КОМП'ЮТЕРНІ НАУКИ

УДК 004.4 DOI: https://doi.org/10.32515/2414-3820.2024.54.106-116

О.С. Улічев, канд. техн. наук
Центральноукраїнський національний технічний університет, м. Кропивницький, Україна
О.В. Ревнюк, асп.
Приватний вищий навчальний заклад "Європейський університет", м. Київ, Україна
e-mail: askin79@gmail.com,o.revnyuk@edu.ua

Стиснення заголовків у протоколі HTTP 2, аналіз переваг та недоліків використовуваного методу

Використання стиснення заголовків у HTTP/2 значно знижує обсяг даних, які передаються мережею, що робить комунікацію ефективнішою порівняно з HTTP1, де стиснення було можливе лише для тіла запиту. Стиснення заголовків потрібне через їх повторюваність, особливо при однакових значеннях, таких як заголовки method, path тощо. Для реалізації стиснення в протоколі HTTP 2 використовується метод HPACK, який комбінує таблиці з частими значеннями заголовків і кодування Хаффмана для зменшення розміру даних. Це дозволяє досягти суттєвих переваг при повторних запитах до того самого ресурсу. У дослідженні детально досліджуються реалізації HPACK у HTTP 2 та переваги в порівнянні з іншими методами стиснення. Також аналізуються методи, які включають механізм динамічних та статичних таблиць, використання кодування Хаффмана та дослідження ризиків, пов'язаних із компрометацією інформації під час стиснення. Переваги такого підходу також підтверджені на практичних прикладах.
стиснення заголовків HTTP2, HPACK, таблиці заголовків, кодування Хаффмана, HTTP2

Постановка проблеми. Сучасні веб-додатки стали більш складні і створюють велику кількість запитів на сервер. Часто запити не є сильно унікальними між собою і відправляють багато інформації, що дублюється. Також в мобільних пристроях заголовки запитів є більші по розміру, але так само повторюються та девайси підключені, наприклад через мережу EDGE або 3G, мають обмежену пропускну здатність. Новий протокол HTTP 2 був створений для оптимізації мережевої комунікації, але він ніяк не вирішує проблему зменшенняоб'єму передачі даних між клієнтом і сервером.

Аналіз останніх досліджень і публікацій. Варун Сінгх у подкасті «Evolution of Internet Protocols» [1] акцентує увагу на важливості переходу на HTTP2 для підвищення ефективності передачі даних та стиснення заголовків. Також він зробив огляд еволюції інтернету загалом та передумови створення нових версій HTTP. Олександр Кравчук у своєму дослідженні «Аспекти переходу на HTTP 2» [2] аналізує складність переходу на HTTP 2, наголошуючи на скороченні затримок запитів завдяки використанню стиснення заголовків. Автор також розглядає можливості покращення продуктивності веб-додатків за рахунок впровадження нового протоколу. Роберт Пеон та ЕвреРуеллан є провідними інженерами, які приймали участь у роботі над HTTP 2 та HPACK. У документі RFC7541 [3] детально описано алгоритм HPACK для стиснення заголовків, що поєднує динамічні та статичні таблиці з кодуванням Хаффмана. Автори підкреслюють, що це дозволяє суттєво зменшити обсяг даних при повторюваних запитах до сервера. Перевозніков С. І. та Горобець Ю. В. [4] порівнюють час

завантаження веб-сторінок при використанні протоколів HTTP1 та HTTP2, демонструючи, що другий значно швидший завдяки стисненню заголовків. Їх дослідження підтверджують, що це особливо ефективно в умовах мобільних мереж із високою латентністю. Дрововозов В. І. та Хемраєв А. К. [5] аналізують надлишковість у протоколах TCP/IP, що створює додаткове навантаження на мережу. Вони обґрунтовують необхідність мінімізації об'єму передачі даних. У роботі «HTTPS-Only: Upgrading all connections to HTTPS in web browsers» [6] обговорюється важливість використання HTTPS з'єднання для передачі даних мережею і автоматичне оновлення всіх з'єднань в браузері. Також автори акцентують увагу на передачі cookies, саме захищеним підключенням з шифруванням даних. Автор роботи «Дослідження та розроблення системи вибору оптимального алгоритму стиснення даних при резервному копіюванні» [7] розглядає проблему вибору ефективного алгоритму стиснення, саме порівняння існуючих та автоматичний вибір оптимального алгоритму в залежності від типу даних. Розглядає різні перспективи стиснення даних, які можна використати для заголовків. Тема швидкості передачі даних активно обговорюється авторами, згаданими в цій статті, і ведеться наукова діяльність навколо нової версії HTTP протоколу. Це підкреслює постійний інтерес оптимізації в області мережевих технологій, а саме, швидкості роботи протоколів HTTP.

Постановка завдання. Метою статті є аналіз алгоритму стиснення заголовків у протоколі HTTP 2 для оптимізації передачі даних мережею у сучасних веб додатках. А також огляд передумов створення HPACK, та потенційних загроз і ризиків втрати даних, що пов'язані з стисненням. Проблеми передачі cookies у заголовках та вибір оптимального способу стиснення для даних невеликого об'єму.

