РОЗДІЛ ІІ. ІНФОРМАЦІЙНО-КОМП'ЮТЕРНІ ТЕХНОЛОГІЇ

УДК 004.7
DOI: 10.25140/2411-5363-2019-1(15)-115-126

Роман Андрущенко

ПОРІВНЯЛЬНИЙ АНАЛІЗ ПОКАЗНИКІВ ЕФЕКТИВНОСТІ МЕТОДІВ СЕРІАЛІЗАЦІЇ ДАНИХ У КОМП'ЮТЕРНИХ МЕРЕЖАХ

Актуальність теми дослідження. *У наш час спостерігається значне збільшення трафіку в мережі Інтернет та в локальних мережах. Тому необхідно приділяти увагу ефективності процесів прийому/передачі даних у мережевому програмному забезпеченні. Програмне забезпечення, яке тим чи іншим чином передає дані по комп'ютерній мережі, виконує операції серіалізації/десеріалізації внутрішніх структур у потік байтів. Ці процеси є необхідними і можуть безпосередньо впливати на механізми комунікації між хостами. У статті проведено порівняльний аналіз форматів серіалізації структурованих даних, та досліджено вплив компресії на прикладному рівні моделі OSI. Проведене дослідження дасть змогу покращити процеси передачі даних у межах моделі OSI, беручи до уваги високорівневу структуру даних, що передаються.*

Постановка проблеми. *Процеси перетворення внутрішніх структур даних програмного забезпечення до вигляду, прийнятного для передачі через мережу, можуть впливати на швидкість та надійність роботи програмного забезпечення. Проблема полягає в тому, що сучасне програмне забезпечення виконує значну, а іноді й зайву, роботу при передачі даних, передаючи надлишкову інформацію. Також протоколи різних рівнів моделі OSI зазвичай не беруть до уваги особливості самих даних, розглядаючи їх лише як потік байтів, що призводить до менш ефективних результатів. Одним із способів покращення цієї ситуації є виокремлення структурних особливостей даних, що передаються, та аналіз того, як вони впливають на процес серіалізації та десеріалізації.*

Аналіз останніх досліджень та публікацій. *Розглянуто публікації, матеріали конференцій у сфері інформаційних технологій за темою дослідження, а також офіційну документацію популярних форматів даних та інтернет-стандарти RFC. Проведено аналіз наявних досліджень роботи протоколів прикладного рівня та форматів серіалізації даних.*

Виділення недосліджених частин загальної проблеми. *Дослідження впливу внутрішньої структури та формату даних, що передаються, на показники ефективності передачі даних з урахуванням стандартних методів компресії на прикладному рівні моделі OSI (GZIP).*

Постановка завдання. *Провести порівняльний аналіз показників роботи текстових та бінарних форматів серіалізації даних, дослідити ефективність їх роботи.*

Виклад основного матеріалу. *Проаналізована, протестована робота реалізацій текстових та бінарних форматів серіалізації даних на наборах повідомлень різного розміру та різної структури за допомогою експерименту.*

Висновки відповідно до статті. *У статті наведено результати порівняльного аналізу текстових та бінарних форматів серіалізації даних. Сформульовані переваги та недоліки використання стандартних механізмів компресії на прикладному рівні моделі OSI у комбінації з різними механізмами серіалізації.*

Ключові слова: *HTTP; серіалізація даних; компресія даних; комп'ютерна мережа; кодування; мережеве програмне забезпечення.*

Рис.: 10. Табл.: 1. Бібл.: 20.

Актуальність теми дослідження. Прикладне програмне забезпечення, яке тим чи іншим чином передає структуровані дані через комп'ютерні мережі, виконує такі операції: серіалізацію даних, безпосередню передачу по мережевому каналу, прийом даних та десеріалізацію. Ці етапи є необхідними при передачі будь-яких внутрішніх структур даних і від них безпосередньо залежить швидкість і надійність роботи програмного забезпечення. Тому у зв'язку зі значним збільшенням трафіку в мережі Інтернет та локальних мережах необхідно приділяти увагу ефективності процесів прийому/передачі даних у мережевому програмному забезпеченні [1; 2].

Постановка проблеми. Прикладне програмне забезпечення працює на вищих рівнях моделі OSI для передачі даних через комп'ютерні мережі: від транспортного рівня до прикладного. Процеси перетворення внутрішніх структур даних програмного забезпечення до вигляду, прийнятного для передачі через мережу, впливають на швидкість та надійність роботи програмного забезпечення. Існує багато способів серіалізації/десеріалізації даних, і кожен із них має як свої переваги, так і недоліки. Проблема полягає в тому, що більшість працюючого програмного забезпечення виконує значну, а іноді й зайву, роботу при передачі даних, передаючи надлишкову інформацію, яка і так відома обом сторонам передачі – як передавачу, так і приймачеві.

Особливо це стосується найпопулярніших форматів серіалізації JSON, XML та найпопулярнішого текстового протоколу прикладного рівня – HTTP. У цій статті розглянуто та проаналізовано роботу і продуктивність 9 різних механізмів серіалізації в комбінації з компресією GZIP, яка є де-факто стандартом у протоколі HTTP.

Аналіз останніх досліджень і публікацій. Найпопулярніші формати серіалізації даних (наприклад: XML, JSON) мають декілька реалізацій. Кожна з реалізацій має як свої переваги, так і недоліки. Зокрема, дослідження різних реалізацій проведено в статті [3]. У статті вказано, що бібліотечні реалізації, які використовують автоматичний парсинг програмних об'єктів і структур даних, потребують додаткових ресурсів на обробку даних, що негативно впливає на швидкість їхньої роботи.

