ___ від ____.____. 20___ р.

Завідувачка кафедри
_____ Алла РАЧИНСЬКА
(підпис)

Захищено на засіданні ЕК № _____
протокол № __від ____.____.20___ р.

Оцінка_____/____/_____
(за національною шкалою/шкалою ECTS/ бали)

Голова ЕК
_____ Микола МАЛАКСІАНО
(підпис)

Одеса 2024

АНОТАЦІЯ

У роботі розглядаються методи, засоби та обмеження задач синхронізації у комп'ютерних мережах. Зокрема, проводиться аналіз параметрів синхронізації для мережної інфраструктури, здійснюється огляд наявних засобів синхронізації, включаючи протоколи та бібліотеки. Розглядаються упрощена та афінна моделі та параметри синхронізації годинника. Висвітлюються фундаментальні обмеження як однокрокової так і багатокрокової синхронізації годинників в мережах.

Також у роботі представлені розширені підходи до завдань синхронізації. Розглядаються інтервали оцінки неузгодженості тимчасових міток клієнта та сервера, а також методи їх коригування. Описується парадігма синхрокоординації розподілених додатків реального часу та її роль у замкнутих системах.

Заключна частина роботи - проведення дослідження характеристик каналів зв'язку для задач синхронізації, для чого розробляється програма моніторингу параметрів синхронізації в мережі, описується її архітектура та оптимізація параметрів синхронізації реальних мереж.

ABSTRACT

This paper considers the methods, means and limitations of time synchronization in computer networks. In particular, the synchronization parameters for the network infrastructure are analyzed, and the available synchronization tools, including protocols and libraries, are reviewed. Simplified and affine models and parameters of clock synchronization are considered. The fundamental limitations of multistage clock synchronization in networks are highlighted.

The paper also presents advanced approaches to synchronization tasks. The intervals for assessing the inconsistency of client and server timestamps, as well as methods of their adjustment, are considered. The paradigm of synchro-coordination of real-time distributed applications and its role in closed systems is described.

The final part of the work is a study of the characteristics of communication channels for synchronization tasks, for which a program for monitoring synchronization parameters in the network is developed, its architecture and optimization of synchronization parameters of real networks are described.

ЗМІСТ

ВСТУП

Синхронізація часу та синхрокоординація дій - це фундаментальні аспекти у забезпеченні надійності та ефективності роботи комп'ютерних мереж, за допомогою яких працюють розподілені системи, мультиагентні системи, та системи реального часу. В таких системах узгоджена інформація вузлів мережі про час відіграє критичну роль. Порушення синхронізації як правило призводить до неправильної інтерпретації дій, рішень, даних, що спричиняє збої системи або некоректні результати обчислень. Особливо це стосується областей, де потрібна висока точність та надійність розрахунків, таких як медичні системи, фінансові транзакції, системи управління чи критичні виробничі процеси. Складні системи, в яких вузли повинні оперувати в єдиній часовій системі, неминуче стикаються з проблемами неузгодженості годинників, розкоординації дій та часових затримок. Навіть якщо годинники налаштовані точно, через певний проміжок часу між ними неминуче почнуть виникати розбіжності в часі. Ці проблеми можуть виникати через фізичні фактори, такі як різні швидкості тактових генераторів у пристроях, або через мережні затримки, викликані маршрутизацією даних через різні вузли. Розробка методів, алгоритмів та протоколів, здатних подолати ці відмінності та забезпечити синхронізацію часу за таких умов - це задачі для академічних досліджень і ключові питання для промислової практики.

На сьогоднішній день представлені і широко використовуються протоколи синхронізації комп'ютерних мереж NTP (Network Time Protocol) з точністю синхронізації до 10 мілісекунд в глобальних мережах і PTP (Precision Time Protocol) в локальних, точність якої вже обчислюється в мікросекундах. Назважаючи на довге існування і широке використання ціх протоколів, можна виділити такі практичні передумови завдання данної роботи:

1) Операційні системи (ОС) можуть використовувати ці протоколи або інші методи синхронізації часу з відповідними серверами, але ці, вбудовані в ОС методи, можуть бути відключеними.
2) Точність синхронізації для глобальних мереж не є достатньою для сучасних розподілених програмних систем.

3) Браузери (і відповідно браузерні програми, що працюють на сьомому рівні моделі стеку протоколів OSI) не використовують протоколи NTP та PTP для синхронізації часу в розподілених системах. Зазвичай браузери отримують інформацію про час безпосередньо від операційної системи пристрою, де вони запущені.

Тому, якщо потрібно синхронізувати час у веб-застосунку, треба використовувати JavaScript для взаємодії з серверами за допомогою вбудованих в браузери API AJAX запитів, метода fetch() для протоколу HTTP, або вбудованих методів протоколу WebSockets. Реалізація задач синхронізації залежить від кількості клієнтських вузлів, топології зв'язку вузлів, дистанції до сервера (або кластеру серверів) та можливостей реалізації хостінгових додатків. На сьогоднішній день існують бібліотеки, які здійснюють синхронізацію на базі протоколів прикладного рівня. Наприклад, це відкрита бібліотека timesync, а також платна бібліотека unichron. Однак методи синхронізації timesync не достатньо обгрунтовані - не гарантують монотонність синхронізації, та можуть забезпечити синхронізацію лише з точністю не більшою декількох мілісекунд, чого може бути недостатньо для розподілених додатків, де точність синхронізації відіграє ключову роль. Платна бібліотека unichron орієнтована на синхронізацію вузлів, що належать до одного кластеру хмарного хостінгу і не можуть бути безболісно модіфіковани для інших потреб.

Завдання даної роботи полягає в обгрунтуванні та розробці методів оптимізації параметров синхронізації розподілених Web-додатків, що дасть можливість підвищити точність синхронізації, наприклад, перевершуватимуть точність бібліотеки timesync, в розробці і моделюванні парадигми синхрокоординації, а також в побудові системи моніторингу параметрів, що впливають на синхронізацію розподілених клієнт серверних систем. Код додатків є відкритий для загального доступу та подальших модернізацій.

МЕТОДИ, ЗАСОБИ ТА ОБМЕЖЕННЯ СИНХРОНІЗАЦІЇ ЧАСУ В КОМП'ЮТЕРНИХ МЕРЕЖАХ

1.1 Параметри синхронізації для мережевої інфраструктури

Основною метою існуючих методів синхронізації є визначення та подальше коригування параметрів, що характеризують поточний стан годинників мережі. Синхронізація здійснюється в результаті обміну повідомленнями з часовими помітками між вузлами мережі, наприклад, між клієнтами та сервером для клієнт-серверних додатків. Ці показання годинників використовуються для розрахунку параметрів синхронізації в кожен момент сумісноє роботи вузлів та визначення неузгодженості між часовими системами (Рисунок. 1.1) В результаті цього обміну клієнт повинен обчислити параметри: rtt (round-trip-delay, Δ), тобто час, який потрібен на відправку запиту на сервер плюс час, за який відповідь від сервера буде отримана, та θ (offset, часове зміщення годинника). Для більш точного визначення стану годинників ще обчислюється відносний дрейф σ годинників клієнтів до серверного годинника.

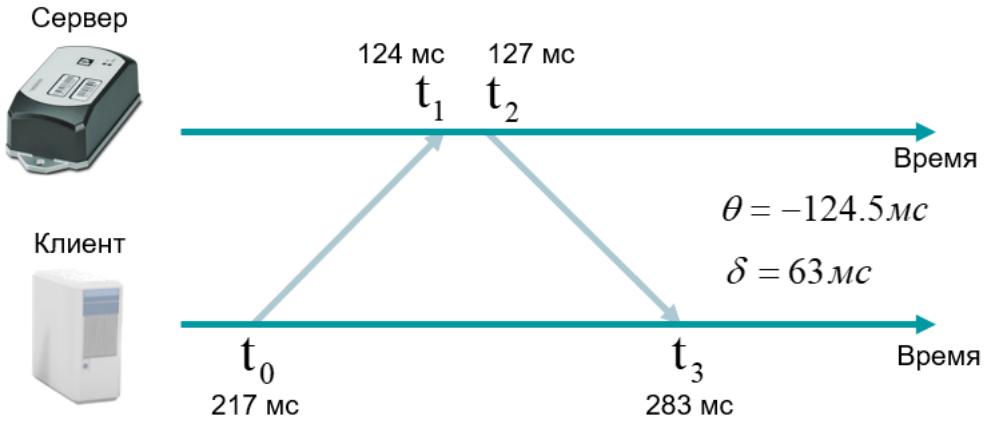

Рисунок. 1.1. — Базовий принцип синхронізації годинників.

1.2 Огляд існуючих засобів синхронізації

Існує багато різноманітних засобів синхронізації, кожен з яких має своє призначення, переваги та недоліки. У цьому розділі розглянемо основні категорії засобів синхронізації: протоколи та бібліотеки.

1.2.1 Протоколи синхронізації

Протокол NTP був розроблений в 1985 Девідом Міллсом в Університеті Делавера. NTP забезпечує синхронізацію в глобальних мережах з точністю до 10 мілісекунд. Спочатку NTP був створений для поширення часової інформації в мережі для систематичної передачі національного стандартного часу через Інтернет по дротових або радіоканалах.

Остання версія протоколу, NTPv4, широко використовується для синхронізації локального годинника комп'ютерних систем у мережах з комутацією пакетів щодо координованого всесвітнього часу (UTC). Основні привабливі особливості NTP включають його масштабованість, стійкість до відмов, можливість само-конфігурації у великих багатоланкових мережах і поширення.

NTP є багаторівневою клієнт-серверною архітектурою, заснованою на повідомленнях UDP (User Datagram Protocol), які синхронізують годинник ієрархічно. Модель підмереж NTP включає множину загальнодоступних первинних серверів часу, які також синхронізуються з національними стандартами через дротові або радіоканали. Завдання протоколу NTPv4 полягає в передачі часової інформації від цих первинних серверів до вторинних серверів часу та клієнтів через приватні мережі, так і публічний Інтернет.

Точна синхронізація досягається, коли маршрути між клієнтом та сервером мають симетричну номінальну затримку. Для цього ідеального випадку виникає детерміноване зміщення передачі, що дозволяє легко обчислювати параметр синхронізації. Але реальні мережі є асиметричними [3]. Було запропоновано множина підходів для вимірювання асиметрії, але всі вони є дуже затратними. Якщо, нам вже відома асиметрія каналу, з практичних реалізацій тільки chrony (системний клієнт і сервер протоколу мережевого часу, призначений для синхронізації системних годинників із зовнішніми серверами NTP) включає функцію корекції параметрів синхронізації.

Однією з основних проблем NTP є те, що він не враховує мережеві затримки та буферизації для точних часових додатків. У локальній мережі (LAN) на основі NTP точність годинників не ліпше 1-2 мілісекунд через

затримку і джиттер, доданих мережними пристроями. У глобальних мережах (WAN) точність синхронізації годинників на основі NTP знижується до 10-20 мілісекунд.

Реалізація NTPv4 не відповідає вимогам високої точності для архітектур синхронізації в мобільній мережі LTE (Long Term Evolution) або мережі 4G. Наприклад, розгортання систем LTE або 4G вимагає точності синхронізації часу/фази в межах 1 мікросекунди та точності частотної синхронізації в межах від 50 до 16 ppb (parts per billion).

Для систем, де точність синхронізації є критично важливою (мікросекунди і навіть наносекунди), 2002 року було розроблено протокол PTP (Precision Time Protocol). Цей протокол розроблено організацією IEEE (Institute of Electrical and Electronics Engineers) та визначено у стандарті IEEE 1588. Протокол PTP [3] був спочатку розроблений компанією Agilent Technologies у період з 1990 по 1998 роки для тестування та застосування у промисловій автоматизації. PTP пропонує високу точність та економічність, порівнянну з протоколом NTP [13], використовуючи існуючі мережі Ethernet, що підтримують мультимасштабні комунікації. Протокол дозволяє досягти точності синхронізації в межах субмікросекундного діапазону. Нова версія IEEE 1588 була опублікована в 2008 році і включає покращені функції та підвищену продуктивність.

Протокол IEEE 1588 вирішує більшість проблем із затримкою та джиттером, характерних для NTP, за рахунок апаратної мітки часу фізично мережі. Важливо відзначити, що стандарт працює дуже добре при симетричних затримках. Однак у реальності затримки пакетних мережах не є симетричними, і односпрямовані затримки можуть відрізнятися. Щоб мінімізувати вплив асиметрії, PTP ввів спеціалізовані елементи, відомі як граничний годинник (boundary clocks) та прозорі комутатори (transparent switches) з додатковими функціями для збереження точності. З додаванням граничного годинника або прозорих комутаторів протокол здатний досягти точності в діапазоні від 20 до 100 наносекунд у локальній мережі (LAN) і в ідеальних умовах до 10 мікросекунд у глобальній мережі (WAN).

1.2.2 Бібліотеки синхронізації

Бібліотека timesync є однофайловою реалізацією синхронізації часу по мережі, що підходить для різних пристроїв, включаючи мобільні (iOS та Android), а також комп'ютери (Linux, Windows, Mac), використовуючи UDP/IP сокети. Новий метод синхронізації часу має низку переваг у порівнянні з попередніми реалізаціями, такими як PTP та NTP, особливо при роботі в умовах високих джиттерів, характерних для стільникових мереж.

Основний принцип роботи алгоритму полягає в тому, що кожен відправлений пакет відзначається часовою міткою відправника, яка потім порівнюється з часовою міткою, проставленою одержувачем, що дозволяє обчислити затримку та відхилення часу між двома пристроями. Повторюючи цей процес багаторазово і аналізуючи результати, алгоритм відкидає аномальні значення і використовує середнє значення результатів для корекції часу. Цей підхід мінімізує вплив мережевих затримок та джиттера, забезпечуючи точну синхронізацію.

Переваги даної бібліотеки включають високу відносно високу точність синхронізації, що досягається за рахунок великої кількості проб і статистичної обробки даних, що дозволяє забезпечити мілісекундну точність для розрахованих на багато користувачів ігор і 16-мікросекундну точність для наукових додатків. Використання компактних часових міток (2 або 3 байти) знижує мережне навантаження, а підтримка різних платформ, включаючи мобільні пристрої та комп'ютери, забезпечує гнучкість та широкі можливості застосування. Бібліотека особливо ефективна в умовах високої нестабільності мережі, що робить її ідеальною для використання в стільникових мережах.

У відповідь на потребу у високій точності часової синхронізації було створено систему UniChron. Вона призначена для забезпечення акуратної синхронізації часу з динамічними межами для будь-якого середовища. Принцип роботи UniChron ґрунтується на поданні кожної часової мітки у вигляді інтервалу (begin, end), з точністю до субмілісекундних меж. Ця система підтримує як програмні, так і апаратні позначки часу і може функціонувати в будь-якому середовищі, від хмарних платформ до корпоративних мереж.

UniChron забезпечує покращену точність на три порядки вище, ніж типовий протокол мережного часу (NTP), що підвищує надійність та ефективність розподілених систем. На відміну від традиційних методів, UniChron скорочує час збіжності синрхронізації годинника з годинником, забезпечуючи поліпшення продуктивності у додатках розподілених баз даних. UniChron також дозволяє скоротити повторні спроби синхронізаціїї, що знижує навантаження на системи і підвищує їх ефективність.

1.3 Спрощена модель і параметри синхронізації годинників

Для розробки та аналізу алгоритмів синхронізації годинників і, у ширшій постановці, часовому узгодженні роботи розподілених додатків, слід визначити на базі відповідних формальних моделей фундаментальні обмеження завдань синхронізації годинника в комп'ютерних мережах. Зручно аналізувати простий алгоритм синхронізації, запропонований Крістіаном [5], у якому послідовно виконуються прості кроки, що оцінюються часовими змінними. У процесі синхронізації годинника необхідно враховувати безліч різних факторів, що впливають на точність і ефективність цього процесу (наприклад, тип матеріалу і характеристики часозадаючих елементів годинників - цезій, кварц і т.п., тип і довжина ліній зв'язку - волоконно-оптичні, радіорелейні, провідні і т.д., рівень часзалежних додатків та обладнання в стеку OSI). Залежно від специфіки завдання та конкретних умов деяким із цих факторів можна відвести другорядну роль або навіть знехтувати ними. Однак, є ціла низка параметрів, які завжди впливатимуть на стан синхронізації системи.