Виклад основного матеріалу. HTTP 2, представлений у 2015 році, став важливим етапом у розвитку веб-протоколів, покращуючи продуктивність та ефективність обміну даними між клієнтом і сервером [1]. На відміну від попередньої версії HTTP/1.1, яка використовувала окремі з'єднання для кожного запиту, HTTP 2 підтримує мультиплексування – передачу кількох запитів і відповідей через одне з'єднання. Це дозволяє зменшити затримки, викликані очікуванням окремих відповідей, і значно прискорює завантаження веб-сторінок, особливо на сайтах із великою кількістю ресурсів[2].

Ще одним важливим нововведенням є стиснення заголовків, що вирішує проблему надмірності та повторюваності інформації у заголовках запитів і відповідей. У HTTP 1.1 заголовки кожного запиту мали повторювані дані, які щоразу надсилалися на сервер, створюючи зайве навантаження на мережу. HTTP 2 вирішує цю проблему за допомогою технології HPACK [3], яка реалізує стиснення заголовків через використання статичних та динамічних таблиць заголовків. Це дозволяє зменшити обсяг даних, які передаються між клієнтом і сервером, а також знижує затрати на обробку інформації.

HPACK використовує кодування Хаффмана та інші техніки, які дозволяють ефективно кодувати дані змінною довжиною, залежно від частоти їх використання. Це дозволяє економити мережеві ресурси, зокрема для популярних сайтів, які обробляють величезну кількість запитів [4]. Незважаючи на загальні переваги, компресія даних може створювати ризики втрати даних, а також неоптимальне використання алгоритмів стиснення.

Загалом, розмір HTTP заголовків менший у порівнянні з тілом запиту, проте вони містять значну кількість повторюваних даних, що створює зайве навантаження. Наприклад, звичайний запит HTTP2 GET до Facebook виглядає наступним чином (рис.1).

:authority:	static.xx.fbcdn.net
:method:	GET
:path:	/btmanifest/1017706399/facebook/main
:scheme:	https
Accept:	*/*
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	en-GB,en-US;q=0.9,en;q=0.8
Origin:	https://www.facebook.com
Priority:	u=1, i
Referer:	https://www.facebook.com/
Sec-Ch-Ua:	"Google Chrome";v="129", "Not=A?Brand";v="8", "Chromium";v="129"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"macOS"
Sec-Fetch-Dest:	empty
Sec-Fetch-Mode:	cors
Sec-Fetch-Site:	cross-site
User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36

Рисунок 1 – Запит до Facebook

Джерело: розроблено авторами

Із 612 символів у цьому запиті лише 39 можуть змінитися при наступному запиті: метод запиту (GET) та шлях (/btmanifest/1017706399/facebook/main). Фактично, у більшості випадків метод залишається незмінним, тому, що змінюється лише значення path [5]. Отже, 573символи у кожному запиті повторюються, створюючи надмірні дані. Це питання стає ще більш критичним, коли деякі заголовки: accept і user-agent, можуть бути дуже довгими, а заголовки з cookies – ще довшими. Наприклад, типовий запит до Google включає значну кількість cookies (рис.2):