У статі [4] порівнюється ефективність JSON, XML, Thrift та Protobuf на мобільних пристроях, що актуально у зв'язку зі збільшенням популярності мобільних платформ та операційних систем iOS та Android. Наголошується, що мобільні пристрої мають свої особливості, пов'язані зі значно меншою кількістю доступних ресурсів.

Залежність часу, необхідного для серіалізації та десеріалізації даних у файл, від типу даних та їх об'єму, досліджується у статті [5] для форматів JSON, YAML. Також у цій же статі розглянуто стандартні механізми серіалізації Java та C#: JDK Serialization, Object Serializaton in .NET Framework. Зазначено, що не рекомендується використовувати стандартні механізми серіалізації.

У статті [6] запропоновано спосіб оптимізації серіалізації даних, який показує кращі результати (як за необхідним часом, так і за розміром вихідної структури), аніж JSON, XML та MessagePack.

У роботі [7] розглянуто використання Protocol Buffers у сфері мікроблогингу, де аналізується робота Protocol Buffers при передачі великої кількості невеликих за розміром повідомлень. У статті [8] проведено дослідження концепцій збереження цілісності даних у текстових форматах JSON та XML.

Розглянуто офіційні документи-специфікації форматів серіалізації даних: JSON, XML, BSON, Smile, MessagePack, Protocol Buffers, Flat Buffers, Apache Thrift [9-16].

У цій роботі проаналізовано швидкість та результати роботи таких форматів: XML, JSON, BSON, MessagePack, Smile, Protocol Buffers, Flat Buffers, Apache Thrift.

XML – Extensible Markup Language. Формат описує клас об'єктів даних, що називаються XML-документами, і частково описує поведінку комп'ютерних програм, які їх обробляють [9].

JSON – JavaScript Object Notation. Це текстовий незалежний формат, який використовує конвенції мов програмування сімейства C. JSON побудований на двох структурах: колекції пар «назва/значення» та упорядковані списки значень, що реалізуються як масив, вектор, список або послідовність. [10]

BSON – Binary JSON. Як і JSON, BSON підтримує багаторівневі структури документів та масивів. BSON можна порівнювати з бінарними форматами обміну, наприклад, Protocol Buffers. На відміну від останнього, він більш гнучкий. Однак BSON має накладні витрати, адже схема даних передається разом із самими даними [11].

Smile – це бінарний формат даних, еквівалент стандартного формату даних JSON. Дані кодуються по секціях. Кожна секція складається з набору токенів, які формують відповідні ключі та значення [12].

MessagePack – механізм серіалізації/десеріалізації об'єктів та формат обміну даними, подібний до BSON та Smile. В MessagePack є своя система типів та присутні оптимізації для складних об'єктів та бінарних/текстових даних довільної довжини [13].

Protocol Buffers – нейтральний та платформо-незалежний спосіб серіалізації та передачі структурованих даних, розроблений Google. Кожне повідомлення в Protocol

Buffers є невеликим логічним записом інформації, що містить серію пар імя-значення. Схема даних задається у файлах спеціального формату *.proto. Protocol Buffers використовує подвійне кодування цілих чисел: ZigZag + VarInt [14].

Flat Buffers – формат та метод серіалізації даних, подібний до Protocol Buffers, але спеціально розроблений для систем із дійсно високим навантаженням. Його особливість полягає в оптимізованому використанні системних ресурсів – як часу на серіалізацію/десеріалізацію, так і на розмір використовуваної пам'яті. Flat Buffers надає доступ до даних без процесу серіалізації/десеріалізації. [15].

Варто зазначити, що в роботах [3; 5; 6] проаналізовано процес серіалізації внутрішніх структур даних на пристрої постійної пам'яті, тобто зберігання у файл. У цій же статті проаналізовано процес серіалізації для передачі даних через мережу.

Виділення недосліджених частин загальної проблеми. Дослідження впливу внутрішньої структури та формату даних, що передаються, на показники ефективності передачі даних з урахуванням стандартних методів компресії на прикладному рівні моделі OSI (компресія GZIP).

Постановка завдання. Метою цієї роботи є визначення, аналіз та порівняння швидкості та результатів роботи текстових та бінарних механізмів серіалізації/десеріалізації даних на масивах різного розміру та структури та з урахуванням стандартних алгоритмів компресії з метою прийому/передачі даних через комп'ютерні мережі.

Виклад основного матеріалу. Передача даних у комп'ютерних мережах у протоколах верхнього рівня (транспортний рівень моделі OSI та вище) у прикладному програмному забезпеченні виконується за допомогою абстракцій. В операційних системах для цього застосовуються такі абстракції, як сокети (Sockets). У прикладному ж ПЗ використовують таку абстракцію, як потоки даних (streams) або потоки вводу/виводу (input/output streams). Для зменшення навантаження на канал передачі даних застосовують алгоритми компресії. Компресія в цьому випадку – це додатковий етап, який не є обов'язковим, але який може бути застосований до будь-яких даних, незалежно від їх походження, протоколу, використовуваного формату серіалізації і т. ін. Однак, враховуючи особливість роботи з комп'ютерними мережами, а саме те, що дані приймаються та відправляються до каналу передачі як однонаправлений потік, то не всякі алгоритми компресії можуть бути застосовані, а тільки ті, які можуть працювати з однонаправленими потоками даних. Де-факто стандартом у мережі Інтернет є компресія даних у форматі GZIP, що використовує формат Deflate. Тому варто розглядати всі формати саме з урахуванням можливості їх поєднання з додатковим кодуванням GZIP (Deflate) [17].