Для наочності розглянемо систему, що складається з n вузлів, у тому числі $(n-1)$- це звичайні годинники (клієнти), які працюють на кварцових генераторах. Для коригування часу кожен вузол у системі періодично звертається до умовного "точного годинника" - сервера (годинник якого теж найчастіше працює на кварцових генераторах). Клієнти та сервер можуть бути розташовані на дуже великих відстанях, тому не можна нехтувати часом поширення сигналів у прямому та зворотному напрямках. Крім того, немає жодної гарантії, що час поширення сигналів у прямому та зворотному напрямку стабільні та рівні (Рис. 1.2). Далі буде розглянута спрощена

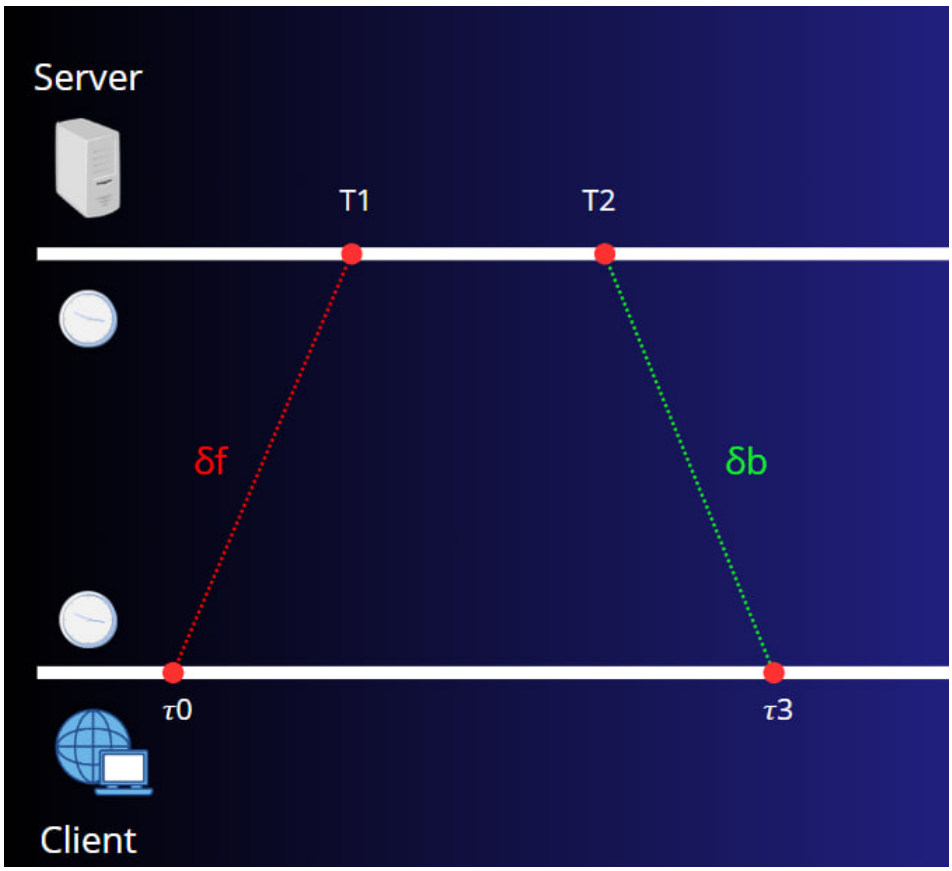

Рисунок. 1.2. — Базовий обмін повідомленнями для синхронізації годинників.

детермінована модель годинника та обміну сигналами, в якій у годинника відсутній дрейф, а канали зв'язку не схильні до флуктуацій, для оцінки фундаментальних обмежень завдання синхронізації.

У спрощеній детермінованій моделі годинника та обміну сигналами, причому для симетричного каналу зв'язку, можна визначити θ за формулою (1) (для реальних систем це буде початкова оцінка параметра θ):

$$\tilde{\theta}_0 = \frac{(T_1 - \tau_0) + (T_2 - \tau_3)}{2} \tag{1}$$

де

τ_0 — часова помітка клієнта про передачу пакета запиту,

T_1 — часова помітка сервера прийому пакета запиту,

T_2 — часова помітка сервера передачі пакету у відповідь,

τ_3 — часова помітка клієнта про прийом пакету у відповідь.

Нехай Δ - кругова затримка (rtt) поширення сигналу від клієнта до

сервера і назад:

$$\Delta = (\tau_1 - \tau_0) - (\tau_3 - \tau_2) \tag{2}$$

Вираз для Δ можна переписати, використовуючи точки τ_1 і τ_2 на часовій шкалі клієнта (Рис.1.2), що відповідають моментам T_1 прийому повідомлення на сервері та T_2 відправки повідомлення клієнту:

$$\Delta = (\tau_1 - \tau_0) - (\tau_3 - \tau_2) = \delta_f + \delta_b, \tag{3}$$

де δ_f — час поширення сигналу від клієнта до сервера, а δ_b — назад. Тоді легко показати, що вираз (3) отримано як розв'язок системи лінійних рівнянь (2), за умови, що час відправки пакета даних на сервер дорівнює часу відправки пакета до клієнта:

$$\begin{cases} \tau_0 + \tilde{\theta}_0 + \frac{1}{2}\Delta = T_1 \\ \tau_3 + \tilde{\theta}_0 - \frac{1}{2}\Delta = T_2 \end{cases}, \tag{4}$$

Незважаючи на те, що модель (4) перевизначена (два рівняння щодо однієї змінної), вона не є суперечливою і її можна вирішити у вигляді (1). Крім того, у цьому ідеальному симетричному випадку достатньо одного етапу обміну повідомленнями між клієнтом та сервером для точної синхронізації. Однак у реальних системах більшість каналів зв'язку не симетричні (Рис. 1.3), тобто час поширення сигналу в одному напрямку може відрізнятися від часу розповсюдження у зворотному напрямку. Це створює труднощі за точної синхронізації систем, оскільки формула (1) не враховує асиметрію каналу і може дати лише початкову оцінку параметрів синхронізації.

Припущення про симетричність каналу зв'язку є дуже оптимістичним. Тому систему (4) слід переписати у вигляді системи з обмеженням (5) з урахуванням можливої асиметрії каналу:

$$\begin{cases} \tau_0 + \tilde{\theta}_0 + (p)\Delta = T_1 \\ \tau_3 + \tilde{\theta}_0 - (1-p)\Delta = T_2 \\ \quad 0 < p < 1 \end{cases}, \tag{5}$$

де p – параметр асиметрії (для симетричних каналів $p = 1/2$).

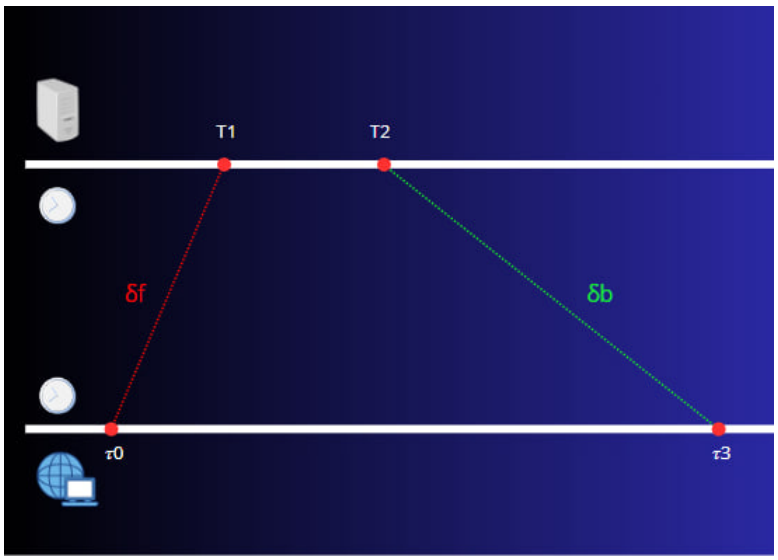

Рисунок. 1.3. — Ілюстрація асиметрії каналу зв'язку між клієнтом та сервером.

Систему (5) щодо незалежних змінних θ і p не можна розв'язати, оскільки визначник цієї системи дорівнює 0. З врахуванням того, що $\Delta = \delta_f + \delta_b$, можна переписати систему (5) у вигляді (6):

$$\begin{cases} \tau_0 + \delta_f + \tilde{\theta}_0 = T_1 \\ \tau_3 - (\Delta - \delta_f) + \tilde{\theta}_0 = T_2 \end{cases}, \tag{6}$$

Ця система еквівалентна системі:

$$\begin{cases} \delta_f + \tilde{\theta}_0 = T_1 - \tau_0 \\ \delta_f + \tilde{\theta}_0 = T_1 - \tau_3 + \Delta \end{cases}, \tag{7}$$

Визначник цієї системи щодо незалежних змінних δ_f і θ дорівнює 0, а ранг дорівнює 1. Це означає, з одного боку, що навіть у ідеальному випадку неможливо одночасно визначити зміщення часу годинника та час поширення сигналу вперед (і назад). І, крім того, легко одержати суперечливу систему з емпіричних даних. Враховуючи, що реальний параметр асиметрії визначати дуже дорого, завдання синхронізації навіть для ідеального годинника стає дуже складним у мережі з асиметричними каналами зв'язку. Очевидно, що одного етапу обміну повідомленнями навіть для детермінованих каналів зв'язку недостатньо. Нижче буде показано, що це завдання не можна вирішити в асиметричному випадку (при невідомому параметрі p)

будь-якої кількості етапів обміну інформацією між клієнтом і сервером.

1.4 Афінна модель і параметри синхронізації годинників

У моделі реальних годинників потрібно враховувати факт, що будь-який годинник практично завжди змінює швидкість свого ходу. Різниця тактових частот одного годинника за певний інтервал часу називається дрейфом тактової частоти. З цієї причини системи (4) і (5) припускають, що значення змінюватиметься з часом. І системи (4) і (5) можна розглядати як моделі "миттєвого знімка" стану синхронізації.

Для уточнення моделі годинника та більш ефективного прогнозування стану синхронізації в роботі [8] була введена афінна модель годинника. Позначимо час фіксованого ідеального опорного годинника через t (як реального представника такого опорного годинника можна розглядати атомний цезієвий годинник, вартість якого дуже висока).

В афінній моделі відображення часу τ_j на годиннику j-го клієнта у мережі в момент часу t позначається $\tau_j(t)$, і задовольняє рівнянню:

$$\tau_j(t) = \sigma_j t + \theta_j \tag{8}$$

де
θ_j - зміщення часу j-го клієнта щодо ідеального годинника в мережі,
σ_j - відносини швидкостей ходу годинника j-го клієнта щодо ідеального годинника в мережі.
Таким чином, афінний годинник відносно ідеального годинника в мережі може бути представлений парою параметрів (θ_j, σ_j).

Можна виразити залежність показань годинника T сервера від показань годинника τ клієнта:

$$T(\tau) = \frac{\sigma_0}{\sigma_1}\tau + \theta_0 - \frac{\sigma_0}{\sigma_1}\theta_1, \tag{9}$$

де (σ_0, θ_0) - афінні параметри годинника сервера,
(σ_1, θ_1) - афінні параметри годинника клієнта.

Виходячи з цього, показання годинника клієнта можна виразити через показання годинника сервера:

$$\tau(T) = \frac{1}{\sigma}T - \frac{\theta}{\sigma} = \sigma_s T + \theta_s \tag{10}$$

де (σ_s, θ_s) - афінні параметри залежності годинника клієнта від годинника сервера.

Систему (5) щодо змінних $\tilde{\theta}_0$ і p можна переписати так:

$$\begin{cases} 1 \cdot \tilde{\theta}_0 + \Delta \cdot p = T_1 - \tau_0 \\ 1 \cdot \tilde{\theta}_0 + \Delta \cdot p = T_2 - \tau_3 + \Delta \end{cases}, \tag{11}$$

Очевидно, що афінні залежності годинника T сервера від годинника τ клієнта (і навпаки) не роблять цю систему більш визначеною, а значить, неможливо за допомогою методів синхронізації з одним етапом обміну інформацією між клієнтом і сервером, отримати одночасно невідомі $\tilde{\theta}_0$ і p, і слідом за ними параметр σ.

Параметр σ можна отримати за допомогою інших методів оцінки взаємного дрейфу годинників ([8]). Наприклад, можно показати що для афінної моделі годиннинів

$$\sigma = \lim_{i \to \infty} \frac{T_i - T_1}{\tau_{i-1} - \tau_0}. \tag{12}$$

Але ці рішення не дають нам можливість визначити синхронізовані пари τ_i та T_i. Це дозволяє зробити висновок про непристосованість однокрокових алгоритмів синхронізації годинників для роботи в мережах з невідомою асиметрією. І, навпаки, якщо нам наперед точно відома асиметрія каналу зв'язку, ми можемо вирішити завдання точної синхронізації годинника в детермінованому каналі за один крок.

З урахуванням того, що ми використовуємо афінну модель для представлення кожного вузла в мережі, можна переписати систему (5) у вигляді:

$$\begin{cases} \sigma\tau_0 + \sigma p\Delta + \theta = T_1 \\ \sigma\tau_3 - \sigma(1-p)\Delta + \theta = T_2 \end{cases}, \tag{13}$$

Слід наголосити, що θ_0 в системі (5) є величина змінна (вона змінюється кожному етапі обміну між клієнтом і сервером), тоді як в системі (13) θ є величина постійна. Визначник цієї системи :

$$\begin{vmatrix} \sigma\Delta & 1 \\ \sigma\Delta & 1 \end{vmatrix} = 0 \tag{14}$$

Легко довести, що права частина (13) є вектор з рівними компонентами. Отже, система (13) є узгодженою і має безліч рішеннь. У наступному розділі розглянемо можливість синхронізації в мережі, коли проводиться більше одного етапу обміну інформацією між клієнтом і сервером.

1.5 Фундаментальні обмеження багатоетапної синхронізації годинників в мережах

Як показано в роботі [8], завдання синхронізації можна розділити на 3 типи, що являють собою своєрідну ієрархію:

1. Упорядкування подій (Ordering of Events)

Мета: Створити правильну хронологію подій у мережі.

Точний час не потрібний: Необхідно лише визначити порядок подій, а не точних часових міток. Конфліктні часові мітки узгоджуються алгоритмом вирішення конфліктів. Це найменш строга форма синхронізації: Достатньо для багатьох програм.

Приклади:

Фінансові транзакції з блокуваннями в базах даних: Важливо знати чи визначити, які транзакції сталися раніше, щоб уникнути конфліктів.

Деякі програми моніторингу: Наприклад, моніторинг систем безпеки, де важливо знати послідовність подій, але не їх точний час.

Технологія: Віртуальний годинник: Використання логічних позначок для впорядкування подій. Ця ідея була вперше досліджена у роботах з розподілених систем.

2. Відносна синхронізація (Relative Synchronization)

Мета: Оцінити відносну розбіжність між годинниками у мережі.

Переклад часових міток: Використання інформації про дрейф для перекладу часових міток між різними годинниками без їх корекції. Уникнення небажаних залежностей: Годинники у вузлах не корегуються, що запобігає проблемам із синхронізацією.

Приклади:

Мультимедійні системи: Синхронізація аудіо та відео потоків, де важливе узгодження часових міток, але не абсолютний час.

Розподілені обчислення: Узгодження часових міток між серверами для координації завдань.

Технологія: Відносний годинник: Використання механізмів, які відстежують та компенсують дрейф годинника між вузлами.

3. Абсолютна синхронізація (Absolute Synchronization)

Мета: Встановити абсолютний годинник на всіх вузлах у мережі, щоб досягти глобального узгодження часу.

Найсуворіша форма синхронізації: Потрібен для додатків, де критично важливий точний час.

Приклади:

Мережеві системи управління: Наприклад, системи управління повітряним рухом, де точний час критичний для координації дій.

GPS: Система глобального позиціонування потребує точної синхронізації часу для розрахунку розташування.

Наукові експерименти: Наприклад, синхронізація даних із різних телескопів під час спостереження астрономічних явищ.

Технологія: Абсолютний годинник: Використання атомних годинників, сигналів GPS для протоколів, таких як NTP (Network Time Protocol) або PTP (Precision Time Protocol), для досягнення точної синхронізації часу між вузлами.

Абсолютна синхронізація передбачає відносну синхронізацію, яка, своєю чергою, передбачає упорядкування подій у мережі. Однак якщо ми

визначимо конкретний вузол як еталонний, то відносна синхронізація може використовуватися для досягнення абсолютної синхронізації. Ідеальним сценарієм було б досягнення абсолютної синхронізації годинника, коли всі вузли мережі показують один і той же глобальний час.