:authority:	waa-pa.clients6.google.com
:method:	POST
:path:	/$rpc/google.internal.waa.v1.Waa/Ping
:scheme:	https
Accept:	*/*
Accept-Encoding:	gzip, deflate, br, zstd
Accept-Language:	en-GB,en-US;q=0.9,en;q=0.8
Authorization:	SAPISIDHASH 1730028886_9479d12acce65c03fe48e7071129cb74de186527
Content-Length:	1696
Content-Type:	application/json+protobuf
Cookie:	__Secure-3PAPISID=8Wep-o_bMCMQF1Qm/ArxeHD6J0bX_kBlaf; __Secure-3PSID=g.a000pQq2j9FxT5m7l5IL9eAiBSSkNV9RPjs_EtlcM7hjFesYTWZCuLjiApLmrSRVAc0DShZ1FgACgYKAWISAQASFQHGX2MixEwVkslcf6TpETGQwkDDHhoVAUF8yKqr_TcQCwe3rP5Cdpa-r6rG0076; NID=518=Ne9UQmENwGnuUELP3C2eY5ZmY0y4NhB6kxxMl9p3ct89JHW0ZYqRg9dlA3kGz-J-jV9NtdH-Tvrf2lHsdLZCHKSHWz4DeoOAhyHQzJBjbdl3K9pZdCDi73wwUeVuYepjPaBxlk893i6a_LC8WW1lGtZKj_Qik2u7wkX2PbYNMLOytGfjGiQSr5vcrpHr4s0FNCkTEobVre5qERG3LW3rol05ap0rYSGmpGYKNaAUZ-N6dUummLGC6TGhCjqvexhZzu3fbuHkgdvHFgTmVjzoT9R-mWVRHZdOhL-4ShkTMOEoCFZW2vcUFlazJ6zDhdJT4IEcppjRPRp8o_zhjQL2QCKdWOS057i0kSGiFWBwVBKoi5KQdIeqmmorPg2oN98sGQgLpHGRo0SHcDrWg83K9qTy8KNg1x_IS9PKBYJ7tQm3gx4u_6qZfyqe48EMqJPFks9NbaaSoHeZGeUAiTy83SnbVHNXAlww1DnhrEiTZVKuX1z_abC6AyH6b8jD2qWn8vNxk7EaFL78d-mBtpdXJ9Ej4vyccnqrjXDmHtNOW3PTKXVPHaoRm3UuowHU3c1_brRHUCRG1PmrjSe3BZZ_68oGWJP-b1U6P1Dsw_rHJvtVMoVqJWZbm_VMu1ZBi2PMLtai-meh-0z0rsH4zzt0PJ84a6El8lfa3HXQmejqhoic5oag9XsbNphuF8nlETJ05E-v7OvB4NagxfO5DhQC-ch_Kyg-3sl_QOMWTN55R0CTvBi-8oSF42UrZA7ItxenzTcMSmDOGE6Q7XES6Jbie3ITiXIfgMz0kp9HRg1cs6FBWG8pU4Aun5gV6bGGTdgS_Q5akr_KL8au_3Oc1Qva2oU9PJ_RiDCGCdlryKhXyRxYTOFhU3kv0ibOCTKTJeo; __Secure-3PSIDTS=sidts-CjIBQT4rXz_Egh_wVNrTq9U14__B2Z5BeBSg1gU8ztxUDkSfhv8TbPmKYpRUDBJKZ0C8ABAA; __Secure-3PSIDCC=AKEyXzWkLMiZXydGDv4Wod_Y8bQ1v3zNmRwQioLLRhNnxJsfC9v62eYKh_G6OlUbfjzom1BFAGk
Origin:	https://www.google.com.ua
Priority:	u=1, i
Referer:	https://www.google.com.ua/
Sec-Ch-Ua:	"Google Chrome";v="129", "Not=A?Brand";v="8", "Chromium";v="129"
Sec-Ch-Ua-Mobile:	?0
Sec-Ch-Ua-Platform:	"macOS"
Sec-Fetch-Dest:	empty
Sec-Fetch-Mode:	cors
Sec-Fetch-Site:	cross-site
User-Agent:	Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36
X-Client-Data:	CJS2yQEIpbbJAQipncoBCIf7ygEIkqHLAQjxossBCIagzQEIuMjNAQisns4BCMS2zgEI+MLOAQj+w84BCJPGzgEIvsfOAQioyM4BCLDIzgEYj87NARjDvc4B
	Decoded:
	message ClientVariations { // Active Google-visible variation IDs on this client. These are reported for analysis, but do not directly affect any server-side behavior. repeated int32 variation_id = [3300116, 3300133, 3313321, 3325319, 3330194, 3330417, 3362822, 3367992, 3378988, 3382084, 3383672, 3383806, 3384083, 3384254, 3384360, 3384368]; // Active Google-visible variation IDs on this client that trigger server-side behavior. These are reported for analysis *and* directly affect server-side behavior. repeated int32 trigger_variation_id = [3368719, 3382979];

- Рисунок 2 – Типовий HTTP запит до Google

Джерело: розроблено авторами

Такий заголовок включає 2854 символи, що є досить багато. В cookies багато даних, що стосуються безпеки і вони відправляються завжди.

Фахівці з комп'ютерних мереж намагаються уникати повторюваності, тому стиснення заголовків стало ключовим компонентом HTTP 2 і було запроваджено навіть у його попередника SPDY [5]. Хоча процес стиснення і розпакування даних вимагає часу та обчислювальних ресурсів, ці витрати є відносно невеликими порівняно з часом, потрібним для передачі повнорозмірних даних по мережі. З огляду на те, що HTTPS вимагає додаткових ресурсів для шифрування, спочатку стискати дані, а потім шифрувати їх є ефективнішим рішенням для зменшення обсягу переданої інформації [6].

Стиснення даних може бути з втратою або без втрат. У стисненні з втратою частина деталей відкидається, оскільки вони не критично важливі. Таке стиснення часто використовується для медіа файлів, які можна стискати не втрачаючи загального сенсу даних. Однак надмірне стиснення призводить до втрати якості, наприклад, зображення стає неможливо збільшити без втрати чіткості [7]. Отже, стиснення з втратою – це завжди баланс між розміром даних і якістю.

Оскільки HTTP заголовки містять важливу інформацію, навіть якщо вона часто повторюється, стиснення з втратою не є варіантом, хоч воно зазвичай ефективніше. Стиснення без втрат працює шляхом видалення повторюваних елементів, які легко відновити при розпаковуванні даних. Існує три способи реалізації такого стиснення:

- Таблиці пошуку
- Ефективніші методи кодування
- Компресія з посиланням на попередні дані

Метод таблиць пошуку полягає в заміні довгих, повторюваних фрагментів даних посиланнями. Для розпакування ці посилання замінюються початковим текстом із таблиці пошуку. Цей метод може використовуватися динамічно та особливо ефективний для структурованих даних [8]. Наприклад, звернемо увагу на рисунок 1, його можна стиснути використовуючи статичну таблицю пошуку.