Формат Deflate. Deflate – це формат компресії даних без втрат, який використовує комбінацію алгоритму LZ77 та кодування Хаффмана. Особливістю Deflate є те, що його можна застосовувати в умовах обмежених ресурсів проміжного буферу зберігання даних. Тобто навіть якщо на вході маємо потік розміром в 1 Гб та більше, Deflate дозволяє обробити цей потік послідовно в умовах ліміту ресурсів. Також Deflate може бути реалізованим способами, які не підпадають під патентні обмеження.

Код Хаффмана – це префіксний код, такий що жодне закодоване слово не є префіксом іншого закодованого слова. Символи, частота появи яких вища, кодуються меншим кодом і навпаки. Тобто для набору символів $a_1, a_2, a_3, \ldots, a_N$ алфавіту A, частота появи яких задається деякими значеннями з множини P $\{p_1, \ldots, p_N\}$ будується відсортована таблиця символів, така що $p_1 > p_2 > \ldots > p_N$, тобто в порядку зменшення частоти появи символів.

Далі будується дерево Хаффмана для кодування символів. Усі символи є листями дерева. Далі беруться два символи з найменшою частотою та об'єднуються у вузол $a_{N,N-1}$, значенням якого є сума частот об'єднаних символів $p = p_N + p_{N-1}$.

Після цього операція повторюється доти, поки всі символи не будуть об'єднані в дерево (рис. 1) [18].

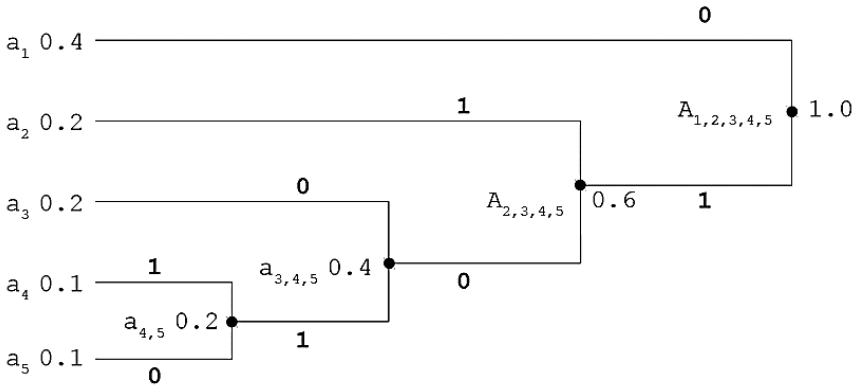

Рис. 1. Приклад дерева Хаффмана

Далі, спускаючись від кореня дерева, отримаємо такі коди для символів:

$$a_1 – 0,\ a_2 – 11,\ a_3 – 100,\qquad a_4 – 1011,\ a_5 – 1010.$$

Біти 0 та 1 можна назначати гілкам дерева в довільному порядку. Головна вимога – від кожного вузла має відходити 2 вітки – з «0» та з «1».

Варто зазначити, що побудова коду Хаффмана не є однозначною. Може існувати декілька способів сформувати дерево. З множини варіантів кодів Хаффмана кращим вважається той, у якого менша дисперсія. Дисперсія показує, як сильно відрізняються розміри кодів від середнього значення. Дисперсія розраховується таким чином:

$$D = \sum (c_N - \bar{c})^2; \tag{1}$$

$$\bar{c} = \sum \frac{c_N}{N}, \tag{2}$$

де $\bar{c}$ – середнє значення розміру закодованого символу;

c_N – розмір закодованого символу, у бітах;

N – загальна кількість символів.

Алгоритм LZ77. У Deflate кодування Хаффмана поєднується з алгоритмом LZ77, який дозволяє замінювати входження однакових послідовностей посиланнями на перше входження. LZ77 працює наступним чином:

1. Задається структура на потоці вхідних даних (рис. 2).

Словник (Dictionary) та випереджаючий буфер (Buffer) – складові частини вікна, де нові вхідні символи витісняють найдавніші. Розмір вікна – I та розмір буфера – J є фіксованими значеннями та задаються перед початком виконання алгоритму. Від цих значень залежить швидкість та ступінь компресії.

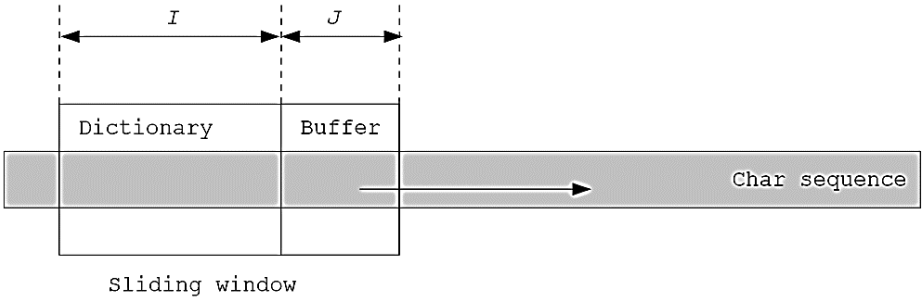

Рис. 2. Механізм потокового кодування LZ77

2. Послідовність T_j починаючи з позиції i кодується трійкою у формі $\langle d, l, s \rangle$, де:

d – дистанція, відносне зміщення до послідовності;

l – довжина послідовності;

s – символ.

Трійка значень $\langle d, l, s \rangle$ означає послідовність символів $T_j = T[\, i \ldots i+l \,] = T[\, i-d \ldots d+l]s$, де i – поточна позиція початку буфера.

Тобто послідовність $T[\, i \ldots i+l \,]$ довжиною l має інше входження в оригінальну послідовність символів, та знаходиться раніше на відстані в d символів від поточної позиції.

3. Значення d та l мають відповідати таким умовам: $d \in [1 \ldots I]$, $l \in [1 \ldots J]$, тобто відстань до послідовності не має перевищувати розміру вікна, а довжина не має бути більшою за розмір буфера.