Розглянемо набір параметрів однокрокової системи (13):
$\tau_0,\ \tau_3,\ T_1\ i\ T_2$ - параметри системи, які точно встановлені на першому етапі сінхронізації,
$\sigma,\ p\Delta,\ (1-p)\Delta\ i\ \theta$ - невідомі параметри системи після першого етапу сінхронізації.
Вирішення завдання синхронізації передбачає єдиність рішення системи рівнянь (13), оскільки, для узгодження показників часу парі вузлів потрібно однозначне визначення всіх параметрів, необхідні синхронізації пари вузлів. Якщо система рішень немає, чи, має безліч рішень, навіть формально в ідеальних умовах визначити справжні значення всіх параметрів не можна і синхронізація буде некоректною. У вихідній системі рівнянь (13) невідомі $\sigma,\ p\Delta,\ (1-p)\Delta\ i\ \theta$ входять нелінійно, що ускладнює аналіз існування єдиності рішення. Нелінійна параметризація (13) для багатошагової синхронізації дозволяє лінеаризувати систему шляхом подання її у матричному вигляді

$$Y = Ax, \tag{15}$$

де
Y - вектор невідомих величин
A - матриця коефіцієнтів,
x - вектор нових невідомих змінних.
В розгорнутому вигляді її можно записати як

$$\begin{pmatrix} T_1 \\ T_2 \\ T_5 \\ T_6 \\ \vdots \end{pmatrix} = \begin{pmatrix} \tau_0 & 1 & 0 & 1 \\ \tau_3 & 0 & -1 & 0 \\ \tau_4 & 1 & 0 & 1 \\ \tau_7 & 0 & -1 & 0 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \cdot \begin{pmatrix} \sigma \\ \sigma p\Delta \\ \sigma(1-p)\Delta \\ \theta \end{pmatrix} \tag{16}$$

Як видно, Y та A містять відому інформацію, засновану на відомих (одержуваних з експерименту) часових мітках (T_1,T_2,T_5,T_6 - помітки годинника сервера, τ_0,τ_3,τ_4,τ_7 - часові помітки клієнта), у той час, як x - вектор невідомих складних параметрів. Визначник матриці A системи (16) дорівнює 0.

Необхідною та достатньою умовою для єдиності цього рішення є умова повноти рангу матриці A (тобто передбачається, що її ранг має дорівнювати 4). Ця умова також є необхідною і достатньою для існування єдиності рішення $\sigma,\ p\Delta,\ (1-p)\Delta\ \ i\ \ \theta$ в (13), оскільки параметризація $\sigma,\ p\Delta,\ (1-p)\Delta\ \ i\ \ \theta \to x$ є бієктивною, за умови, що $\sigma\ \ \neq\ \ 0$.

Однак, 4 стовпець матриці A (16) являє собою різницю 2 і 3 стовпців, і, отже, A має ранг не більше 3. Таким чином, навіть при обміні нескінченної кількості пакетів між двома вузлами в мережі, оцінка всіх чотирьох параметрів $\sigma,\ p\Delta,\ (1-p)\Delta\ \ i\ \ \theta$ неможлива, з чого випливає, що абсолютна та відносна синхронізація годинника в розподіленій мережі є недосяжним завданням.

Тим не менш, приблизна відносна синхронізація та оцінка швидкостей годинника цілком здійсненна, що є більш ніж достатнім для деяких додатків. Крім того, є цілий спектр додатків реального часу, для ефективного функціонування яких достатньо лише інформації про параметр початвокої неузгодженності часових поміток $\tilde{\theta}_0$, визначеного за формулою (1). Це буде доведено у розділі 2.

РОСШИРЕНІ ПІДХОДИ ДО ЗАДАЧ СИНХРОНІЗАЦІЇ

Завдання синхронізації принципово ускладнюється при поширенні сигналів реальними каналами зв'язку, оскільки значення θ (різниця часових міток між клієнтом та сервером) неможливо однозначно визначити через асиметрію мережних затримок. Тому завдання синхронізації зводиться до пошуку максимально наближеного до нього значення. Фундаментальні обмеження не дозволяють забезпечувати хорошу якість синхронізації без наявності у вузлах мережі високоточних (атомних) годинників або обладнання з аналогічними функціями. Але за наявності апріорної інформації про тип стабільного каналу зв'язку та його мінімальної та максимальної довжини або інформації про коефіцієнт асиметрії каналу ξ, отриманої шляхом досліджень реальних каналів передачі даних, можна значно збільшити точність синхронізації. Крім того, є ряд систем, які не вимагають точної фізичної синхронізації годинників для точної часової координації дій. Тому для них є сенс переформулювати поняття синхронізації, замінивши його на поняття синхрокоординації. Тому росширені підходи до завдань синхронізації можно поділити на дві основні спрямованості:

1) досягнення максимальної наближеності до справжнього значення параметра θ,
2) зміна парадигми синхронізації на синхрокоординацію, що передбачає забезпечення максимальної точності координування дій вузлів у системі.

Як буде показано нижче, другий підхід не потребує знання реального значення θ, але потребує точного обчислення параметрів синхрокоординації.

2.1 Інтервали оцінки неузгодження часових поміток клієнта і сервера та методи їх коригування

2.1.1 Помилки синхронізації

Щоб отримати рівняння для початкової синхронізації $\tilde{\theta}_0$ потрібно вирішити систему (17) , припускаючи, що затримка сигналу від клієнта до

сервера дорівнює затримці від сервера до клієнта:

$$\begin{cases} \tau_0 + \tilde{\theta}_0 + \frac{1}{2}\Delta = T_1 \\ \tau_3 + \tilde{\theta}_0 - \frac{1}{2}\Delta = T_2 \end{cases}, \tag{17}$$

Припустимо, що ми маємо дві затримки: пряму δ_f і зворотну δ_b. Якщо ми припускаємо, що вони симетричні, то ми припускаємо, що $\delta_f = \delta_b = \frac{\Delta}{2}$, де Δ - це загальна затримка. Але в реальності ці затримки можуть бути асиметричними. Тоді ми можемо визначити коефіцієнт асиметрії $\xi_a = \frac{\delta_f}{\delta_b}$, який в лежить в межах $(0,\infty)$, а в симетричних каналах дорівнює 1.

Твердження 1. Якщо припущення про симетричність затримок виявилося хибним, помилка $|\partial\theta| = |\tilde{\theta}_0 - \theta|$ визначення поточної неузгодженості за формулою (1) обмежена нерівністю:

$$|\partial\theta| \leq \frac{1}{2}\left|\frac{\xi_a - 1}{\xi_a + 1}\right|\Delta \tag{18}$$

Доказ. Розглянемо компоненти помилки $|\partial\theta| = |\tilde{\theta}_0 - \theta|$. Якщо ми використовуємо формулу (1) для оцінки $\tilde{\theta}_0$, то ми отримуємо $\tilde{\theta}_0 = \frac{\Delta}{2} = \frac{\delta_f + \delta_b}{2}$. Але якщо затримки асиметричні, то реальне значення θ буде відрізнятися. Ми можемо визначити $\delta_f = \xi_a \cdot \delta_b = \xi_a \cdot \frac{\Delta}{\xi_a + 1}$.

Тоді максимальна помилка буде

$$|\partial\theta| = |\tilde{\theta}_0 - \theta| = \left|\frac{\Delta}{2} - \frac{\xi_a \cdot \Delta}{\xi_a + 1}\right| = \frac{1}{2}\left|\frac{\xi_a - 1}{\xi_a + 1}\right|\Delta. \tag{19}$$

Твердження 2. Верхня межа оцінки (18) помилки $\partial\theta$ є точною та фіксуючою. Тобто

$$\partial\theta = \frac{\xi_a - 1}{\xi_a + 1}\frac{\Delta}{2} \tag{20}$$

Якщо припущення про симетричність затримок виявилося хибним, але нам відомий коефіцієнт асиметрії ξ_a, скоригована помилка $\partial\theta' = \tilde{\theta}'_0 - \theta = (\tilde{\theta}_0 + \partial\theta) - \theta$ визначення поточної неузгодженості $\tilde{\theta}_0$ за формулою (1), дорівнює 0.

Доказ. Якщо нам відомий коефіцієнт ξ_a, то ми можемо скоригувати

нашу оцінку $\tilde{\theta}_0$ так, щоб врахувати асиметрію. Замість використання в системі (4) виразу $\frac{\Delta}{2}$, ми можемо використати формулу $\frac{\xi_a \cdot \Delta}{\xi_a + 1}$, яка враховує асиметрію. Тоді скоригована помилка буде:

$$|\partial\theta|' = |\tilde{\theta}_0' - \theta| = \left| \frac{\xi_a \cdot \Delta}{\xi_a + 1} - \frac{\xi_a \cdot \Delta}{\xi_a + 1} \right| = 0 \tag{21}$$

Отже, якщо нам відомий коефіцієнт асиметрії ξ_a, то скоригована помилка визначення поточної неузгодженості $\tilde{\theta}_0$ за формулою (1) буде рівна нулю, незалежно від того, чи виявилося припущення про симетричність затримок хибним. Це доводить, що знання ξ_a дозволяє нам точно визначити поточну неузгодженість мережевих годинників, навіть якщо затримки асиметричні.

2.1.2 Інтервал невизначенності

Якщо для інформаційного каналу нам відомі межі вимірювань коефіцієнта асиметрії $[\xi_{\min}, \xi_{\max}]$, то ми отримуємо інтервал невизначеності для оцінки $\tilde{\theta}_0 = T_0 - \tau_0$ поточної неузгодженості між часовими помітками клієнта та сервера:

$$\begin{aligned} [\tilde{\theta}_{\min}, \tilde{\theta}_{\max}] &= [\tilde{\theta}_0 + \partial\theta_{\min}, \tilde{\theta}_0 + \partial\theta_{\max}] = \\ &= \left[\tilde{\theta}_0 + \frac{\xi_{\min} - 1}{\xi_{\min} + 1} \frac{\Delta}{2}, \tilde{\theta}_0 + \frac{\xi_{\max} - 1}{\xi_{\max} + 1} \frac{\Delta}{2} \right] \end{aligned} \tag{22}$$

Легко показати, що $\tilde{\theta}_{\min} < \tilde{\theta}_{\max}$. Насправді розглянемо $\partial\theta$ як функцію від ξ при Δ розглядаємому як параметр. Оскільки функція $\partial\theta_\Delta(\xi) = \frac{\xi - 1}{\xi + 1} \frac{\Delta}{2}$ є монотонно зростаючою на інтервалі $[0,\infty]$, то і функція $\tilde{\theta}(\xi) = \tilde{\theta}_0 + \partial\theta_\Delta(\xi)$ також є монотонно-зростаючою функцією, тобто $\tilde{\theta}(\xi_{\min}) < \tilde{\theta}(\xi_{\max})$, оскільки $\xi_{\min} < \xi_{\max}$. І, отже, $\tilde{\theta}_{\min} < \tilde{\theta}_{\max}$.

2.1.3 Умови уточнення інтервалів параметрів синхронізації

Якщо ми розглядаємо пов'язану інформаційну мережеву систему з кількістю вузлів більшим 2 і фізична топологія, що лежить в основі побудови каналів зв'язку цієї системи, містить незалежні канали (або частини

каналів) між усіма парами вузлів цієї системи, у нас з'являється можливість уточнення інтервалів параметрів синхронізації між клієнтами та сервером, тобто зменшення інтервалів невизначеності $[\tilde{\theta}_{\min},\tilde{\theta}_{\max}]_i$ i-го клієнта з сервером. Для цього слід між клієнтами i та j встановити безпосереднє з'єднання (у сучасному інтернеті це нетривіальне завдання, але одним із варіантів є з'єднання за протоколом WebRTC), вибрати одного з цих клієнтів, наприклад i, часовим сервером і виконати процедуру синхронізації годинників між ними з визначенням інтервалу $[\tilde{\theta}_{\min},\tilde{\theta}_{\max}]_{ij}$ невизначеності для оцінки $\tilde{\theta}_{ij} = \tau_i - \tau_j$ поточної неузгодженості між часовими помітками клієнта i та клієнта j. В ідеальній узгодженій системі з трьох вузлів - сервера, клієнта i, клієнта j, має виконуватися рівність:

$$\tilde{\theta}_{0,i} + \tilde{\theta}_{i,j} = \tilde{\theta}_{0,j} \tag{23}$$

де $\tilde{\theta}_{0,i}$, $\tilde{\theta}_{i,j}$, $\tilde{\theta}_{0,j}$ - це оцінки поточної неузгодженості між часовими помітками сервера з клієнтом i, клієнта i з клієнтом j, та сервера з клієнтом j відповідно.

2.1.4 Методи узгодження інтервалів

Однак ми маємо замість точних оцінок інтервали і тому маємо виконати узгодження інтервалів. Це можна зробити багатьма способами, адекватність яких потребує подальших досліджень. У нашій роботі ми коротко окреслимо два способи.

Узгодження інтервалів за допомогою використання інтервальної арифметики

Першим кроком у цьому підході буде отримання суми інтервалів. Розглянемо інтервали $[a,b]$ та $[c,d]$. Тоді їхньою сумою згідно [4] буде $[a,b] + [c,d] = [a + c,b + d]$. А потім ми уточнюємо інтервал $[e,f]$ з отриманим сумарним інтервалом $[a + c,b + d]$ за формулою:

$$[e',f'] = [\max(a + c,e), \min(b + d,f)] \tag{24}$$

Узгодження інтервалів за допомогою методів нечітких множин

У цьому підході на інтервалах визначаються нечіткі множини і, крім самих інтервалів, слід задати функції приналежності до нечітких множин (наприклад, трикутну функцію приналежності), функції узгодження інтервалів (наприклад, мінімум) та методу нормалізації нечітких множин.

2.2 Точна координація розподілених додатків реального часу

Для багатьох розподілених систем реального часу [7] синхронізація необхідна не стільки для узгодження часових міток, скільки для того, щоб вузли в цій системі були здатні координувати свої дії з гранично високою точністю. Незважаючи на те, що досягнення точної синхронізації може бути неможливим без знання коефіцієнта асиметрії ξ_a, точна координація дій є цілком досяжною метою, що буде доведено нижче. З цієї причини доцільно для таких систем замінити поняття синхронізації на поняття синхрокоординації майбутніх подій в частинах розподілених додатків для детермінованих каналів зв'язку.

2.2.1 Доказ концепції синхрокоординації

Твердження.
Визначення початкової оцінки неузгодженості часових міток θ_0 за формулою (1) необхідно і достатньо для здійснення синхрокоординації клієнта та сервера без знання коефіцієнту асиметрії ξ за умовою детермінованності каналу зв'язку.
Доказ:
Розглянемо двохфазний етап обміну часовими мітками між клієнтом та сервером для отримання початкової оцінки неузгодженості θ_0 (Рис. 2.1).

Вихідні часові мітки фіксуються наступним чином: τ_0 - час надсилання запиту клієнтом.
T_1 - час отримання запиту сервером.
T_2 - час надсилання відповіді сервером.
τ_3 - час отримання відповіді клієнтом.
T_5 - запланований сервером час отримання повідомлення від клієнта.

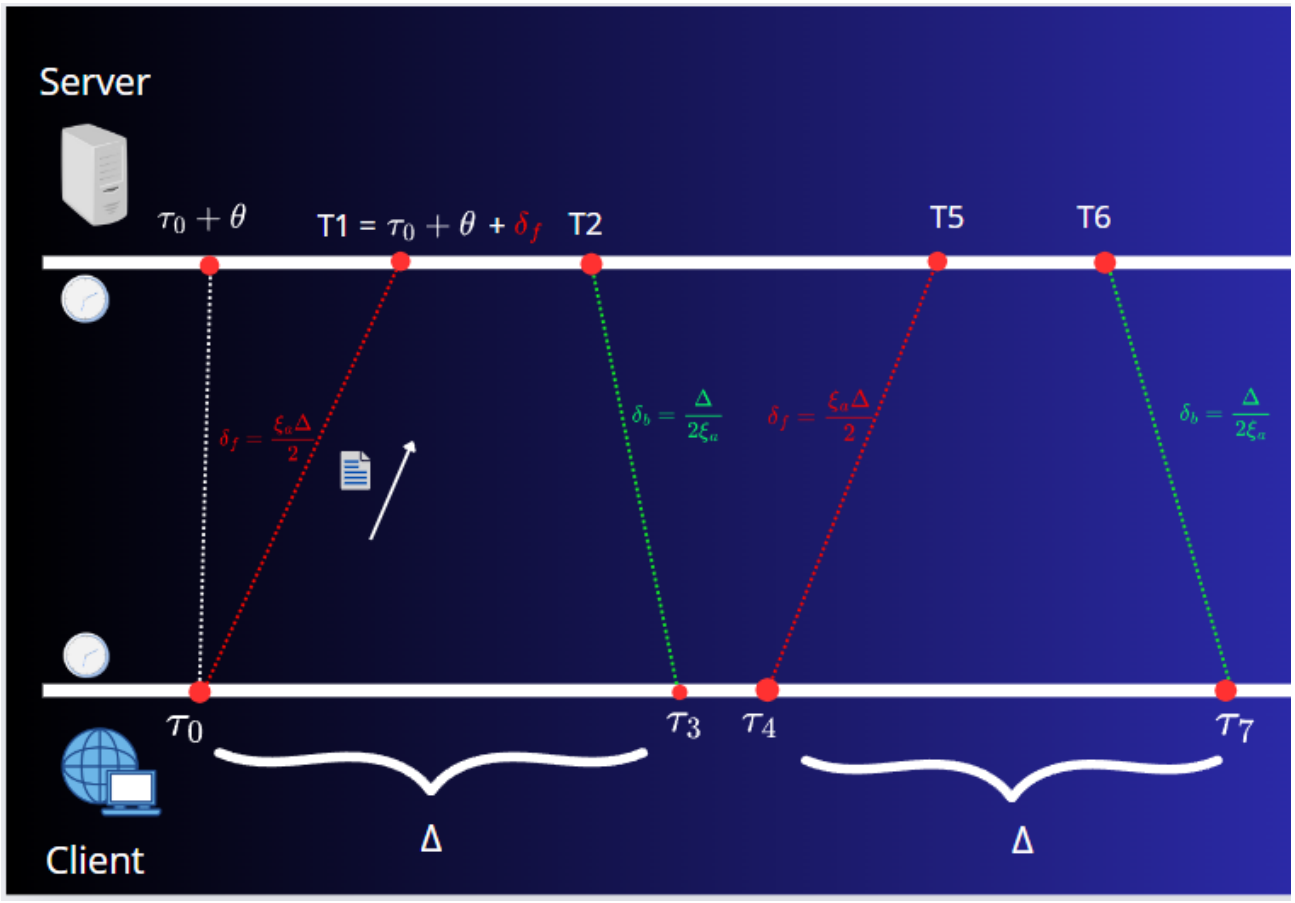

Рисунок. 2.1. — Двохфазний етап обміну часовими мітками між клієнтом та сервером для отримання початкової оцінки неузгодженості θ_0.