Таблиця 1 – Стиснуті дані за допомогою методу таблиць пошуку

:authority:	$ 1
:method:	$ 2
:path:	$ 3
:scheme:	$ 4
Accept:	$ 5
Accept-Encoding:	$ 6
Accept-Language:	$ 7
Origin:	$ 8
Priority:	$ 9
Referer:	$ 10
Sec-Ch-Ua:	$ 11
Sec-Ch-Ua-Mobile:	$ 12
Sec-Ch-Ua-Platform:	$ 13
Sec-Fetch-Dest:	$ 14
Sec-Fetch-Mode:	$ 15
Sec-Fetch-Site:	$ 16
User-Agent:	$ 17

Джерело: розроблено авторами

Статичні таблиці не змінюються під час з'єднання, на відміну від динамічних таблиць, які оновлюються в реальному часі. Використовуючи створену таблицю 1 стиснутий варіант запиту матиме наступний вигляд (рис.3).

```
$1 Static.xx.fbcdn.net
$2 GET
$3. Jetmanifest/1017706399/facebook/main
$4 httos
$5 a
$6 gzip, deflate,
$7 en-GB,en-US;q=0.9,
$8 https://www.facebook.com
$9 u=1
$10 https://www.facebook.com/
$11 "Google Chrome";v="129"
$12 0?
$13 "macOS"
$14 empty
$15 cors
$16 cross-site
$17 Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/129.0.0.0 Safari/537.36
$18 "Not=A?Brand";v="8", "Chromium";v="129"
```

Рисунок 3 – Типовий HTTP запит до Facebook

Джерело: розроблено авторами

Цей запит займає 455 символів, що на 26% менше початкового обсягу. Однак таблиця пошуку може бути додана до стисненої версії. У такому випадку загальний розмір може зрівнятися з оригіналом, що зведе економію нанівець.

Таблиці пошуку корисні лише при частому повторенні значень. У цьому простому прикладі можна використовувати узгоджену статичну таблицю для популярних HTTP заголовків.

Різні методи стиснення базуються на принципі, що дані можуть бути представлені компактніше за допомогою спеціалізованого підходу. Для тексту використовуються різні формати кодування: ASCII, UTF-8 чи UTF-16. Ми знаємо, що кожен символ зберігається у вигляді послідовності з 0 та 1 і займає 8 біт, як у випадку ASCII. Це називається кодуванням фіксованої довжини, оскільки кожен символ використовує однакову фіксовану кількість бітів для зберігання.

Одним із методів кодування змінної довжини, де символи кодуються бітами, залежно від частоти їх використання – є кодування Хаффмана. Основна ідея полягає в кодуванні змінної довжини, де символам присвоюється різна кількість бітів залежно від частоти їх появи. Часті символи кодуються коротшими кодами, тоді як рідкісні – довшими. Таке кодування зменшує загальну кількість бітів для тексту. В англійській мові літера *E* є найпоширенішою, за нею йдуть *T* і *A*,найрідше використовуються: *X, J, Q* та *Z*.

Однак виникає проблема у декодуванні: потрібно точно ідентифікувати межі символів, щоб правильно відновити початкову послідовність із бітового потоку. Щоб уникнути цієї неоднозначності, ми повинні гарантувати, що наше кодування задовольняє таке поняття, як префіксне правило, яке у свою чергу передбачає, що коди можна декодувати лише одним унікальним способом. Префіксне правило гарантує, що жоден код не буде префіксом іншого.

Алгоритм Хаффмана призначений для ефективного кодування даних, зменшуючи їх початковий розмір. Він працює за допомогою бінарного дерева, де кожному символу присвоюється унікальний код, залежно від частоти його появи: частіші символи отримують коротші коди, а рідкісні – довші. Для побудови дерева алгоритм спочатку сортує символи за частотою, об'єднує їх у вузли, а потім рекурсивно формує коди для кожного символу. Це дозволяє оптимально стискати дані, мінімізуючи обсяг пам'яті. Основна ідея методу полягає в заміні вихідних даних на більш ефективні коди. Кожному унікальному значенню присвоюється бінарний код, причому коротші

коди отримують значення, які зустрічаються частіше. Ці відповідності зберігаються в таблиці перекодування, яка завантажується в програму декодування перед обробкою самих кодів [9].

Можна взяти до прикладу запит на рисунку 1, проаналізувавши лише символи *"a", "b", "c", "d"* и *“e", “f”*, частота їх появи дорівнюють 28, 10, 36, 8, 47 і 12 відповідно, зображено на рисунку 4

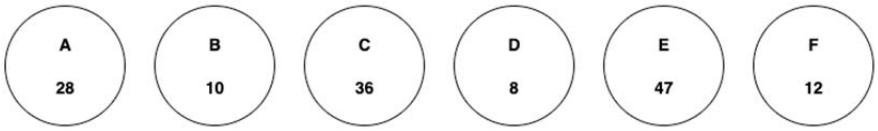

Рисунок 4 – Частота повторень символів у тексті
Джерело: розроблено авторами на підставі [10]

Необхідно сортувати символи за частотою їх появи, рисунок 5.