4. Алгоритм LZ77 шукає найбільш можливу повторювану послідовність, тобто кодує підпослідовність із максимально можливою довжиною l в межах заданого буфера.

Отже, GZIP дозволяє зменшити розмір вхідних даних, використовуючи комбінацію кодування Хаффмана та алгоритм LZ77, причому він може ефективно працювати на послідовностях великого розміру та може бути застосованим у комбінації з будь-якими іншими форматами даних [19-20].

Програмний стек Apache Thrift. На відміну від інших форматів, перелічених в аналізі літератури, окремо необхідно розглянути Apache Thrift. Зазвичай на високих рівнях моделі OSI розділяють протоколи від форматів серіалізації. Програмний стек надає більш комплексний підхід. Він об'єднує протоколи передачі даних та формати серіалізації в одну систему, що дозволяє досягати кращих характеристик при передачі даних через комп'ютерну мережу. Це пояснюється тим, що формат серіалізації та протокол створюються з урахуванням особливостей один одного.

Apache Thrift – це легковісний та незалежний від мови програмування програмний стек із відповідним механізмом генерації коду для RPC. Thrift надає абстракції для передачі даних, їх серіалізації та обробки на рівні додатків.

На відміну від Protocol Buffers та Flat Buffers, Apache Thrift дозволяє не тільки описати схему даних, а й методи їх обробки – сервіси. Thrift підтримує різні мови програмування, включаючи C++, Java, Python, PHP та Ruby [16].

У цьому дослідженні інтерес становлять TBinaryProtocol та TCompactProtocol.

Варто окремо відмітити те, що з погляду реалізації, Apache Thrift є дуже зручним у використанні. Apache Thrift є гнучким рішенням, оскільки він із коробки надає різні модифікації протоколів передачі даних. При цьому він простіший у використанні, аніж Flat Buffers та Protocol Buffers, а результати кодування дають кращі результати, ніж Flat Buffers, та не набагато гірші, аніж Protocol Buffers [16].

Аналіз показників ефективності передачі даних різного об'єму

Дослідження ефективності розглянутих форматів кодування структурованих даних необхідно проводити на наборах різного розміру.

Для повної картини мають бути проаналізовані такі показники:

1) час, необхідний для серіалізації структур даних в in-memory бінарний потік;
2) результуючий розмір бінарного потоку після серіалізації.

Також усі ці показники необхідно перевірити як без додаткової компресії GZIP, так і з компресією GZIP. Швидкість передачі даних через мережевий канал залежить у цьому випадку лише від розміру серіалізованих даних.

Для генерації структурованих даних, схожих на реальні, використано інструмент Faker. Зокрема, за допомогою цього інструменту були згенеровані три структури різного розміру, що містять числові дані, об'єкти, масиви та текстові дані:

- data_tiny.bin (1 Кб у форматі Java Serialized);
- data_small.bin (10 Кб у форматі Java Serialized);
- data_medium.bin (100 Кб у форматі Java Serialized).

Заміри для кожного тесту повторювались 100 разів. З отриманих результатів відкидувались перші 10 тестів, які можна вважати «холодним запуском». Це потрібно для того, щоб уникнути спотворення результатів (рис. 3).

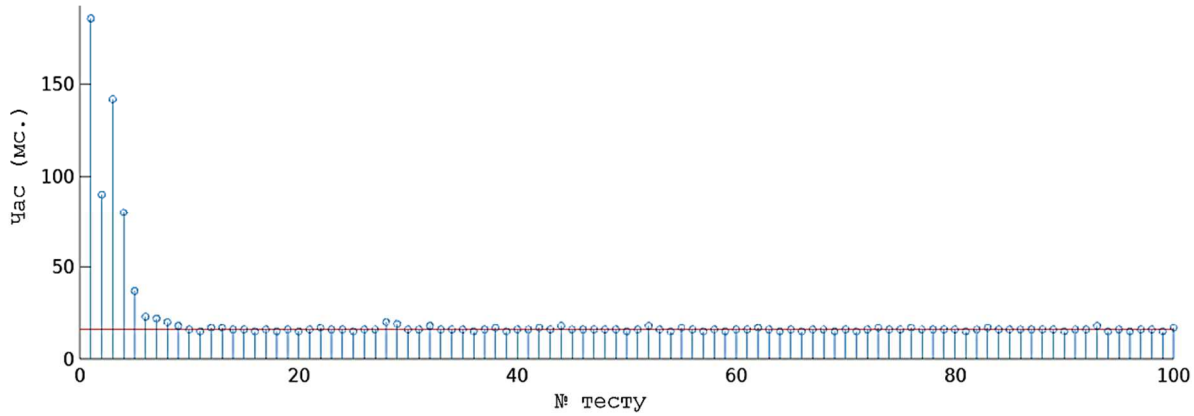

Рис. 3. Спотворення результатів першими запусками

Замір часу, необхідного на формування структур даних та їх серіалізацію в бінарних потік байтів, відбувався за допомогою спеціального буфера в оперативній пам'яті, який створювався безпосередньо перед кожним тестом. Також перед кожною ітерацією виконувався запит на запуск GC (Garbarge Collection), щоб уникнути переривання процесу аналізу та впливу на результат. Оскільки Garbarge Collection не є детермінованим, то був застосований метод перевірки на основі Weak References.