Час кругової затримки Δ визначається на першому етапі обміну повідомленнями як $\tau_3 - \tau_0$. У разі симетричного каналу (коефіцієнт асиметрії $\xi = 1$) час поширення сигналу туди і назад визначається як $\frac{1}{2}\Delta$. Для асиметричного каналу ($\xi \neq 1$) час поширення сигналу від клієнта до сервера можна позначити як $\xi\frac{1}{2}\Delta$, а від сервера до клієнта - як $\frac{1}{\xi}\frac{1}{2}\Delta$. Початкова оцінка неузгодженості $\tilde{\theta}_0$ визначається за формулою (1).

З рівняння (1) введемо поняття прямої координаційної різниці γ_f:

$$\gamma_f = T_1 - \tau_0 = \tilde{\theta}_0 + \xi\frac{1}{2}\Delta \tag{25}$$

Для синхрокоординації на другому етапі, для отримання на сервері повідомлення від клієнта рівно о T_5, використовуючи отриману на першому етапі пряму координаційну різницю γ_f отримуємо значення часу відправлення

повідомлення від клієнта τ_4 як: $\tau_4 = T_5 - \gamma_f$.

Оскільки координаційний параметр γ_f, по-перше, визначається точно по годинниках клієнта і сервера, і, по-друге, не змінюється в ідеальному каналі зв'язку, його можна використовувати для каузальної синхрокоординації дій частин розподіленого докладання. Можна розширити початковий етап синхрокоординації і аналогічно отримати параметр γ_b для зворотної синхрокоординації, тим самим створюючи можливість вирішувати завдання кругової синхрокоординації. Однак слід зазначити, що синхрокоординація із зворотним зв'язком, враховуючи запізнення в поширенні сигналів, може виконуватися з частотою, не більшою за $F = 1/\Delta$, де Δ - кругове запізнення в системі клієнт-канал-сервер.

Приклад синхрокоординації: якщо сервер за власним годинником призначає для клієнтів розклад рандеву, то після однокрокової процедури обчислення параметрів синхрокоординації клієнти обчислюють точний час надсилання повідомлень за власними годинниками, тим самим виконуючи неявно скоординований обмін повідомленнями, але тільки після отримання клієнтами розкладу рандеву, що може відбутися для узгодженого розкладу ініціації повідомлень в загальному випадку не раніше, ніж через час Δ після обчислення параметрів синхрокоординації останнім із множини клієнтів.

Важливою умовою коректності синхрокоординації станів у таких системах є їхня інформаційна замкненість на період Δ обміну повідомлень. Між етапами синхрокоординації компоненти системи можуть отримувати інформацію із зовнішнього світу, виконуючи під час етапів синхрокоординації кроки щодо обміну отриманої інформації та узгодження загального стану розподіленої системи.

Для каналів зв'язку з недетермінованими затримками потрібна розробка ймовірнісних протоколів кратного узгодження дій у часі.

РЕАЛІЗАЦІЯ РОСШИРЕНИХ ПІДХОДІВ ДО ЗАДАЧ СИНХРОНІЗАЦІЇ

3.1 Дослідження характеристик каналів зв'язку для задач синхронізації

Враховуючи фундаментальні обмеження та припущення, викладені раніше, для максимального наближення до реальної різниці у часі між вузлами мережі необхідні експерименти та засоби виявлення факторів та особливостей, що впливають на якість обміну часовими повідомленнями між вузлами. Для досягнення цієї мети створено систему моніторингу основних параметрів синхронізації, таких як значення RTT (час розповсюдження сигналу туди та назад) та θ (оцінка різниці часових міток між клієнтом та сервером).

Метою дослідження в цієї частині роботи було виявлення залежностей значень RTT та θ від характеристик з'єднання, таких як тип каналу зв'язку, пропускна здатність, стабільність тощо. Крім того, було досліджено вплив інтервалів між запитами синхронізації та навантаження на мережу на значення RTT та θ. Необхідно було також досліджити кореляцію між значеннями RTT, θ і часом доби, коли відбувається обмін повідомленнями, з метою врахування періодів високої та низької стабільності мережі. Важливим аспектом роботи був аналіз статистичних характеристик отриманих наборів значень RTT та θ, таких як мода , медіана, а також відстеження тенденцій зміни цих параметрів у часі. Треба було також контролювати вплив відстані між вузлами мережі, топології мережі та можливі зміни маршрутів передачі на асиметрію затримок.

У більшості систем неможливо напряму застосовувати вищенаведені методи синхронізації, оскільки реальні системи працюють в недетермінованих каналах зв'язку. В моїй роботі розглянуто моделі ймовірнісного розподілу часу затримки, сумісного ймовірнісного розподілу часу затримки і параметру синхронізації. Після аналізу гістограм розподілення контрольованих величин було обрано критерій та модель оптимізації так, щоб, по-перше, підняти вирогідність обчислювання стабильного та монотонного параметру синхронізації θ і, по-друге, для розробки моделів протоколів синхрокоординації, що виконують свое завдання із заданою ймовірністю при повторенні

запитів. Комплексне завдання пошуку максимально наближеного до точного значення параметра неузгодженості зводиться до пошуку найстабільніших значень параметрів rtt та оцінки θ. Іншими словами, завдання зводиться до пошуку максимально стабільних станів системи, що складається з активних вузлів та каналів зв'язку, а значить - наближитися до системи з детермінованими каналами зв'язку, що дозволяє використовувати в протоколах методи вирішення задач синхронізації та синхрокоординації. Цей підхід ділить завдання на 2 фази:

1) фаза оптимізації.
2) фаза розв'язання задачі синхронізації і/або синхрокоординації

З метою вирішення цього завдання було побудовано додаток та проведено дослідження розподілу параметрів rtt и θ.

3.2 Програма моніторингу для аналізу параметрів синхронізації в мережі

Рисунок. 3.1. — Глобальне розташування вузлів тестування: локальний хост в Одесі та сервер хостингу replit.com в США (отримані за допомогою програми Open Visual Trace Route 2.1.0.)

Програма була створена як клієнт-серверний додаток, де сервера були розташовані на хостінгах replit.com та glitch.com, що забезпечує інформа-

Рисунок. 3.2. — Глобальне розташування вузлів тестування: локальний хост в Одесі та сервер хостингу в Одесі та сервер хостингу glitch.com в США (отримані за допомогою програми Open Visual Trace Route 2.1.0.)

тивну інфраструктуру для розробки та тестування веб-додатків. Сервера replit.com та glitch.com росташовані за адресами US, San Francisco та US, Ashburn, Washington відповідно, що дозволяє проводити тестування в умовах реальних географічних затримок, що є критично важливим для аналізу параметрів синхронізації в глобальних комп'ютерних мережах (Рис. 3.1), (Рис. 3.2). Цей програмний продукт було створено з огляду на вимоги сучасного інтернет-простору, де швидкість та ефективність взаємодії відіграють найважливішу роль. Цей продукт має універсальну доступність завдяки можливості його запуску з будь-якого браузера на будь-якому пристрої, підключеному до Інтернету. Обчислювальними пристроями для клієнтської частини може бути як персональний компьютер, так і мобільні пристрої.

Програма дозволяє візуально оцінити розподіл параметрів RTT та θ для різних інтервалів надсилання запитів (Рис. 3.3). У програмі були досліджені протоколи HTTP та WebSocket для визначення найбільш стабільного зв'язку між клієнтом та сервером. Після практичних мережевих експеримен-

тів було обрано протокол WebSocket, як найбільш швидкий і стабільний для поставленої задачі і той, що дозволяє реалізувати двонаправлене з'єднання для обміну даними у реальному часі.

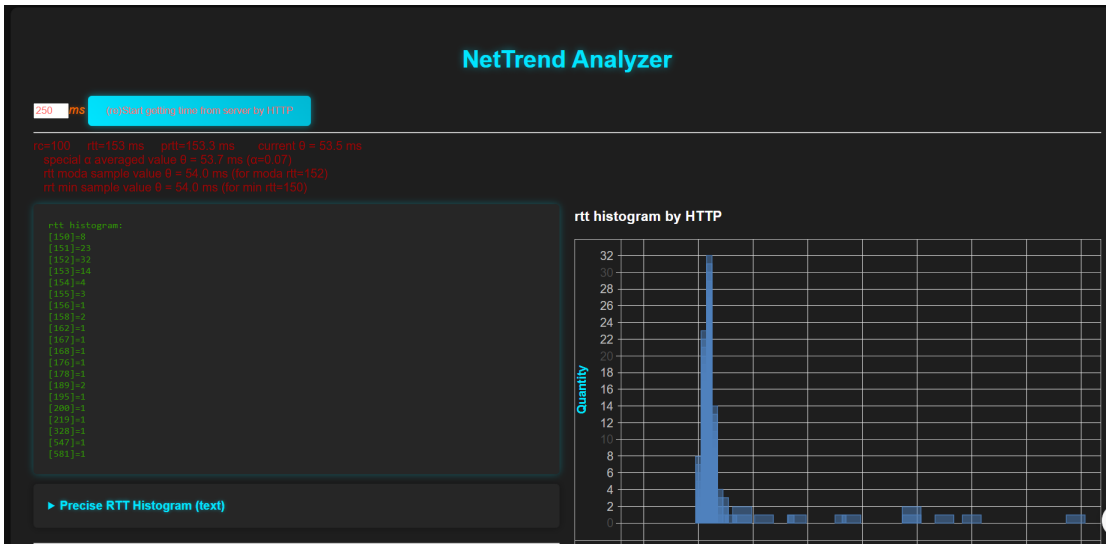

Рисунок. 3.3. — Інтерфейс моніторінгового додатку.

3.2.1 Архітектура програми

Архітектура програми складається з клієнтської та серверної частин.

Клієнтська частина

Інтерфейс користувача, створений з використанням HTML, CSS та JavaScript. Візуалізація здійснена даних за допомогою vis.js та розробленої в роботі функціональної гістограми, що дозволяє зручно контролювати сумісний розподіл двох важливих для роботи параметрів.

Логіка інтерфейсу включає: 1. Аналіз параметрів мережі, проведений з використанням протоколу HTTP. В рамках цього аналізу відображаються текстові параметри RTT та θ, кожен з яких супроводжується тимчасовим значенням та номером запиту. Щоб забезпечити точне відстеження та зіставлення даних із конкретними запитами параметри було росподілено за номерами відповідних запитів. Це дозволяє виявити закономірності та аномалії в мережевій поведінці (Рис. 3.4). Візуалізація параметра RTT здійснюється за допомогою гістограми, на якій розподілені значення часу

RTT, що дозволяє наочно оцінити їхню варіативність і стабільність (Рис. 3.5).

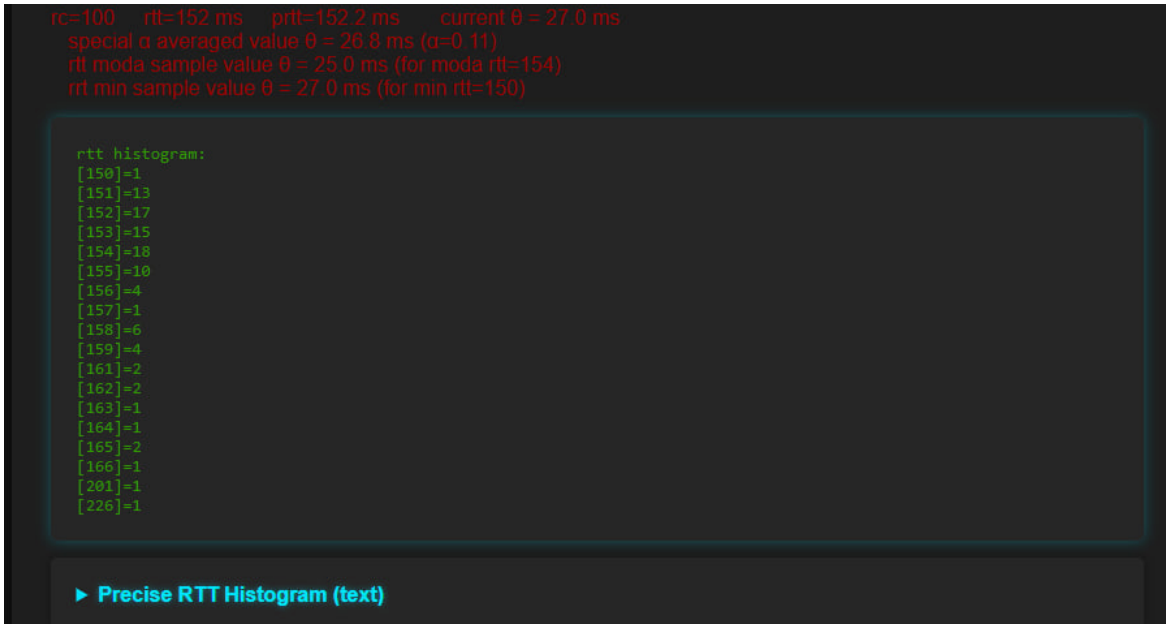

Рисунок. 3.4. — Дані параметрів RTT і θ для аналізу за допомогою HTTP з'єднання

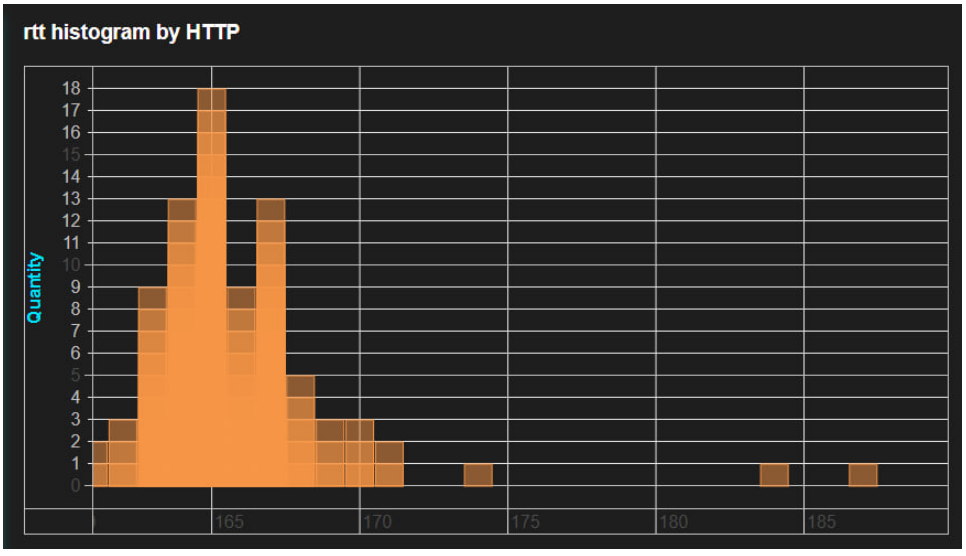

Рисунок. 3.5. — Візуалізація розподілу параметра RTT за допомогою HTTP з'єднання

2. Аналіз параметрів мережі, проведений за допомогою протоколу WebSocket. Цей аналіз аналогічний першому та включає текстове відображення параметрів мережі RTT та θ з тимчасовими значеннями та номерами запитів (Рис. 3.6), а також візуалізацію RTT за допомогою гістограми (Рис. 3.7).