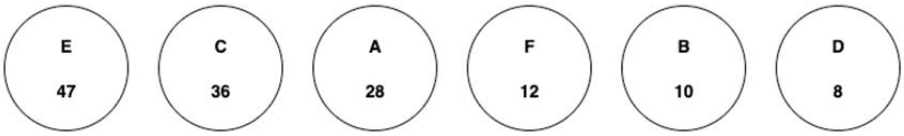

Рисунок 5 – Відсортований масив символів за повторенням їх у тексті
Джерело: розроблено авторами на підставі [10]

Основний алгоритм об'єднує разом всі елементи, що з'являються як найрідше, потім пара розглядається як один елемент і їх частоти об'єднуються. Це повторюється до тих пір, поки всі елементи не об'єднаються в пари (рис.6).

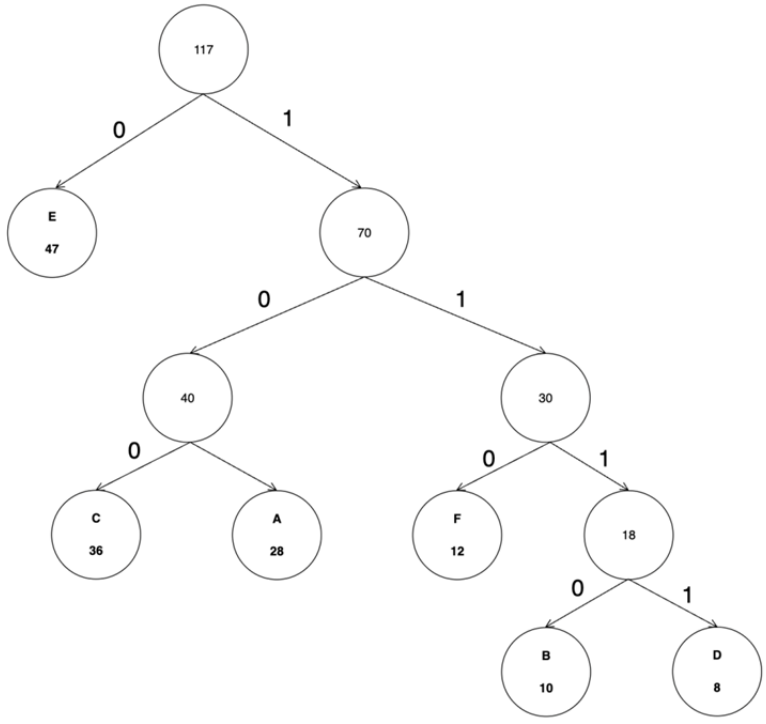

Рисунок 6 – Розкладений граф за методом Хаффмана
Джерело: розроблено авторами на підставі [10]

Можна заповнити утворені значення Хаффмана в таблиці 2 за допомогою рисунку6.

Таблиця 2 – Таблиця, побудована на основі утвореного графу Хаффмана

Символ	Значення Хаффмана	Бінарне значення
A	101	01000001
B	1110	01000010
C	100	01000011
D	1111	01000100
E	0	01000101
F	110	01000110

Джерело: розроблено авторами

Можна закодувати слово "cafe". Використовуючи звичайний ASCII (4 x 7 біт = 28 біт) або кодування Хаффмана (3 + 3 + 3 + 1 = 10 біт). Кодування Хаффмана займає менше місця навіть у такому простому випадку. При більшому обсязі тексту економія буде помітна краще [10].

Після атаки CRIME виявилась уразливість у методах стиснення заголовків, HTTP/2 потребував нові підходи достиснення, які б були захищені від подібних атак. CRIME використовує алгоритми стиснення даних у SSL для доступу до конфіденційної інформації, наприклад, до cookies, і може дозволити зловмисникам несанкціонований доступ до системи. Цей підхід змінює розміри стиснених повідомлень [11]. Стиснення здійснюється до шифрування, що дозволяє зловмисникам можливість додавати дані cookie до повідомлень і таким чином отримувати доступ до зашифрованих даних [12].

CRIME становить серйозну загрозу, особливо для компаній та користувачів, які використовують хмарні технології для зберігання важливих даних. Зловмисники можуть використати цю вразливість для перехоплення конфіденційної інформації. Команда HTTP розробила специфікацію HPACK, яка базується на таблицях пошуку і кодуванні Хаффмана, але не використовує стиснення із зворотним переглядом (lookback-based compression), що робить його більш безпечним.

HPACK містить статичну таблицю з 61 найпоширенішою назвою HTTP заголовків. Ця таблиця використовується як для запитів, так і для відповідей сервера, дозволяючи ефективно стискати часто вживані назви HTTP заголовків, а також деякі поширені ключ пари.

Наприклад, заголовок method: DELETE відсутній у таблиці, але його можна стиснути, посилаючись на ключ значення (method: GET) для назви заголовка і закодоване значення DELETE. Іншими словами, записи таблиці для ключ значення можуть використовуватися тільки для назви наприклад, method. Однак у зворотному порядку це неможливо – немає можливості скористатися значенням, пов’язаним з іншим заголовком. Наприклад, заголовок header1: GET не може скористатися значенням GET з method: GET [13].