Кожен тест складався з декількох ітерацій, зокрема:

1) для повідомлень розміром 1 Кб у кожному тесті проводилось 5000 ітерацій
2) для повідомлень розміром 10 Кб у кожному тесті проводилось 1000 ітерацій
3) для повідомлень розміром 100 Кб у кожному тесті проводилось 100 ітерацій

На діаграмі (рис. 4) зображено порівняльні результати серіалізації малих повідомлень (1 Кб) без компресії. Взято значення медіани за результатами 100 тестів. Для більш точних результатів у кожному тесті серіалізація/десеріалізація даних виконувалась 5000 разів, оскільки розміри пакетів даних надто малі.

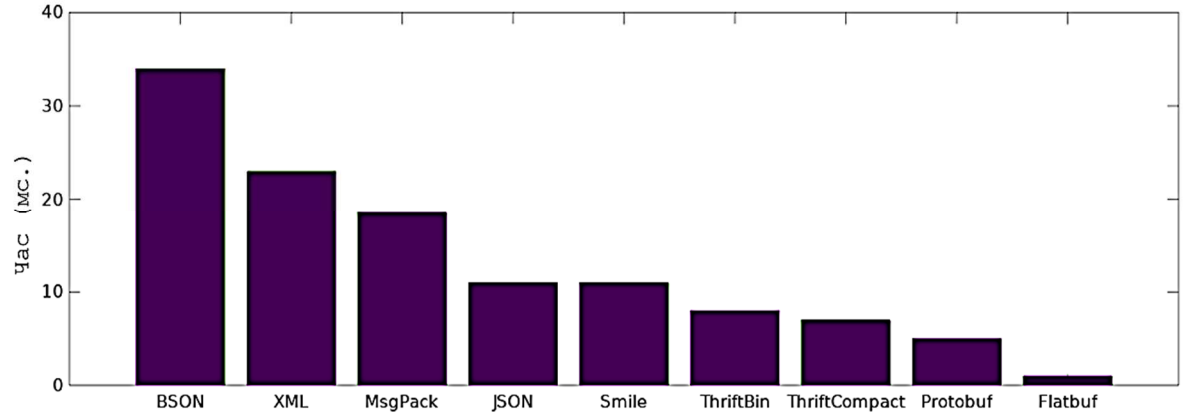

Рис. 4. Час серіалізації 5000 малих повідомлень по 1 Кб

Як можна бачити з вищенаведених рисунків, Flat Buffers майже миттєво працює з кодуванням/декодуванням даних. Це пояснюється тим, що Flat Buffers не трансформує дані взагалі ніяким чином. Внутрішнє представлення будь-яких структурованих даних є просто байтовий буфер. Декодування відбувається лише якщо є явний запит конкретних значень полів структури. Також цікавий результат дали BSON та XML. Виявляється, що незважаючи на бінарну природу BSON, працює він гірше, аніж JSON.

Далі порівнюється час серіалізації малих повідомлень із компресією GZIP.