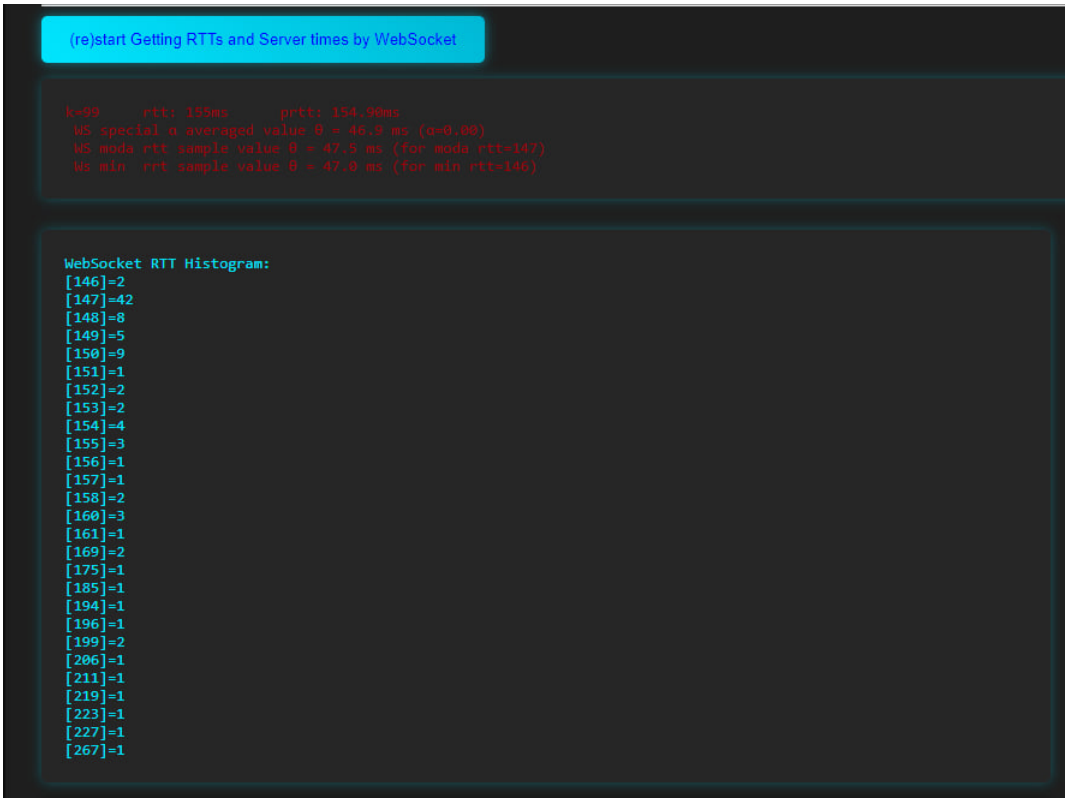

Рисунок. 3.6. — Дані параметрів RTT і θ для аналізу за допомогою WebSocket з'єднання

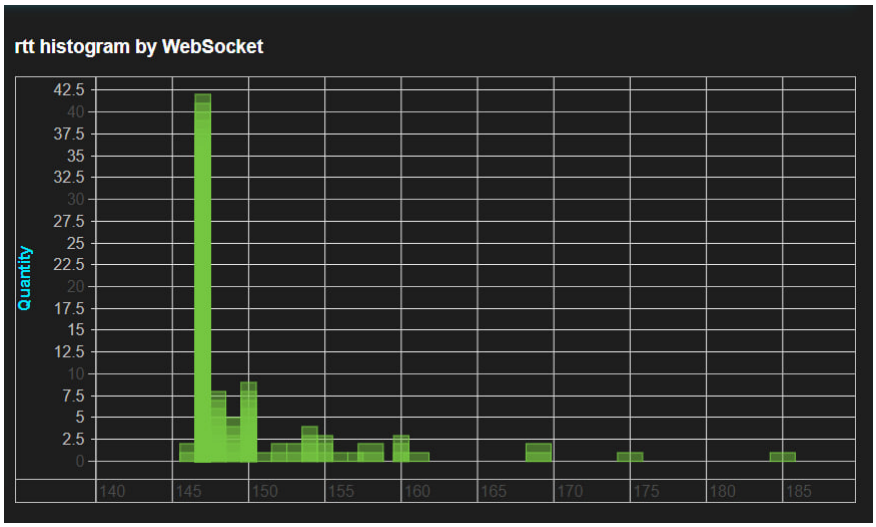

Рисунок. 3.7. Візуалізація розподілу параметра RTT за допомогою WebSocket з'єднання

Реалізація програми з використанням кількох протоколів дозволило виявити їх особливості, забезпечивши можливість вибору оптимального варіанта для поставленого завдання.

3. Одним із ключових нововведень у клієнтській частині додатку стала побудова гістограми спільного розподілу Round-Trip Time (rtt) та

параметра θ (Рис. 3.8). Параметр θ обчислюється на основі часу надсилання та отримання повідомлень між клієнтом та сервером, а також значення RTT.

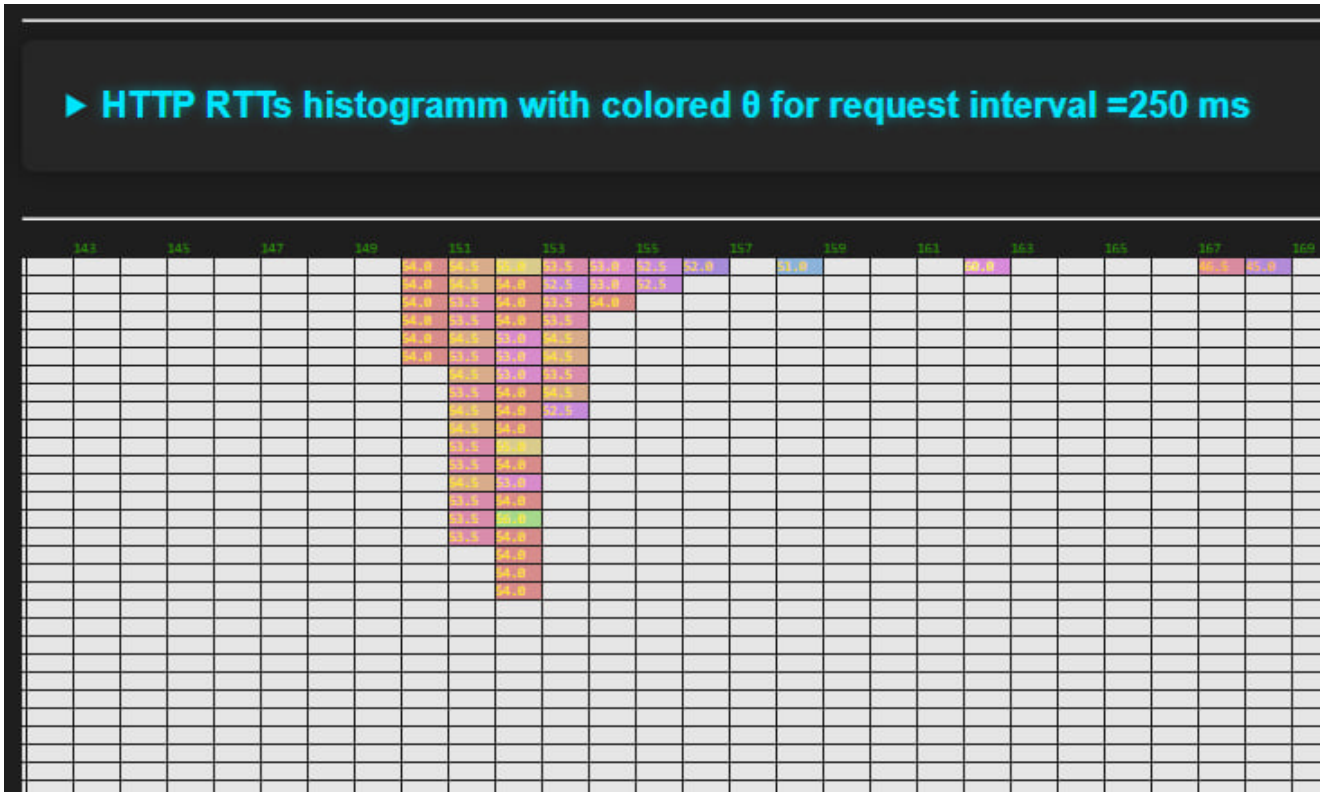

Рисунок. 3.8. — Суміщена (функціональна) гістограма розподілу RTT і θ

Таким чином, інтерфейс програми надає комплексний інструментарій для глибокого аналізу та візуалізації мережевих параметрів, забезпечуючи користувачу всі необхідні засоби для оптимізації синхронізації годинника в комп'ютерних мережах.

Серверна частина

Сервер Node.js, який використовує Express.js для зручної обробки стандартних HTTP-запитів та побудові серверної частини протоколу обміну інформації між клієнтами та сервером. В даному сервері також зручно використовувати реалізацію протокола WebSocket за допомогою бібліотеки ws. Обробка WebSocket-з'єднань та обмін даними з клієнтами виконується за допомогою стандартного API, на базі якого будуються обробники запитів, що імплементують серверну частину прикладного протоколу.

На основі аналізу розподілів для різних інтервалів запиту можна вибрати оптимальний інтервал надсилання запитів для досягнення найкращої продуктивності (Рис. 3.9).

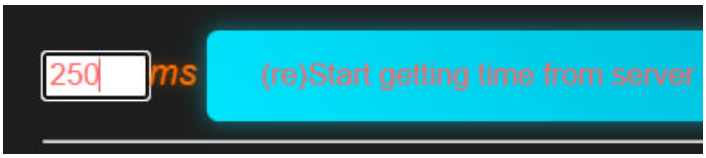

Рисунок. 3.9. — Інтерфейс для встановлення інтервалу запитів до сервера.

Були також перевірені спеціальні алгоритми згладжування та усереднення значень для більш точної оцінки мережевих параметрів. Таким чином, програма надає інструменти для моніторингу та візуалізації параметрів мережі, а також дозволяє оптимізувати ці параметри шляхом вибору відповідних інтервалів надсилання запитів та застосування спеціальних алгоритмів обробки даних.

У результаті було визначено, що з усього потоку даних потрібно вибирати саме ті, які максимально близько знаходяться до мінімуму аргументу розподілу (min rtt), і максимально стабільне значення емпіричних оцінок θ перебуває в першої моді розподілу rtt (найчастіше стабільні значення θ зустрічаються біля мінімального значення rtt в першій моді, що було неодноразово підтверджено в результаті тестування програми на різних типах з'єднання та в різні періоди часу (Рис. 3.10), (Рис. 3.11), (Рис. 3.12), (Рис. 3.13).

На гістограмі горизонтальної осі розташовані значення RTT (у мілісекундах), а комірки гістограми містять значення параметра θ (у мілісекундах). Комірки одного кольору відповідають однаковим значенням θ. Цей підхід до візуалізації був розроблений для інтуїтивного сприйняття користувачем, що дозволяє легко і швидко проводити приблизний аналіз даних, що відображаються.

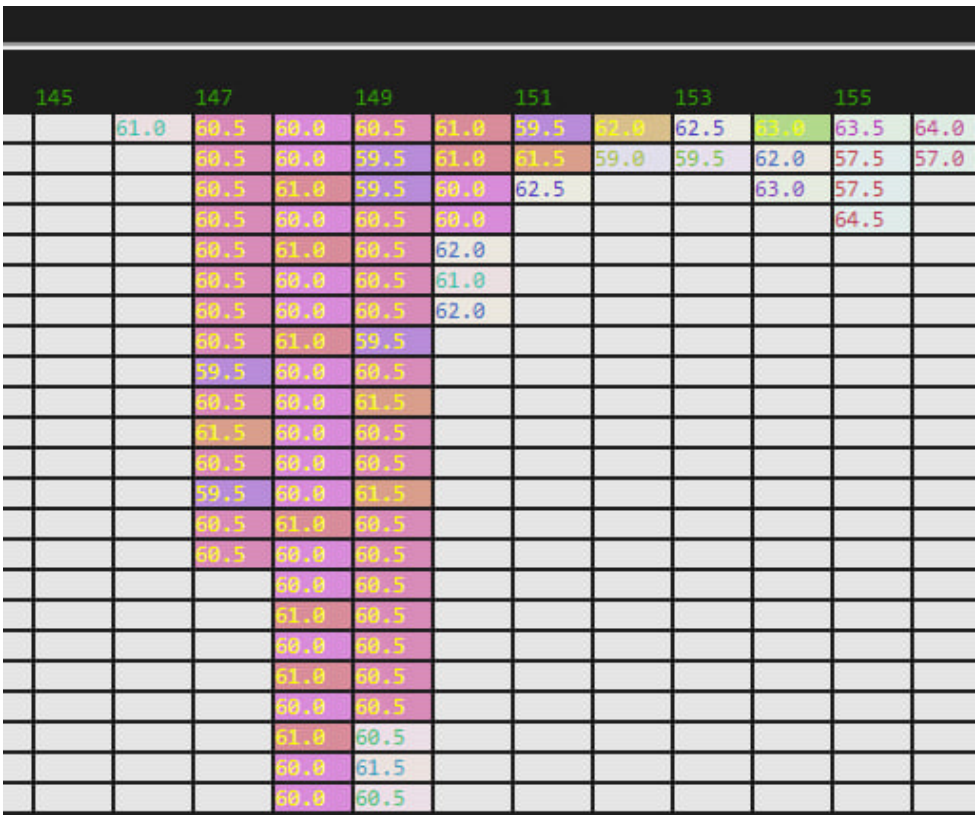

Рисунок. 3.10. — Функціональна гістограма сумісного росподілу Rtt і θ. Wifi-мережа, тестування на пк (ос Windows 10).

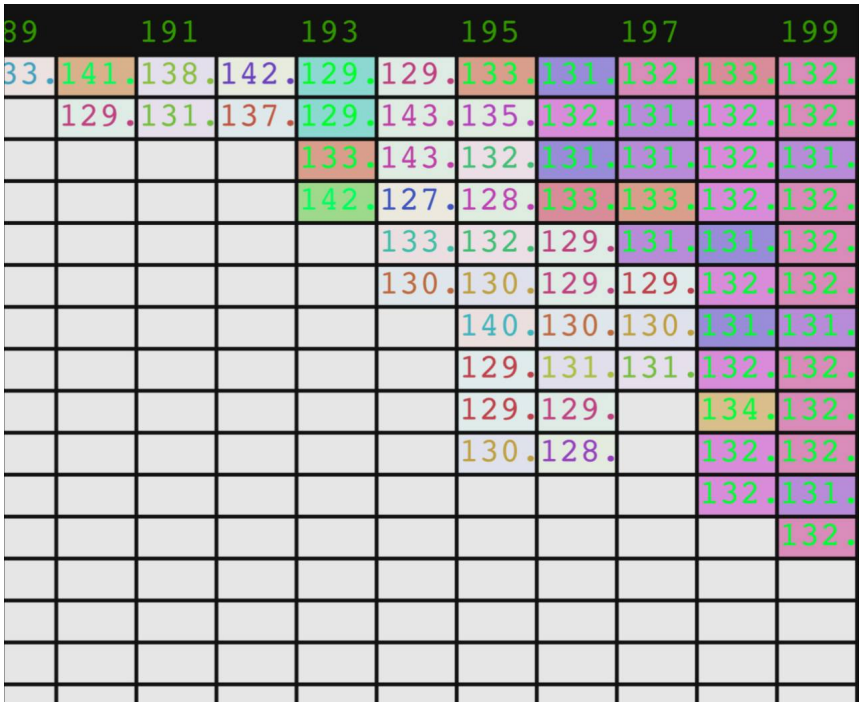

Рисунок. 3.11. — Функціональна гістограма сумісного росподілу Rtt і θ. Мобільний зв'язок, тестування на мобільному пристрої (IOS).

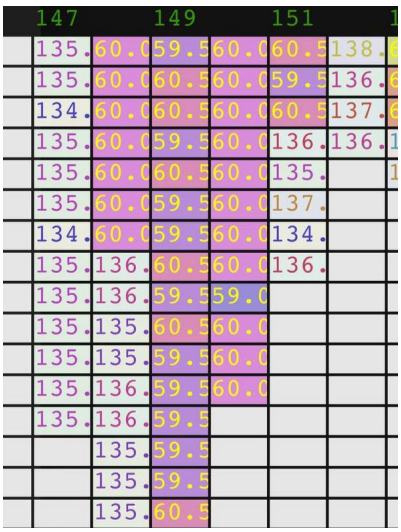

Рисунок. 3.12. — Функціональна гістограма сумісного росподілу Rtt і θ. Wifi мережа, тестування на мобільному пристрої (IOS).