Крім статичної таблиці, HPACK використовує динамічну таблицю на рівні з'єднання, яка починається з позиції 62, тобто після статичної таблиці, і може досягати максимальної величини, визначеної параметром SETTINGS_HEADER_TABLE_SIZE у налаштуваннях [14]. За замовчуванням ця величина становить 4096 октетів, якщо не вказана інша. Коли максимальний розмір таблиці досягається, найстаріший запис видаляється. Щоб полегшити цей процес, кожен запис зміщується при додаванні нових заголовків.

У деяких випадках кодування Хаффмана може призводити до більшого розміру значень, ніж якби було використано звичайний ASCII. Якщо ви вирішите закодувати, наприклад, слово "delete" в ASCII, ви можете відразу перейти до шістнадцяткового формату, оскільки кожен код ASCII займає 1 октет. У цьому випадку немає різниці між використанням кодування Хаффмана чи його відсутністю – обидва варіанти займають по 7 октетів. Хоча всі коди Хаффмана в цьому прикладі складаються з 7 бітів, для повноти октетів потрібне додаткове заповнення, що зрештою дає однаковий розмір.

У деяких інших заголовках ASCII кодування може виявитися компактнішим, наприклад, якщо у заголовку використовуються рідкісні символи з таблиці Хаффмана, довжина кодування яких перевищує 8 бітів. Через це специфікація HPACK дозволяє клієнту вибирати, застосовувати кодування Хаффмана чи ні, і змінювати його для кожного заголовка, залежно від того, який метод дозволяє використовувати найменшу кількість октетів.

Загалом, кодування Хаффмана часто є ефективнішим, ніж ASCII. Це зумовлено тим, що хоча ASCII потребує лише 7 бітів, для кожного значення використовується повний 8-бітний октет, тож 1 біт витрачається даремно. Кодування Хаффмана допускає змінну довжину коду, що дозволяє уникнути втрати бітів. Проте у цьому випадку не часто символи можуть мати коди довші за 8 бітів, і в таких випадках кодування ASCII може бути ефективнішим. Однак ці символи мають бути рідковживаними, якщо таблиця Хаффмана для HPACK відображає реальне використання. Зрештою, пошук у статичних або динамічних таблицях завжди буде ефективнішим, ніж кодування як у форматі Хаффмана, так і в ASCII.

Cookies- це невеликі файли, які вебсайти зберігають на комп'ютері користувача під час відвідування. Вони містять інформацію, яка дозволяє сайту запам'ятовувати дії, налаштування або інші дані користувача між сесіями. Наприклад, cookie можуть зберігати дані про логін, уподобання, товари у кошику, щоб користувач не втрачав їх між відвідуваннями сайту.

Cookies містять чутливі дані і здаються ідеально підходять під метод Хаффмана. та часто мають великий розмір і повторюються, тому бажано, щоб вони стискалися.

Але на відміну від часткового стиснення із зворотним переглядом, у HPACK весь вміст cookie потрібно вгадати. Це зменшує ефективність від стиснення такого типу даних.

Деякі браузери як Firefox не використовують індексування для невеликих cookies, які менші за 20 байтів, аргументуючи це тим, що якщо cookie маленькі, то навіть за високої частоти повторень зловмисникам буде складніше вгадати їх вміст, тому вигода від стиснення тут обґрунтована. Для великих cookie браузер індексує значення, щоб його можна було використовувати в подальших запитах. Chrome завжди використовує індексування незалежно від довжини cookies [15].

Висновки. Враховуючи необхідність оптимізації швидкості передачі даних в мережі і зростання складності сучасних веб сервісів, ефективність стиснення заголовків є одним із напрямків розвитку. Стиснення заголовків у HTTP 2 за допомогою HPACK суттєво зменшує обсяг переданих даних, що підвищує ефективність мережевих запитів. HPACK використовує статичні і динамічні таблиці заголовків разом з кодуванням Хаффмана, яке дозволяє зберігати заголовки у вигляді бітових послідовностей змінної довжини, зокрема для елементів заголовків, що часто використовуються. Це дає змогу знижувати обсяг даних при повторних запитах, оптимізуючи навантаження на мережу за рахунок меншого об’єму інформації, що передається.

Застосування кодування Хаффмана в HPACK може бути не завжди оптимальним, зокрема для великих, чи рідко використовуваних частин заголовків, використання стандартного ASCII кодування може давати кращі результати. У більшості випадків, кодування Хаффмана дає змогу значно зменшити обсяг даних, що будуть передаватися. Однак, аналіз конкретної системи та дослідження специфіки й статистики запитів можуть бути аргументом для вибору іншого варіанту стиснення даних, що буде більш оптимальним для даної конкретної системи.

Безпека при передачі даних, особливо таких як cookies, залишається важливим аспектом HPACK. Хоча cookies можуть мати великий розмір і повторюються в кожному запиті, стиснення таких даних потребує особливої уваги. Зокрема необхідно враховувати, що стиснення cookies, які мають менше 20 біт розміру не є обов'язковим. Загалом для стиснення даних, що мають невеликий розмір, бажано не використовувати метод Хаффмана. Виграш в швидкості передачі для даних малого об'єму не компенсує витрат часу на дешифрування заголовку.