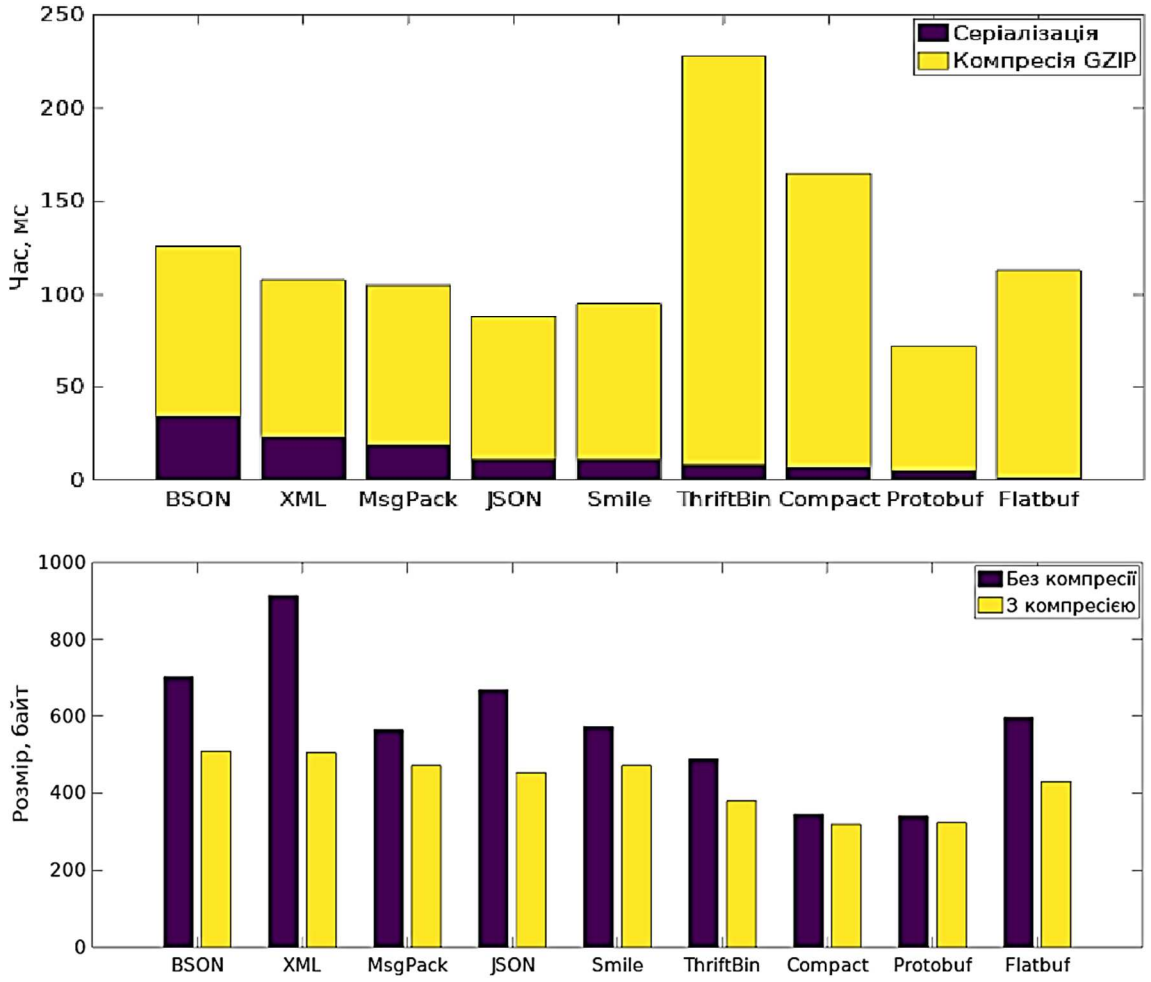

Рис. 5. Час серіалізації 5000 малих повідомлень по 1 Кб з компресією (зверху) та результуючий розмір серіалізованих повідомлень (знизу)

Apache Thrift як для протоколу TBinary, так і для протоколу TCompact показує значно гірші результати в разі застосування компресії GZIP. Це пов'язано з особливостями програмної реалізації протоколів Apache Thrift, зокрема через використання шарів абстракції над алгоритмами серіалізації та передачі даних.

Отже, загалом застосування компресії несе значний приріст часу на кодування/декодування повідомлень. Тому її необхідно застосовувати з обережністю.

Компресія ефективна для текстових форматів – XML, JSON, а також, як показав аналіз, BSON та Flat Buffers незважаючи на їхню бінарну природу, теж добре піддаються компресії, принаймні для повідомлень розміром в 1 Кб. Protocol Buffers та Thrift Compact майже не піддаються компресії.

Таким чином, можна припустити, що чим більша частка текстових даних, тим краще працює компресія GZIP. Для перевірки цієї гіпотези було створено просту структуру, розміром 10 Кб, яка складається з текстового масиву та масиву чисел, згенерованих інструментом Faker. Далі згенерована структура була серіалізована без GZIP та з GZIP, та розрахований ступінь компресії як:

$$R = \frac{N_{GZIP}}{N_{PLAIN}}. \tag{3}$$

Для зменшення впливу випадкового фактора, тест було повторено 50 разів, та взято середнє значення (рис. 6).

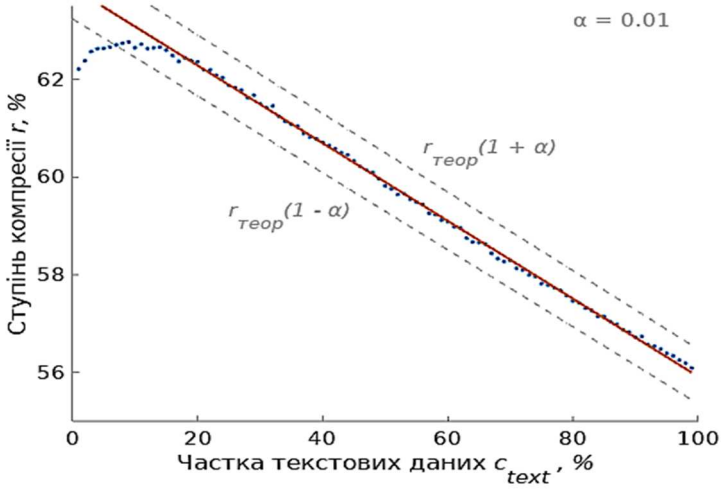

Рис. 6. Залежність ступеня компресії GZIP від частки текстових даних

Починаючи з частки текстових даних у 20 % залежність є лінійною. Під час аналізу масиву даних із відкритих ресурсів було виявлено (таблиця нижче), що частка текстових даних здебільшого вища за 20 %, тому можна допустити використання саме лінійного закону при реалізації форматів серіалізації даних із компресією GZIP.

Таблиця

Відкриті ресурси структурованих даних

Опис	Адреса в мережі Інтернет
Дані по астронавтах	http://api.open-notify.org
Дані по нобелівських лауреатах	http://api.nobelprize.org/v1
Історичні події по регіонах	http://www.vizgr.org/historical-events
Демографічні дані по країнах	http://api.population.io/1.0
Фінансові дані ринків та ВВП країн	http://api.worldbank.org
Інформація по телепрограмах	http://api.tvmaze.com
Бази даних Yahoo	https://query.yahooapis.com/v1
База даних НАСА	https://data.nasa.gov

Далі були проведені тести для структур даних розміром в 10 Кб. У кожній ітерації – по 1000 повідомлень. Нижче показані отримані результати:

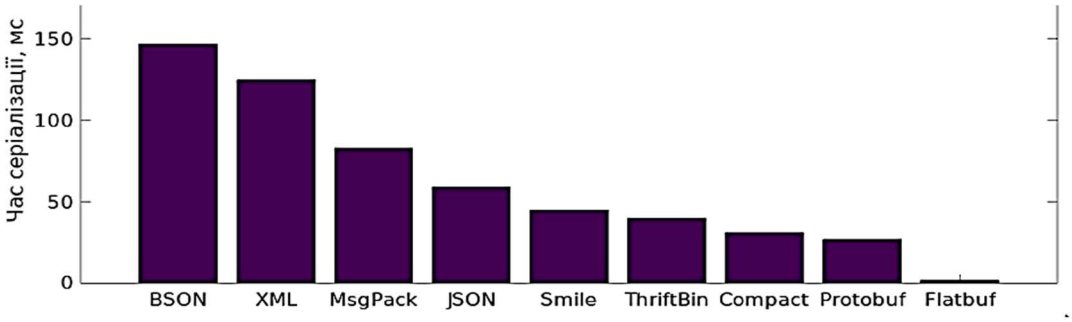

Рис. 7. Час серіалізації 1000 повідомлень, розміром по 10 Кб

Формат XML на даних розміром 10 Кб показує кращі результати, аніж на даних, розміром в 1 Кб, проте він однаково є менш ефективним щодо швидкості. З даних діаграм можна зробити висновок, що незважаючи на те, що BSON, MsgPack є бінарними форматами, швидкість їхньої роботи гірша, аніж у текстового формату JSON.