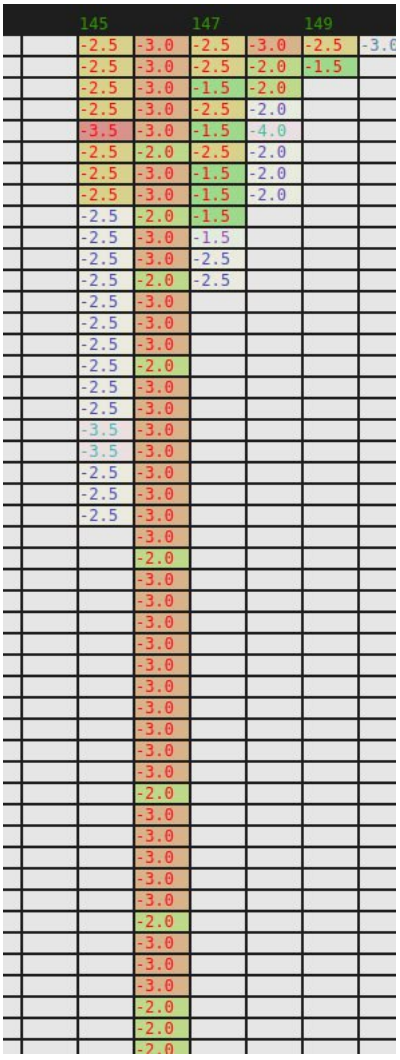

Рисунок. 3.13. — Функціональна гістограма сумісного росподілу Rtt і θ. Дротова мережа, тестування на пк (ос Linux).

3.2.2 Оптимізація параметрів синхронізації реальних мереж

Дослідження і вибір критерія оптимізації

Для пошуку максимально стабільного значення параметра θ необхідно було провести оптимізацію параметрів розподілів RTT і θ. Суть обраного критерія оптимізації полягає у пошуку двох стабільних значень: координати першої моди на осі RTT та моди розподілу θ всередині множини значень θ для цієї моди RTT. На гістограмі видно, що координата першої моди на осі RTT дорівнює 148 мілісекунд, а мода розподілу θ дорівнює 102.5 мілісекунд. (Рис. 3.14). В додатку було побудовано суміщену (функціональну) гістограму розподілу RTT і θ для детального дослідження структури поточного розподілу θ в гістограмі RTT.

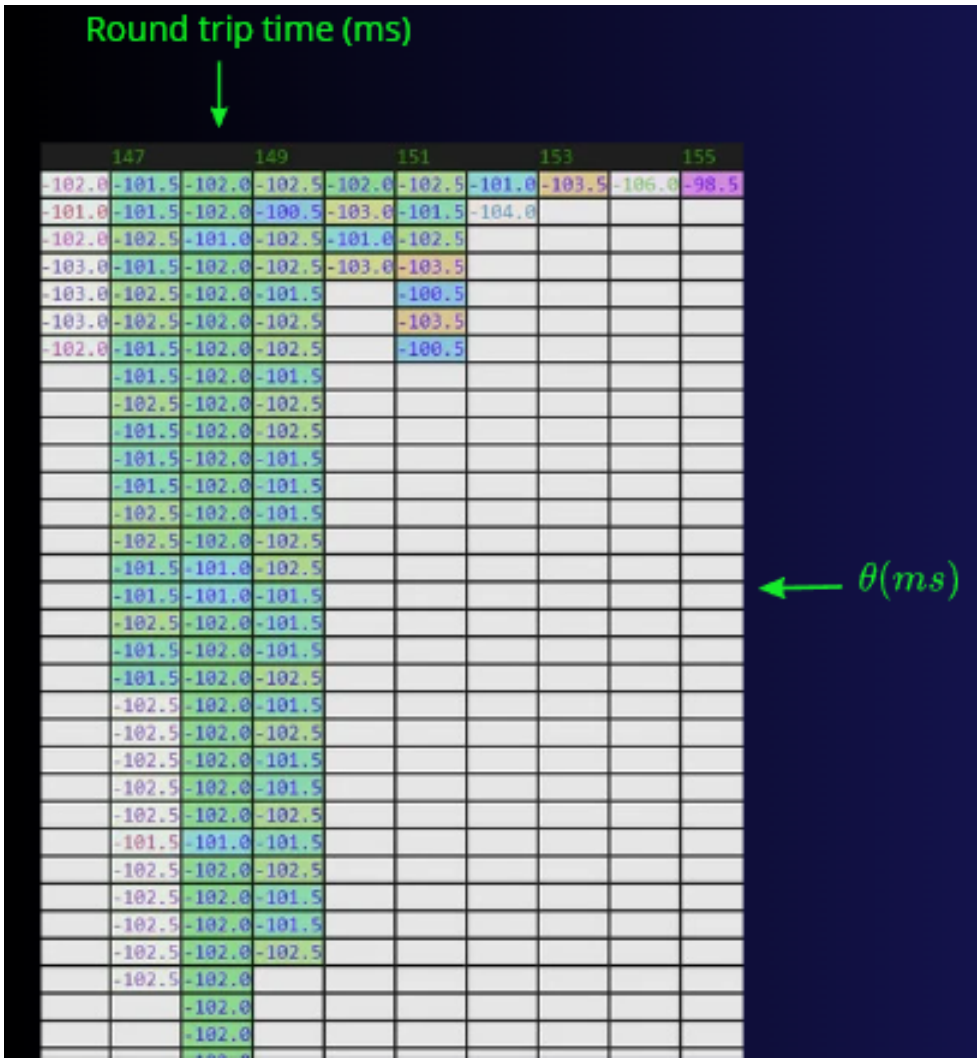

Рисунок. 3.14. — Функціональна гістограма сумісного росподілу rtt і θ.

Експерименти на реальних мережах і аналіз поточних функціональних гістограм демонструють, що стабільна характеристика θ завжди знаходиться біля початку розподілу величини RTT (rtt_{min}). В процесі досліджень необхідно було визначити найкращій критерій оптимізації для зазначеного сумісного розподілу (rtt,θ). У роботі розглядалося кілька варіантів критеріїв: мінімізація RTT, стабілізація середнього значення θ для різних типів згладжування поточних даних, максимізація моди RTT, максимізація моди θ, рівень хвостів розподілу та інші.

На сьогоднішній день в бібліотеках синхронізації існують засоби, що дозволяють провести згладжування та фільтрацію зашумлених даних (без оптимізації параметрів роботи протоколу в мережі). Однак у цих засобах вибір критеріїв згладжування та фільтрації здійснюється емпірично, за принципом простоти і практичності реалізації, без проведення систематичного і глибокого аналізу. Наприклад, у бібліотеці timesync він заснований на усередненні значень після фільтрації найменш поширених значень за допомогою обчислення медіани поточної виборки даних та adhoc крітерію відкидання "зашумлених"значень. Однак такий метод призводит до постійної флуктуації значення θ, що робить оцінку немонотонною, та не дозволяє ефективно обчислити відносний дрейф σ годинника клієнта.

З іншого боку, при стані каналу зв'язку з умовами, близькими до детермінованих, цілком можливо передбачити наступне значення параметра RTT, а значить, і точне значення θ, якщо ми отримаємо інформацію про асиметрію каналу і компенсуємо систематичну похибку. Тому для більш точного пошуку значення θ необхідно уникати банального усереднення отриманих стохастичних значень, що робить вирішення задачі менш стабільним, а натомість сфокусуватися на пошуку найбільш статистично значимих стабільних характеристик параметрів синхронізації, що в нашому випадку є перша мода RTT і для неї внутрішня мода θ. Найкращий крітерій оптимізації полягає в максимізації значень розподілу для ціх взаємопов'язаних мод. Треба підкреслити, що вибраний крітерій оптимізації є двокроковим умовним критерієм: на першому кроці оптимізується значенния першої моди розоділу RTT в залежності від інтервалу запитів, і тільки на другому обчислюється мода θ для знайденого оптимального значення моди RTT. Перевагою даного підходу є автоматичне скорочення "хвостів"розподілу

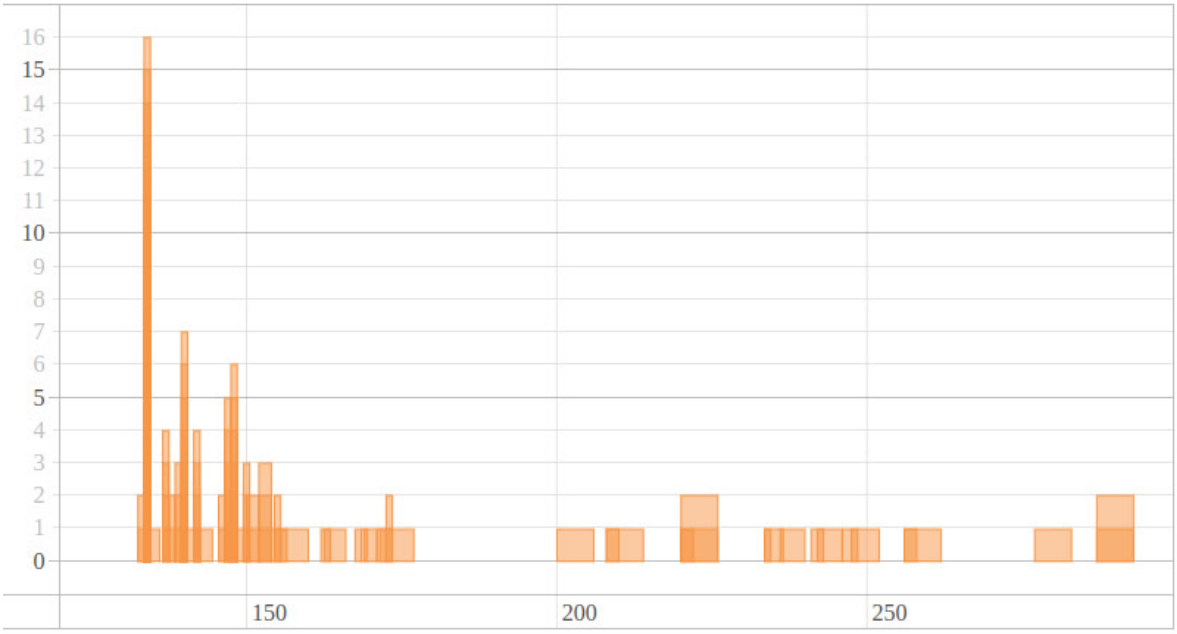

Рисунок. 3.15. — Неоптимізована гістограма розподілу затримок (протокол WebSocket, інтервал запитів – 100ms)

RTT, зменшення значення паразитних мод, збільшення кількості інформації для визначення стабільного θ.

Важливість оптимізації розподілу кругових затримок демонструють рис. 3.15 та рис. 3.16. На них представлені гістограми розподілу затримок для двох різних інтервалів запитів для одного каналу зв'язку з сервером. У той час як на рис. 3.15 ми бачимо низьку першу моду (вона сформована 16 пакетами зі 100 відправлених) і довжин хвіст розподілу затримок, що перевищує в 2 рази час стандартної затримки для цього каналу, на рис. 3.16 ми бачимо більше 60 пакетів з 100, що формують першу моду розподілу кругової затримці, близькій до мінімальної затримці на цьому каналі. І в наборі даних, що відповідають першій моді гістограми (рис. 3.16), виявляється і мода розподілу параметра синхронізації (див. також рис. 3.10, 3.11, 3.12, 3.13, 3.14).

Алгоритм оптимізації моди розподілу RTT

З метою пошуку стабільних значень параметра θ для оптимізації моди розподілу RTT, була розроблена модернізована версія алгоритму Фібоначчі. Це рішення обґрунтоване унімодальністю першого максимума гістограми

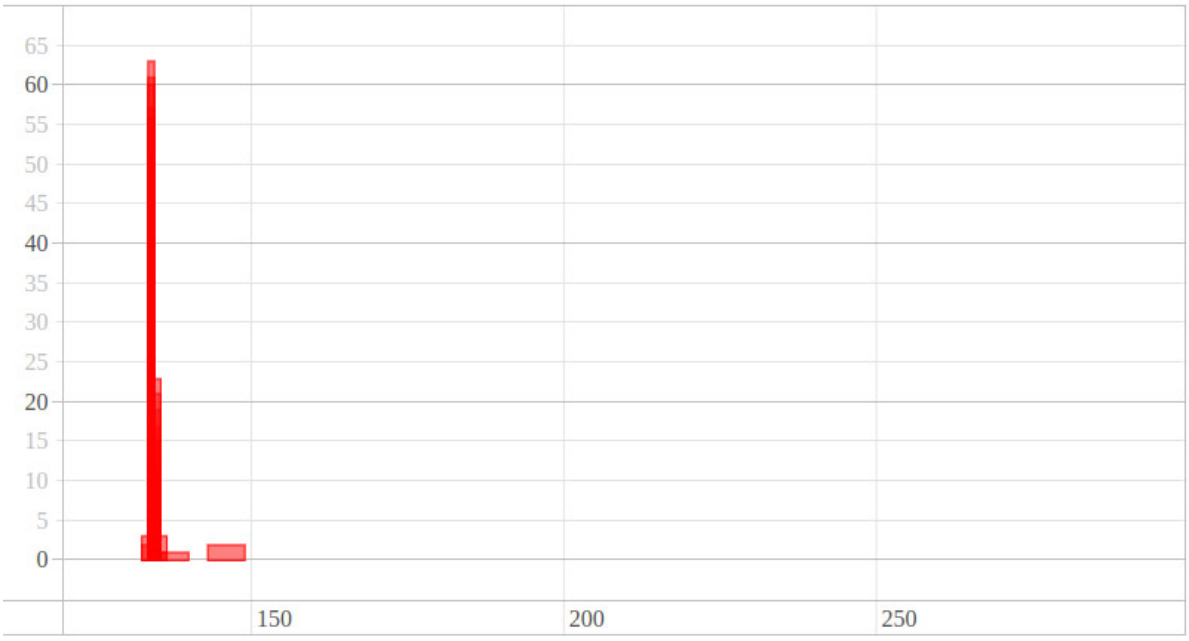

Рисунок. 3.16. — Гістограма розподілу затримок, близька до оптимальної (протокол WebSocket, інтервал запитів – 350ms)

розподілу RTT, та необхідністю виключення аномальних та нестабільних значень параметра неузгодженості з вибірки. Аргумент методу оптимізації - це часовий інтервал ι генерації пакетів запиту інформації. Такий підхід дозволяє підвищити точність обчислень, особливо в умовах асиметричних та нестабільних каналів зв'язку шляхом регулярного обліку варіативності затримок у мережі.

Базовий алгоритм пошуку екстремуму функції $f(x)$ методом Фібоначі на начальному відрізку $[a,b]$ був модіфікований наступним чином:

На начальній стадії алгоритму задаються значення змінних a і b як меж відрізку оптимізації $[\iota_{min},\iota_{max}]$ та обчислюється начальний розмір d_0 цього відрізку: $d_0 = \iota_{max} - \iota_{min}$, та число Фібоначчі F_n і число ітерацій n, що гарантують задану точність eps:

$$F_n \geq \frac{\iota_{max} - \iota_{min}}{eps} = \frac{d_0}{eps}. \tag{26}$$

На другому загальному етапі, що півторюється n раз, на кожному кроці k обчислюються зменшений розмір d_k інтервалу та нові пробні значення

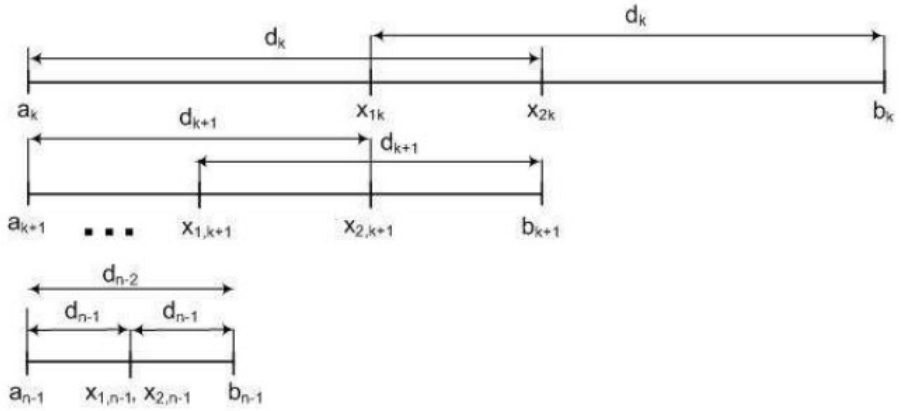

Рисунок. 3.17. Зменшення розміру d_k відрізку оптимізації в методі Фібоначчі

ι_1 та ι_2 за наступними формулами:

$$d_k = \frac{F_{n-k}}{F_{n-k+1}} d_{k-1}; \quad \iota_1 = b - d_k; \quad \iota_2 = a + d_k \tag{27}$$

На цьому кроці виконується оцінювання в точках ι_1 та ι_2 значень першої моди гістограми $H[rtt;\iota]$: $f_1 = \hat{f}(\iota_1)$ та $f_2 = \hat{f}(\iota_2)$ як медіан з m-тестових значень для кожної точки ι_1 та ι_2.

Далі виконується крок порівняння f_1 та f_2 та зміни границь відрізку оптимізації $[a,b]$.