HPACK забезпечує оптимальне стиснення заголовків у HTTP 2, знижуючи навантаження на мережу та покращуючи швидкість обробки запитів. Однак, досягнення балансу між ефективністю стиснення та безпекою залишається актуальним питанням для його подальшого вдосконалення, оскільки технології розвиваються. Зловмисники шукають нові підходи і прогалини у мережі. Тому сучасні веб-браузери повинні постійно оновлюватися, щоб запроваджувати найновіші методи стиснення заголовків та захисту даних.

Одним із перспективних напрямків є використання гібридних методів стиснення, а саме поєднання Хаффмана з такими як Run-Length Encoding (RLE) або LZ77, такий підхід може дати кращі результати при обробці специфічних типів даних. Також можна запропонувати використання штучного інтелекту для аналізу мережевого трафіку й визначення ефективних варіантів стратегій для конкретних випадків. Використання різних підходів до стиснення заголовків і їх декомпресії з урахуванням конкретних характеристик клієнта та сервера може значно покращити продуктивність у мережах із високими затримками.

Список літератури

1. Singh V. Evolution of internet protocols: episode 573. *Software engineering radio*. URL: https://podcasts.apple.com/ua/podcast/software-engineering-radio-the-podcast-for/id120906714?i=1000621654154 (дата звернення: 01.10.2023).
2. Кравчук О. Аспекти переходу на HTTP/2. *Вісник Хмельницького національного університету*. 2016. № 5. С. 221.
3. Peon R., Ruellan H. HPACK: Header compression for HTTP/2 (No. RFC 7541). URL: https://www.rfc-editor.org/rfc/rfc7541.html (дата звернення: 14.11.2024).
4. Перевозніков С. І., Горобець Ю. В. Порівняння часу завантаження сторінки за протоколами HTTP1 та HTTP2. *Матеріали XII Міжнародної науково-практичної конференції «Інтернет-освіта-наука» (IES-2020)*, Україна, Вінниця, 26-29 травня 2020 р. Вінниця, 2020. С. 88-91.
5. Дрововозов В. І., Хемраєв А. К. Аналіз надлишковості протоколів стека TCP/IP. *Інформатизація та управління*. 2020. № 26.
6. Kerschbaumer C., Gaibler J., Edelstein A., van der Merwey T. HTTPS-Only: Upgrading all connections to HTTPS in web browsers . *Workshop on measurements, attacks, and defenses for the web*. 2021. URL: https://doi.org/10.14722/madweb.2021.23010 (дата звернення: 14.11.2024).
7. Чобану В. В. Дослідження та розроблення системи вибору оптимального алгоритму стиснення даних при резервному копіюванні. *Інформаційні технології та системи*. 2021. № 7.

8. Khataei A., Bazargan K. CompressedLUT: An Open Source Tool for Lossless Compression of Lookup Tables for Function Evaluation and Beyond. *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays*. 2024. С. 2-11.
9. Moffat A. Huffman coding *ACM Computing Surveys (CSUR)*. 2019. Т. 52, № 4. С. 1-35.
10. Методи стиску зображень / Кафедра програмного забезпечення Дніпровського державного технічного університету. URL: https://pzs.dstu.dp.ua/ComputerGraphics/ic/index.html (дата звернення: 14.11.2024).
11. Ivanov O., Ruzhentsev V., Oliynykov R. Comparison of modern network attacks on TLS protocol *2018 International Scientific-Practical Conference Problems of Infocommunications. Science and Technology (PIC S&T)*. 2018. С. 565-570.
12. Parveen K., Fatima N. Cookie Hijacking: Privacy Risk *International Journal for Electronic Crime Investigation*. 2023. Т. 7, № 4. С. 61-72.
13. Beckett D., Sezer S. HTTP/2 tsunami: Investigating HTTP/2 proxy amplification DDoS attacks *2017 Seventh International Conference on Emerging Security Technologies (EST)*. 2017. С. 128-133.
14. Jiang M., Luo X., Miu T., Hu S., Rao W. Are HTTP/2 servers ready yet? *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. 2017. С. 1661-1671.
15. Kontaxis G., Chew M. Tracking protection in Firefox for privacy and performance. arXiv:1506.04104. URL: https://arxiv.org/abs/1506.04104 (дата звернення: 14.11.2024).