На рис. 8 зображено результати GZIP компресії повідомлень.

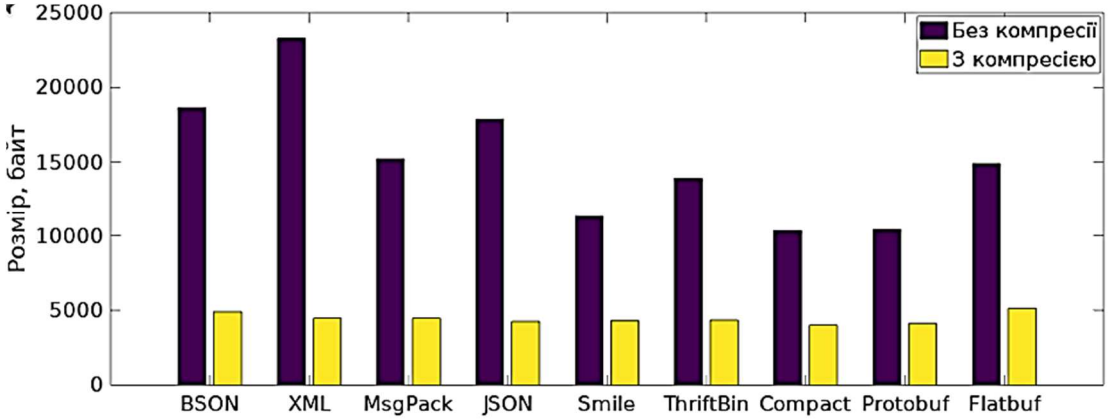

Рис. 8. Порівняння результуючого розміру повідомлень із компресії та без неї

Компресія GZIP майже повністю урівнює результуючі розміри повідомлень для всіх форматів даних. Тому якщо на першому місці за вимогами до інформаційної системи стоїть задача зменшити мережевий трафік, то гарним варіантом буде застосування компресії GZIP з текстовими простими форматами даних, такими як JSON або XML, оскільки з ними набагато легше працювати та відлагоджувати.

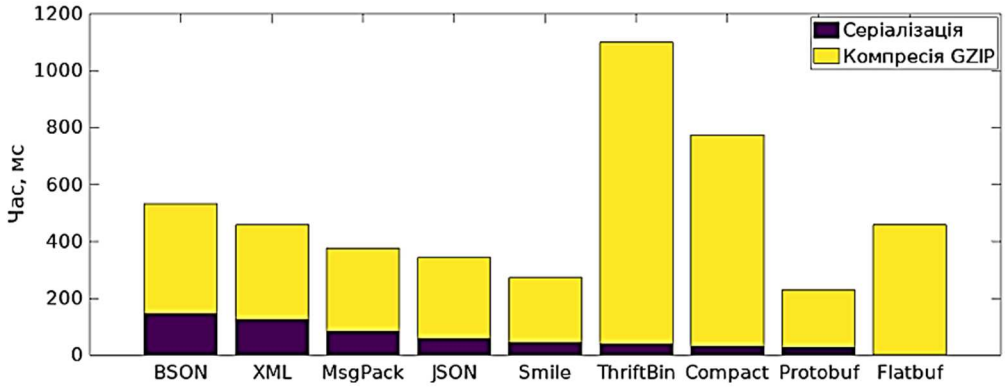

Рис. 9. Час серіалізації 1000 повідомлень по 10 Кб з GZIP компресією

Серіалізація повідомлень розміром 100 Кб (у кожній ітерації по 100 повідомлень):

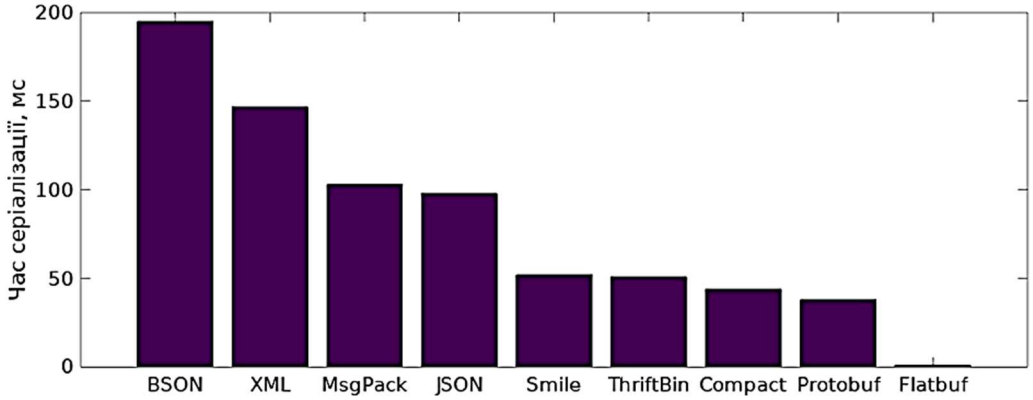

Рис. 10. Серіалізація 100 повідомлень по 100 Кб

Розмір серіалізованих повідомлень (з компресією та без неї) в цьому випадку має такий же самий вигляд, як на рис. 8, відрізняючись лише пропорціями. Тому при передачі структур даних великого розміру, різниця між результуючим розміром повідомлень, які передаються, для різних форматів стає менш помітною, як і у випадку з передачею повідомлень розміром від 10 Кб, якщо застосовувати потокову компресію.