Псевдокод алгоритму оптимізації інтервалу запитів ι

```
//Отримати інтерфейс до обчислювання стохастичної гістограми H[rtt; iota].
// Задати алгоритм обчислення функції моди f(iota)=H[rttm; iota],
//  що використовує метод пошуку першої моди (rttm) гістограми H[rtt; iota],
//   де iota - це аргумент оптимізації - значення інтервалу запитів.
// Задати розмір m тестової вибірки f(iota)
//Задати змінні меж a=iota_min, b=iota_max та необхідну точність eps.

function  iota_optimum(a=iota_min, b=iota_max, eps, H, f, m)
  d = b - a      // початковий розмір  відрізку оптимізації
  // Обчислити таке n, що  F[n] > d/eps,
  //  де F[n] - значення n-го числа Фібоначи (F[n] = F[n-1] + F[n-2], F[0]=F[1]=1)

   // загальний крок
   for(k=1;k<=n; k++){
        d = d * F[n-k]/F[n-k+1] // новий звужений поточний розмір  відрізку оптимізації
         i1 = b - d       // ліве пробне значення на поточному відрізку оптимізації [a,b]
         i2 = a + d       // праве пробне значення на поточному відрізку оптимізації [a,b]
```

```
      /*******************************************************
       f1 - оцінка значення функції f(i1) моди гістограми H[rtt; i1] з вибірки
            поточних тестових даних f(i1)[1],f(i1)[2],...f(i1)[m]
       f2 - оцінка функції f(i2) моди гістограми H[rtt; i2]  з вибірки
            поточних тестових даних f(i2)[1],f(i2)[2],...f(i2)[m]
      *******************************************************/
       f1 = mediana(f(i1)[1],f(i1)[2],...f(i1)[m])
       f2 = mediana(f(i2)[1],f(i2)[2],...f(i2)[m])
        if(f1 > f2) b = i2
              else  a = i1
}//end for
 return (a+b)/2
```

Двокритеріальна оптимізація

Для вищеописаного методу оптимізації передбачається відправлення фіксованого числа пакетів (за замовчуванням $N = 100$) для кожного вимірювання (оцінки) значення функції $f(\iota) = H[rtt_m; \iota]$ при заданому інтервалі запиту ι. Однак у загальному випадку слід проводити двокритеріальну оптимізацію максимуму гістограми за критеріями максимуму першої моди та мінімуму кількості запитів $(\max f, \min N)$.

Можна вирішувати завдання пошуку множини оптимумів за Парето чи якимось чином згортати множину критеріїв в один. Однак, не слід забувати про більш точне статистичне визначення моди за великих N, ніж за менших. Цю загальну інформацію та вимогу стабільності одержуваних параметрів можна конструктивно використовувати у переформулюванні завдання двокритеріальної оптимізації. Завдання двокритерильної оптимізації можна переформулювати в завдання послідовної умовної оптимізації у вигляді наступної двокрокової процедури.

1) Оптимізація інтервалу запитів ι для отримання ι_{opt} при максимальному значенні числа запитів $N = N_{max}$. Визначення стабільної оцінки θ для отриманої першої моди rtt_m.
2) Мінімізація числа запитів N при отриманому ι_{opt} таким чином, щоб величини θ та rtt_m були стабільними.

Для швидкого виконання процедури мінімізації N можна використовувати модифікований бінарний пошук межі області стабільності на відрізку $[N_{min}, N_{max}]$.

ВИСНОВКИ

У межах даної роботи було виконано аналіз завдання синхронізації у глобальних та локальних мережах. Незважаючи на численні праці в цій галузі та наявність робочих стандартів (NTP, PTP та інших), вирішення цього завдання в різних прикладних сферах вимагає докладання подальших зусиль для реалізації та вдосконалення методів вирішення задачі синхронізації. У дипломній роботі такою сферою були задачи синхронізації розподілених Web-додатків, для яких на даний час неможливе безпосереднє використання сервері в та протоколів синхронізації годинників. Крім того, вивчення математичних моделей синхронізації та реальних комп'ютерних мереж дозволяє розширити стандартні постановки задач синхронізації.

У дипломній роботі було отримано прості докази фундаментальних обмежень задачі відносної синхронізації. Зокрема, показано неможливість точного однокрокового визначення параметрів базової афінної моделі (θ та σ). Крім того, показано, що для багатокрокових методів неможливо позбутися систематичної помилки визначення параметра синхронізації без знання асиметрії каналу зв'язку. Однак, при цьому можна значно уточнювати параметр відносного дрейфу годинника σ.

Для оцінки параметра синхронізації (θ) були виведені зручні вирази для визначення меж систематичної похибки та її компенсації при отриманні інформації про коефіцієнт асиметрії каналу. В роботі показано, що, якщо ця інформація нам доступна хоча б частково, можна отримати більш точні інтервальні оцінки систематичної похибки обчислювання θ.

Аналіз реальних задач синхронізації і розуміння неможливості виконання точної синхронізації годинників навіть в ідеальних мережах допомогли сформулювати розширення задачі синхронізації - задачу синхрокоординації. Показано, що задачу синхрокоординації в ідеальних мережах можна виконати точно, а в реальних мережах можна для цієї задачи розробити спеціалізовані протоколи, що дозволяють виконувати надійну синхрокоординацію роботи частин розподілених додатків. Також отримані вимоги та інформаційні обмеження для таких додатків і протоколів.

Для реальних каналів не менш складною була задача оцінки параме-

трів синхронізації при отриманні стохастичних часових даних. Для цього було розроблено дослідницький додаток, на базі якого були одержані дані по розподілах затримок та часових параметрів. Для аналізу складних сумісних розподілів був розроблен модіфікований метод візуалізації - функціональна гістограма, яка дозволила зробити вибір найважливіших стабільних характеристик комплексного розподілу (першої моди розподілу затримок RTT і для цієї моди - першої умовної моди параметру θ).

Аналіз гістограм розподілу даних для синхронізації при різних інтервалах запиту до серверу привів до висновку про необхідність оптимізації вибраних параметрів гістограм (перших мод). Задача оптимізації ускладнювалась стохастичним характером даних, що отримуються. Для виконання оптимізації моди гістограми був розроблений метод стохастичної оптимізації на базі методу унімодальної оптимізації Фібоначчі. Для подальшого зменшення навантаження на канал і сервер було запропоновано процедуру двокритеріальної оптимізації.

Надійність та безпека програм, що спираються на серверні компоненти, які працюють у хмарних системах, істотно залежить від контролю синхронізації та параметрів мережі, що об'єднує компоненти програми в єдине ціле. Для виконання такого завдання розроблено тестовий розподілений Web-додаток, який можна використовувати як для контролю стану синхронізації годинників, так і для контролю каналів зв'язку.

Важливі результати дипломної роботи опубліковано в матеріалах міжнародної наукової інтернет-конференції на тему "Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення"[18], та в науково мультидисциплінарній монографії «Theoretical and Applied Foundations of Innovation in Modern Science» [19].

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A study of internet round-trip delay [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/2335581_A_Study_of_Internet_Round-trip_Delay (дата звернення: 16.06.2024).
2. Real-time Performance Monitoring Using Round-trip Time [Електронний ресурс] // Techniche. – Режим доступу: https://www.technichegroup.com/network-performance-monitoring-round-trip-time/ (дата звернення: 16.06.2024). – Назва з екрана.
3. Delay asymmetry correlation model [Електронний ресурс] – Режим доступу: https://curve.carleton.ca/system/files/etd/74f0b075-d19e-4255-bcc3-b297e876783e/etd_pdf/561dc5d8c3aa529b23a94b5e40fa9845/rahman-delayasymmetrycorrectionmodelforieee1588synchronization.pdf (дата звернення: 16.06.2024).
4. Li Y., Hu J., Ma L., Huang W., Zhang S., Luo Y., Zhou C., Zhang C., Wang H., Pan Y., Shao Y., Zhang Y., Chen X., Chen Z., Yu S., Guo H†, Xu B*. Secure two-way fiber-optic time transfer against sub-ns asymmetric delay attack // Optical Society of America. – 2020.[Електронний ресурс] – Режим доступу: https://www.osapublishing.org/abstract.cfm?URI=cleo-2020-M4I.3.
5. Clock synchronization [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Clock_synchronization (дата звернення: 16.06.2024).
6. Probabilystyc clock synchronization [Електронний ресурс] – Режим доступу: http://www.cs.utexas.edu/users/lorenzo/corsi/cs380d/papers/Cristian.pdf (дата звернення: 16.06.2024).
7. Synchronizing Distributed Applications: Harnessing the Power of Distributed Systems [Електронний ресурс] – Режим доступу: https://tinyurl.com/2mm2dnnw(дата звернення: 16.06.2024).
8. Fundamental Limits on Synchronizing Clocks Over Networks [Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/224183858_Fundamental_Limits_on_Synchronizing_Clocks_Over_Networks (дата звернення: 16.06.2024).

9. Lamport L. Time, clocks and the ordering of events in a distributed system // Commun. ACM. – 1978. – Т. 21, № 7. – С. 558–565.
10. Solis R., Borkar V., Kumar P.R. A new distributed time synchronization protocol for multihop wireless networks // Proc. 45th IEEE Conf. Decision and Control, San Diego, CA. – 2006. – С. 2734–2739.
11. Giridhar A., Kumar P.R. Distributed clock synchronization over wireless networks: Algorithms and analysis // Proc. 45th IEEE Conf. Decision and Control, San Diego, CA. – 2006. – С. 4915–4920.
12. Pathak A., Pucha H., Zhang Y., Hu Y.C., Mao Z. A Measurement Study of Internet Delay Asymmetry // Purdue University, University of Michigan. – Режим доступу: https://web.eecs.umich.edu/~zmao/Papers/pam08_owd.pdf.
13. Mills D.L. Internet time synchronization: The network time protocol // IEEE Trans. Commun. – 1991. – Т. 39, № 10. – С. 1482–1493.
14. Kopetz H., Ochsenreiter W. Clock synchronization in distributed real-time systems // IEEE Trans. Computers. – 1987. – Т. C-36, № 8. – С. 933–939.
15. Allan D.W. Time and frequency (time-domain) characterization, estimation, and prediction of precision clocks and oscillators // IEEE Trans. Ultrasonics, Ferroelectrics, and Frequency Control. – 1987. – Т. UFFC-34. – С. 647–654.
16. Gurewitz O., Cidon I., Sidi M. One-way delay estimation using network-wide measurements // IEEE/ACM Trans. Netw. – 2006. – Т. 14, № 6. – С. 2710–2724.
17. Малярец Л. М., Егоршин А. А. ТЕОРИЯ ВЕРОЯТНОСТЕЙ И МАТЕМАТИЧЕСКАЯ СТАТИСТИКА / Л. М. Малярец, А. А. Егоршин. – Харьков: Изд. ХНЭУ, 2013. – 301 с.
18. Проблеми синхронізації годинників та роботи розподілених додатків реального часу // Наукові конференції [Електронний ресурс]. – Режим доступу: http://www.konferenciaonline.org.ua/ua/article/id-1762/ (дата звернення: 16.06.2024).
19. Проблеми і методи вирішення синхронізації годинників у розподілених системах // Українська мультидисциплінарна монографія. 2024. – С. 196-200. – Режим доступу: https://drive.google.com/file/d/1v6Gh5ZAPLJ698jR8KEfHa-FyAdyES-2E/view.

ДОДАТОК А

HTTP та WebSocket сервер на Express для обробки запитів про час та обмін даних в реальному часі:

```
const express = require('express');
const http = require("http");
const WebSocket = require("ws");
const app = express();
const server = http.createServer(app);
const wss = new WebSocket.Server({ server });
app.use(express.static('./'));
// Обробник для HTTP GET запитів на отримання часу
app.get('/gettime/:num', (request, response) => {
    let t = Date.now();
    let number = request.params.num;
    // Формування відповіді з серверним часом і номером
    let res = "" + t + "/" + number;
    response.write(res);
    response.end();
});
// Обробник підключень WebSocket
wss.on("connection", (ws) => {
    console.log("Клієнт WebSocket підключився");
    const data = { ask: "отримати час сервера через WebSocket!" };
    // Відправлення початкових даних клієнту через WebSocket
    ws.send(JSON.stringify(data));
    // Логіка обробки повідомлень від клієнта
    ws.on("message", (message) => {
        let m = JSON.parse(message);
        console.log("Отримано повідомлення від клієнта:", m.t, " ", m.k);
        let T1 = Date.now();
        // Відправлення відповіді на отримане повідомлення через WebSocket
        ws.send(JSON.stringify({ ts: T1, t: m.t, pt: m.pt, k: m.k }));
    });

    ws.on("close", () => { console.log("Клієнт WebSocket відключився"); });
});
server.listen(8080);
```

ДОДАТОК Б

WebSocket клієнт для обміну даними про час та обчислення RTT (Round-Trip Time), оновлення гістограми RTT та обчислення θ (theta) для різних статистичних параметрів:

```
let socket;
let socketTimerId;
let wsrequestCounter_ = -1;
let websocket_rtt = new Array(100).fill(-1);
let websocket_rtt_Hist = new Array(300).fill(0);
let HistPSocket = Array(10000).fill(0);
let RTTSocket = [];

let websocketItems = [];
let websocketDataset = new vis.DataSet(websocketItems);

let Smaxrttp = 0.0;
let Sminrttp = 1000.0;
let Smaxrtt = 0;
let Sminrtt = 1000;
let SmaxHist = 0;
let SmodaRTT = 0.0;
let S = 0.0;
let Srttmoda = 0.0;
let Srttmin = 0.0;

let websocketOptions = options;

let websocketGraph2d = new vis.Graph2d(visualization_socket,
                                       websocketDataset,
                                       websocketOptions);
websocketGraph2d.setWindow(1, 256, { animation: false });

function startGettingTimeFromServerWithIntervalbyWebSocket() {
    startSound.play();
    let delta = +reqInterval.value;
    wsrequestCounter_ = -1;
    websocket_rtt.fill(-1);
    RTTSocket = [];
    websocket_rtt_Hist.fill(0);
```

```
    websocketDataset.clear();
    summaryName.innerHTML =
    "WebSocket RTTs histogramm with colored  for request interval ="
    + delta + " ms";
    myHistogramm.fillAndDrawFadedHist();
    Smaxrttp = 0.0;
    Sminrttp = 1000.0;
    Smaxrtt = 0;
    Sminrtt = 1000;
    SmaxHist = 0;
    SmodaRTT = 0.0;
    S = 0.0;
    Srttmoda = 0.0;
    Srttmin = 0.0;
    socketTimerId = setInterval(testwebsocket, delta);
}

function processWebSocketRttHistogram() {
    let s = "WebSocket RTT Histogram:\n";
    for (let i = 0; i < websocket_rtt_Hist.length; i++) {
        if (websocket_rtt_Hist[i] > 0) {
            s += `[${i}]=${websocket_rtt_Hist[i]}\n`;
        }
    }
    webSocketRTTMonitor.innerHTML = s;
}

function processPreciseRttHistogramSocket() {
    let s = ``;
    let istart = (Math.floor(Sminrttp)  0) * 10  0;
    let ifinish = (Math.ceil(Smaxrttp)  0) * 10  0;
    let im = istart;
    let m = HistPSocket[im];

    for (let i = istart; i <= ifinish; i++) {
        if (i % 10 == 0) s += '\n[' + ((i / 10) | 0) + ']:        \t';
        let g = HistPSocket[i];
        if (m < g) { m = g; im = i; }
        s += `${g}\t`;
    }
```

```
let modamessage = `moda value=${m} at [${im / 10}]\n`;
PHistMonitorSocket.innerHTML = "Socket Precise RTT Histogram\n"
                                + modamessage + s;

RTTSocket.sort((a, b) => {
    let d = a[0] - b[0];
    if (d == 0) d = a[1] - b[1];
    return d;
});

let l = RTTSocket.length;
s = '(rtt, req)\t-->\t [ms]\n';

for (let i = 0; i < l; i++) {
    let r = RTTSocket[i];
    let rtt = r[0];
    let req = r[1];
    let  = r[2];
    s += `(${rtt}, ${req})\t-->\t${}\n`;
}

ThetaMonitorSocket.innerHTML = s;

RTTSocket.sort((a, b) => {
    let d = a[2] - b[2];
    if (d == 0) d = a[0] - b[0];
    if (d == 0) d = a[1] - b[1];
    return d;
});

let min = RTTSocket[0][2];
let max = RTTSocket[l - 1][2];
let delta = max - min;
let D = Math.ceil(1 + 2 * delta);
let H = new Array(D).fill(0);

for (i = 0; i < l; i++) {
    let r = RTTSocket[i];
    let r_ = (2 * (r[2] - min)) | 0;
    H[r_]++;
}
```

```
    s = ' Histogram:\n';

    for (k = 0; k < D; k++) {
        if (H[k] == 0) continue;
        let t = k / 2 + min;
        s += `=${t.toFixed(1)}\t${H[k]}\n`;
    }

    HistThetaSocket.innerHTML = s;
}

function testwebsocket() {
    try {
        let wsrc = ++wsrequestCounter_;
        if (wsrc > 99) {
            clearInterval(socketTimerId);
            endSound.play();
            processPreciseRttHistogramSocket();
            return;
        }
        let t0 = Date.now();
        let pt0 = performance.now();
        let mess = JSON.stringify({ t: t0, pt: pt0, k: wsrc });
        socket.send(mess);
    } catch (error) {
        console.error("Error during testwebsocket:", error);
        webSocketRTTMonitor.innerHTML = "\n Error during testwebsocket: "
                                        + error;
        clearInterval(socketTimerId);
    }
}

const serverHostName = location.host;
const staticAddress = "wss://" + serverHostName;

function linkToWebSocket() {
    try {
        socket = new WebSocket(staticAddress);

        socket.addEventListener("open", (event) => {
```

```
    console.log("WebSocket opened");
    webSocketRTTMonitor.innerHTML = "\n WebSocket opened";
});

socket.addEventListener("message", (event) => {
    let t3 = Date.now();
    let pt3 = performance.now();
    const data = JSON.parse(event.data);

    let k = +data.k;
    if (data.k == undefined) {
        webSocketRTTMonitor.innerHTML += "\n" + event.data;
        return;
    }

    let T2 = +data.ts;
    let t1 = +data.t;
    let pt1 = +data.pt;

    let rtt = t3 - t1;
    let prtt = pt3 - pt1;
    let 2 = rtt / 2;

    if (rtt <= 0) rtt = 0;
    if (prtt <= 0) prtt = 0;
    if (rtt > 999) rtt = 999;
    if (prtt > 999) prtt = 999;

    if (Smaxrtt < rtt) Smaxrtt = rtt;
    if (Smaxrttp < prtt) Smaxrttp = prtt;
    if (Sminrttp > prtt) Sminrttp = prtt;

    let T3 = T2;
    let d = T3 - t1;
    let d_ = t3 - T3;
    let  = d - 2;
    let _ = 2 - d_;

    RTTSocket.push([rtt, k, ]);
    websocket_rtt[k] = rtt;
    websocket_rtt_Hist[rtt]++;
```

```
if (SmaxHist < websocket_rtt_Hist[rtt]) {
    SmodaRTT = rtt;
    SmaxHist = websocket_rtt_Hist[rtt];
    Srttmoda = ;
}

if (Sminrtt > rtt) {
    Sminrtt = rtt;
    Srttmin = ;
}

let  = 0.0;
if (k < 5) S = Srttmin;
else if (rtt <= 2 * SmodaRTT - Sminrtt) {
    let d = rtt - Sminrtt;
     = 1 / (5 + d * d); // empirical formula
    let _ = 1 - ;
    S =  *  + _ * S; // exponential averaged of
}

HistPSocket[(Math.floor(prtt * 10.0))  0]++;
processWebSocketRttHistogram();
```