References

1. Singh, V. (2023, October 1). Evolution of internet protocols: episode 573. *Software Engineering Radio*. https://podcasts.apple.com/ua/podcast/software-engineering-radio-the-podcast-for/id120906714?i=1000621654154
2. Kravchuk, O. (2016). Aspects of transition to HTTP/2. *Visnyk Khmelnytskoho Natsionalnoho Universytetu*, (5), 221. [in Ukrainian]
3. Peon, R., & Ruellan, H. (2015). HPACK: Header compression for HTTP/2 (No. RFC 7541). https://www.rfc-editor.org/rfc/rfc7541.html
4. Perevoznikov, S. I., & Horobets, Y. V. (2020). Comparison of page loading time using HTTP1 and HTTP2 protocols. In *Proceedings of the XII International Scientific-Practical Conference "Internet-Education-Science" (IES-2020)*, Vinnytsia, Ukraine (pp. 88-91). VNTU. [in Ukrainian]
5. Drovovozov, V. I., & Khemraiev, A. K. (2020). Analysis of redundancy in TCP/IP protocols stack. *Informatyzatsiia ta upravlinnia*, (26). [in Ukrainian]
6. Kerschbaumer, C., Gaibler, J., Edelstein, A., & van der Merwey, T. (2021). HTTPS-only: Upgrading all connections to HTTPS in web browsers. In *Workshop on Measurements, Attacks, and Defenses for the Web*. Internet Society. https://doi.org/10.14722/madweb.2021.23010
7. Chobanu, V. V. (2021). Research and development of a system for selecting the optimal data compression algorithm for backup. *Informatsiini Tekhnolohii ta Systemy*, (7). [in Ukrainian]
8. Khataei, A., & Bazargan, K. (2024, April). CompressedLUT: An open-source tool for lossless compression of lookup tables for function evaluation and beyond. *Proceedings of the 2024 ACM/SIGDA International Symposium on Field Programmable Gate Arrays* (pp. 2-11).
9. Moffat, A. (2019). Huffman coding. *ACM Computing Surveys (CSUR)*, 52(4), 1-35. https://doi.org/10.1145/3338521
10. Methods of image compression. (n.d.). Kafedra programnoho zabezpechennia Dniprovskoho derzhavnoho tekhnichnoho universytetu. https://pzs.dstu.dp.ua/ComputerGraphics/ic/index.html [in Ukrainian]
11. Ivanov, O., Ruzhentsev, V., & Oliynykov, R. (2018, October). Comparison of modern network attacks on TLS protocol. In *2018 International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T)* (pp. 565-570). IEEE. https://doi.org/10.1109/PICST.2018.8632037
12. Parveen, K., & Fatima, N. (2023). Cookie hijacking: Privacy risk. *International Journal for Electronic Crime Investigation*, 7(4), 61-72. https://doi.org/10.1016/j.ijeci.2023.03.002
13. Beckett, D., & Sezer, S. (2017, September). HTTP/2 tsunami: Investigating HTTP/2 proxy amplification DDoS attacks. In *2017 Seventh International Conference on Emerging Security Technologies (EST)* (pp. 128-133). IEEE. https://doi.org/10.1109/EST.2017.7890109
14. Jiang, M., Luo, X., Miu, T., Hu, S., & Rao, W. (2017, June). Are HTTP/2 servers ready yet? In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)* (pp. 1661-1671). IEEE. https://doi.org/10.1109/ICDCS.2017.192

15. Kontaxis, G., & Chew, M. (2015). Tracking protection in Firefox for privacy and performance. *arXiv preprint*. https://arxiv.org/abs/1506.04104

Olexandr Ulichev, PhD tech. sci.
Central Ukrainian National Technical University, Kropyvnytskyi, Ukraine
Olexandr Revniuk, post-graduate
Private Higher Education Establishment "European University", Kyiv, Ukraine

Comparison of HTTP 2 Header Compression Methods

The primary aim of this research is to analyze the key methods of header compression in the HTTP 2 protocol to optimize data transmission over the network. This study explores the prerequisites for developing the HPACK compression algorithm, assesses the potential threats and data loss risks associated with compression, and identifies strategies to minimize these risks. Additionally, the research focuses on the challenges of handling cookies in HTTP headers and determining optimal compression methods for reducing the volume of transmitted data, with practical examples from commercial code.

Introduced in 2015, HTTP 2 significantly improves web communication efficiency by utilizing multiplexing and header compression. Unlike HTTP 1.1, which involves repetitive transmission of headers for each request, HTTP 2 addresses this redundancy through HPACK, which uses static and dynamic header tables alongside Huffman coding. These techniques substantially reduce the data volume transmitted between client and server, lowering the load on network resources, especially in mobile environments with limited bandwidth. This paper examines how HPACK achieves data optimization by compressing frequently used headers while maintaining secure data transmission. It also explores various methods of implementing compression, including Huffman coding and lookup tables, to enhance efficiency in practical web applications. The findings highlight the trade-offs between compression efficiency and security, especially concerning the transmission of sensitive information like cookies.

The use of HPACK for header compression in HTTP 2 drastically reduces the volume of transmitted data, thereby improving network request efficiency. While Huffman coding is often effective, its benefits may not always outweigh those of traditional ASCII encoding, particularly for larger or less frequently used headers. The study concludes that effective compression strategies must consider both data optimization and security, especially when transmitting sensitive information such as cookies. By selecting the appropriate compression methods, it is possible to balance the efficiency of data transfer with the protection of confidential information. Future research should focus on enhancing HPACK's balance between compression and security as technologies evolve and potential vulnerabilities emerge.

HTTP2, header compression in HTTP2, HPACK, cookies, Huffman coding, HTTP requests

Одержано (Received) 18.11.2024 *Прорецензовано (Reviewed) 25.11.2024*
Прийнято до друку (Approved) 02.12.2024