ДОДАТОК В

HTTP клієнт для обміну даними про час та обчислення RTT (Round-Trip Time), оновлення гістограми RTT та обчислення θ (theta) для різних статистичних параметрів:

```
const startSound = new Audio("c6.mp3");
startSound.loop = false;
const endSound = new Audio("c7.mp3");
endSound.loop = false;
const minute_ms = 60 * 1000;
let requestCounter = 0;
let timerId = undefined;
let RTT = [];
let RTT = [];
let items = [];
let dataset = new vis.DataSet(items);

let options = {
    style: 'bar',
    barChart: { width: 25, align: 'center' },
    drawPoints: false,
    dataAxis: {
        icons: false,
        left: {
            title: {
                text: 'Quantity',
                style: 'font-weight: bold; color: #00e5ff;'
            }
        }
    },
    showMajorLabels: false,
    max: 1000,
    min: 0,
    start: 1,
    end: 256,
    orientation: 'bottom',
    xAxis: {
        valueLabels: {
            content: function(value) {
                return value + ' ms';
```

```
            },
            fontSize: 12,
            fontColor: '#00e5ff'
        }
    }
};

let graph2d = new vis.Graph2d(visualization, dataset, options);
graph2d.setWindow(1, 256, { animation: false });

let Hist = Array(1000).fill(0);
let MHist = Array(1000).fill(0);
let HistP = Array(10000).fill(0);
let maxrttp = 0.0;
let minrttp = 1000.0;
let maxrtt = 0;
let minrtt = 1000;
let maxHist = 0;
let modaRTT = 0.0;
let  = 0.0;
let rttmoda = 0.0;
let rttmin = 0.0;

async function getTimeFromServer() {
    const rc = ++requestCounter;
    if (rc > 100) {
        clearInterval(timerId);
        processPreciseRttHistogram();
        return;
    }

    let T1, T2, t3, 2, d, d_, , _;
    let t0 = Date.now();
    let pt0 = performance.now();
    let response = await fetch('/gettime/' + rc);
    let pt1 = performance.now();
    let t1 = Date.now();
    t3 = t1;

    let text = await response.text();
    let parts = text.split("/");
```

```
    let requestCounter_ = +parts[1];
}
let prtt = pt1 - pt0;
let rtt = t1 - t0;
let 2 = rtt / 2;
if (rtt <= 0) rtt = 0;
if (prtt <= 0) prtt = 0;
if (rtt > 999) rtt = 999;
if (prtt > 999) prtt = 999;
if (maxrtt < rtt) maxrtt = rtt;
if (maxrttp < prtt) maxrttp = prtt;
if (minrttp > prtt) minrttp = prtt;
HistP[(Math.floor(prtt * 10.0)) | 0]++;
RTT.push(rtt);
Hist[rtt]++;

let ts = +parts[0];
T1 = ts;
d = T1 - t0;
d_ = t3 - T1;
 = d - 2;

if (maxHist < Hist[rtt]) {
    modaRTT = rtt;
    maxHist = Hist[modaRTT];
    rttmoda = ;
}
if (minrtt > rtt) {
    minrtt = rtt;
    rttmin = ;
}
let  = 0.0;
if (rc < 5)  = rttmin;
else if (rtt <= 2 * modaRTT - minrtt) {
    let d = rtt - minrtt;
     = 1 / (5 + d * d); // empirical formula
    let _ = 1 - ;
     =  *  + _ * ; // exponential averaged of
}

_ = 2 - d_;
```

```
RTT.push([rtt, rc, ]);

let report = `rc=${rc}     rtt=${rtt} ms
                 prtt=${prtt.toFixed(1)} ms
                    current  = ${.toFixed(1)} ms<br>
                special  averaged value  = ${.toFixed(1)} ms
             (=${.toFixed(2)})  <br>
                 rtt moda sample value  = ${rttmoda.toFixed(1)} ms
              (for moda rtt=${modaRTT})  <br>
                 rrt min sample value  = ${rttmin.toFixed(1)} ms
              (for min rtt=${minrtt})  `;

responseMonitor.innerHTML = report;
responseMonitor.style.color = `hsl(${rc * 3.6} 100% 30%)`;
processRttHistogram();
dataset.add({ x: rtt, y: Hist[rtt] });
myHistogramm.addItem(rtt, );
let x0 = ((minrtt / 10) | 0) * 10;
let x1 = x0 + 50;
graph2d.setWindow(x0, x1, { animation: false });
}

function processPreciseRttHistogram() {
    let s = ``;
    let istart = (Math.floor(minrttp)  0) * 10  0;
    let ifinish = (Math.ceil(maxrttp)  0) * 10  0;
    let im = istart;
    let m = HistP[im];
    for (let i = istart; i <= ifinish; i++) {
        if (i % 10 == 0) s += '\n[' + ((i / 10) | 0) + ']:\t';
        let g = HistP[i];
        if (m < g) {
            m = g;
            im = i;
        }
        s += `${g}\t`;
    }
    let modamessage = `moda value=${m} at [${im / 10}]\n`;
    PHistMonitor.innerHTML = "Precise rtt Histogram\n" + modamessage + s;
    RTT.sort((a, b) => {
        let d = a[0] - b[0];
```

```
        if (d == 0) d = a[1] - b[1];
        return d;
    });
    console.dir(RTT);
    s = '(rtt, req)\t-->\t [ms]\n';
    let l = RTT.length;
    for (let i = 0; i < l; i++) {
        let r = RTT[i];
        let rtt = r[0];
        let req = r[1];
        let  = r[2];
        s += `(${rtt}, ${req})\t-->\t${}\n`;
    }
    ThetaMonitor.innerHTML = s;

    RTT.sort((a, b) => {
        let d = a[2] - b[2];
        if (d == 0) d = a[0] - b[0];
        if (d == 0) d = a[1] - b[1];
        return d;
    });
    let min = RTT[0][2];
    let max = RTT[l - 1][2];
    let delta = max - min;
    let D = Math.ceil(1 + 2 * delta);
    let H = new Array(D).fill(0);
    for (i = 0; i < l; i++) {
        let r = RTT[i];
        let r_ = (2 * (r[2] - min)) | 0;
        H[r_]++;
    }
    s = ' Histogram:\n';
    for (k = 0; k < D; k++) {
        if (H[k] == 0)
            continue;
        let t = k / 2 + min;
        s += `=${t.toFixed(1)}\t${H[k]}\n`;
    }
    HistTheta.innerHTML = s;
}
```

```
function processRttHistogram() {
    let s = `rtt histogram:\n`;
    for (let rtt = minrtt; rtt <= maxrtt; rtt++) {
        let g = Hist[rtt];
        if (g == 0)
            continue;
        s += `[${rtt}]=${g}\n`;
    }
    HistMonitor.innerHTML = s;
}
    let delta = +reqInterval.value;
    requestCounter = 0;
    RTT = [];
    RTT = [];
    Hist = Array(1000).fill(0);
    MHist = Array(1000).fill(0);
    HistP = Array(10000).fill(0);
    maxrttp = 0.0;
    minrttp = 1000.0;
    maxrtt = 0;
    minrtt = 1000;
    maxHist = 0;
    items = [];
    dataset.clear();
    summaryName.innerHTML =
    "HTTP RTTs histogramm  with colored  for request interval ="
    + delta + " ms";
    myHistogramm.fillAndDrawFadedHist();
    timerId = setInterval(getTimeFromServer, delta);
}
```

ДОДАТОК Г

Реалізация графічного інтерфейсу для відображення та аналізу параметрів RTT та θ:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>rtt  evaluation</title>
    <link rel="stylesheet" href="https://visjs.github.io/vis-timeline/styles
    /vis-timeline-graph2d.min.css">
    <link rel="stylesheet" href="style.css">
    <link rel="stylesheet" href="myHist.css">
    <script src="https://visjs.github.io/vis-timeline/standalone/umd/vis-timeline
    -graph2d.min.js"></script>
</head>
<body>
    <div class="container">
        <h1>NetTrend Analyzer</h1>
        <input id="reqInterval" value="250"><i>ms</i></input>
        <button id="reqtime" onclick="startGettingTimeFromServerWithInterval()"
        >(re)Start getting time from server by HTTP</button>
        <hr>
        <div id="responseMonitor">........</div>
        <div class="row">
            <div class="col">
                <pre id="HistMonitor">........
                    <div class="hist-label-side">Side Label</div>
                    <div class="hist-label-bottom">Bottom Label</div>
                </pre>
                <details>
                    <summary>Precise RTT Histogram (text)</summary>
                    <pre id="PHistMonitor">........
                        <div class="hist-label-side">Side Label</div>
                        <div class="hist-label-bottom">Bottom Label</div>
                    </pre>
                </details>
                <hr>
                <details>
```

```
                <summary>Computed  Log</summary>
                <pre id="ThetaMonitor">........</pre>
            </details>
            <hr>
            <details>
                <summary>Computed  Histogram (text)</summary>
                <pre id="HistTheta">........
                    <div class="hist-label-side">Side Label</div>
                    <div class="hist-label-bottom">Bottom Label</div>
                </pre>
            </details>
        </div>
        <div class="col">
            <h3>rtt histogram by HTTP</h3>
            <div id="visualization"></div>
        </div>
    </div>
    <hr>
    <button id="requestwebsocket"

    onclick="startGettingTimeFromServerWithIntervalbyWebSocket()">
    (re)start Getting RTTs and Server times by WebSocket</button>
    <pre id="WebSocketRTTMonitorPre">...</pre>
    <div class="row">
        <div class="col">
            <pre id="webSocketRTTMonitor">........
                <div class="hist-label-side">Side Label</div>
                <div class="hist-label-bottom">Bottom Label</div>
            </pre>
            <details>
                <summary>WebSocket Precise RTT Histogram (text)</summary>
                <pre id="PHistMonitorSocket">........
                    <div class="hist-label-side">Side Label</div>
                    <div class="hist-label-bottom">Bottom Label</div>
                </pre>
            </details>
            <hr>
            <details>
                <summary>WebSocket Computed  Log</summary>
                <pre id="ThetaMonitorSocket">........</pre>
            </details>
```

```
                <hr>
                <details>
                    <summary>WebSocket Computed  Histogram (text)</summary>
                    <pre id="HistThetaSocket">........
                        <div class="hist-label-side">Side Label</div>
                        <div class="hist-label-bottom">Bottom Label</div>
                    </pre>
                </details>
            </div>
            <div class="col">
                 <h3>rtt histogram by WebSocket</h3>
                <div id="visualization_socket"></div>
            </div>
        </div>
        <hr>
        <details>
            <summary id="summaryName">test my Histogram</summary>
        </details>
        <hr>
        <div class="Histogramm" id="myHist"

        onclick="startGettingTimeFromServerWithIntervalbyWebSocket()">
            <div class="hist-label-side">Side Label</div>
            <div class="hist-label-bottom">Bottom Label</div>
        </div>
    </div>

    <script src="myHist.js"></script>
    <script src="getByHTTP.js"></script>
    <script src="getByWebSocket.js"></script>
</body>
</html>
```

ДОДАТОК Д

CSS Стилі для зручного Відображення Даних:

```
#visualization,
#visualization_socket {width: 100%;}
h1,
pre,
#RTTprocess,
#requestwebsocket,
#WebSocketRTTMonitorPre
                    {color: blue;}

i,
#reqInterval,
#reqtime,
#responseMonitor {color: hsl(0 100% 70%);}
  #reqInterval {width: 40px;}

#HistTheta {color: hsl(150 100% 30%);}
th,td,
#ThetaMonitor,
#HistMonitor {color: hsl(100 100% 30%);}

 body {
            font-family: 'Roboto', sans-serif;
            background-color: #121212;
            color: #fff;
            margin: 0;
            padding: 20px;
        }
        h1 {
            text-align: center;
            margin-bottom: 30px;
            color: #00e5ff;
            font-size: 32px;
            text-shadow: 0 0 10px rgba(0, 230, 255, 0.5);
        }
        .container {
            max-width: 1700px;
            margin: 0 auto;
            background-color: #1e1e1e;
```

```
    padding: 30px;
    border-radius: 10px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
}
pre {
    background-color: #262626;
    padding: 20px;
    border-radius: 5px;
    overflow-x: auto;
    color: #00e5ff;
    position: relative;
    box-shadow: 0 0 10px rgba(0, 230, 255, 0.3);
}
button {
    background: linear-gradient(45deg, #00e5ff, #00b8d4);
    color: #121212;
    border: none;
    padding: 12px 24px;
    border-radius: 5px;
    cursor: pointer;
    transition: all 0.3s ease;
    box-shadow: 0 0 10px rgba(0, 230, 255, 0.5);
}
button:hover {
    transform: scale(1.05);
    box-shadow: 0 0 20px rgba(0, 230, 255, 0.8);
}
details {
    background-color: #262626;
    color: #fff;
    padding: 20px;
    border-radius: 5px;
    margin-bottom: 20px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
}
summary {
    cursor: pointer;
    font-weight: bold;
    color: #00e5ff;
    text-shadow: 0 0 5px rgba(0, 230, 255, 0.5);
}
```

```
.row {
    display: flex;
    flex-wrap: wrap;
    margin: -10px;
}
.col {
    flex: 1;
    padding: 10px;
    position: relative;
}
.hist-label-side {
    position: absolute;
    left: -50px;
    top: 50%;
    transform: translateY(-50%) rotate(-90deg);
    color: #ff6d00;
    font-weight: bold;
    text-shadow: 0 0 5px rgba(255, 109, 0, 0.5);
}
.hist-label-bottom {
    position: absolute;
    bottom: -30px;
    left: 50%;
    transform: translateX(-50%);
    color: #00e5ff;
    font-weight: bold;
    text-shadow: 0 0 5px rgba(0, 230, 255, 0.5);
}
input[type="text"] {
    background-color: #262626;
    color: #00e5ff;
    border: none;
    padding: 5px 10px;
    border-radius: 3px;
    box-shadow: 0 0 5px rgba(0, 230, 255, 0.3);
}
i {
    color: #ff6d00;
    text-shadow: 0 0 5px rgba(255, 109, 0, 0.5);
}
```