Висновки відповідно до статті.

1. Прикладне програмне забезпечення зазвичай працює на прикладному рівні моделі OSI. Якщо на більш низьких рівнях дані розглядаються просто як масив або потік байтів із заголовком певного формату, то на прикладному рівні тісно розглядається зміст переданих даних. Крім протоколу передачі, розрізняють різні формати серіалізації та десеріалізації даних.

2. Залежно від вимог до програмного забезпечення при серіалізації/десеріалізації даних може використовуватись додаткова потокова компресія. Стандартним алгоритмом компресії нині в мережі Інтернет є GZIP, який поєднує в собі кодування Хаффмана та алгоритм LZ77.

3. Компресія значно збільшує час, необхідний на серіалізацію даних та створює додаткове навантаження на обчислювальні ресурси системи. Тому треба її застосовувати з обережністю. Компресія може дати значний виграш, якщо вона застосовується до текстових даних, або якщо масиви даних передаються в напрямку від високопродуктивного хосту до менш продуктивного (оскільки декомпресія менш вимоглива до ресурсів), або якщо канал передачі має низьку пропускну здатність.

4. Не всі бінарні формати серіалізації показують кращі характеристики, аніж текстові. Зокрема, з цього дослідження можна зробити висновок, що такими є формати BSON та MessagePack. Час, необхідний на серіалізацію та десеріалізацію у ці формати більший, аніж у JSON, незалежно від розміру структури. Розмір серіалізованих повідомлень у BSON в більшості випадків більший, ніж у JSON, а розмір серіалізованих повідомлень MessagePack зазвичай не нижче 80 % від розміру серіалізованих повідомлень у форматі JSON. При застосуванні компресії розміри повідомлень, серіалізованих у BSON, MessagePack та JSON майже не відрізняються. Різниця стає менш помітною зі збільшенням розміру повідомлення, яке серіалізується.

5. Програмний стек Apache Thrift значно знижує швидкість серіалізаці і десеріалізації даних при застосуванні потокової компресії, при цьому розмір серіалізованих повідомлень залишається майже без змін. Тому не рекомендується застосовувати алгоритми компресії у поєднанні з програмним стеком Thrift.

6. З іншого боку, Apache Thrift показує один із кращих результатів серед аналогів, як за швидкістю обробки даних, так і за розміром серіалізованих структур. При цьому він значно зручніший у використанні, аніж Protocol Buffers і Flat Buffers, та дозволяє вести розробку алгоритмів із меншими затратами часу та людських ресурсів.

7. З огляду на те, що текстові масиви даних значно краще піддаються компресії, пропонується підхід до створення бінарного протоколу з розділенням бінарних даних від текстових та частковою компресією лише текстових даних. Це дозволить як зменшити час, необхідний на компресію/декомпресію, так і зменшити розміри серіалізованих повідомлень.

8. Чим більший об’єм даних, до яких застосована компресія, тим менш помітна різниця між розміром результуючих повідомлень для різних форматів даних. Тому час, необхідний на передачу даних також буде майже однаковим, оскільки пропускна здатність каналу не залежить від внутрішньої структури повідомлень. Однак, з іншого боку, як уже було сказано у п. 3, при цьому значно збільшується час на серіалізацію. Тому для повної картини необхідно дослідити наявні моделі та застосувати їх для порівняння швидкості передачі даних, серіалізованих різними форматами, з урахуванням швидкодії апаратного забезпечення та пропускної здатності каналу.

Список використаних джерел

1. Cisco Visual Networking Index: Forecast and Methodology: 2016-2021 / Cisco Public, 2017.

2. Andrew M. Odlyzko. Internet traffic growth: Sources and implications. *Proceedings of SPIE: The International Society for Optical Engineering*, University of Minnesota, USA, 2003.

3. Kazuaki Maeda. Performance evaluation of object serialization libraries in XML, JSON and binary formats. *Second International Conference on Digital Information and Communication Technology and it's Applications* (DICTAP), 2012. P. 177–182.

4. S. Kami Makki. A comparison of data serialization formats for optimal efficiency on a mobile platform. *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, 2012. No. 48. DOI: 10.1145/2184751.2184810.

5. Rama Chawla. Object Serialization Formats and Techniques a review. *Global Journal of Computer Science and Technology Software*, Global Journals Inc., 2013. Vol. 13, Issue 6.

6. Carrera D., Gustavo A., Rosales J. Optimizing Binary Serialization with an Independent Data Definition Format /. *International Journal of Computer Applications*. 2018. Vol. 180. No. 28.

7. Canggih Puspo Wibowo. Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication. *International Conference on Informatics for Development, At Yogyakarta, Indonesia*, 2011. P. 40–43.

8. Gaurav Goyal, Karanjit Singh, Ramkumar K. R. A detailed analysis of data consistency concepts in data exchange formats (JSON & XML). *International Conference on Computing, Communication and Automation* (ICCCA). 2017. P. 72–77.

9. Extensible Markup Language P World Wide Web Consorcium. URL: https://www.w3.org/ XML.

10. ECMA-404 The JSON Data Interchange Standard, 2017. URL: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

11. BSON Spec. BSON (Binary JSON): Specification. URL: http://bsonspec.org/spec.html.

12. FasterXML. Smile Data Format. URL: https://github.com/FasterXML/smile-format-specification.

13. Sadayuki Furuhashi. MessagePack. URL: https://msgpack.org.

14. Google Developers. Protocol Buffers: Developer Guide. URL: https://developers.google.com/protocol-buffers/docs/overview.

15. Flat Buffers Internals. URL: https://google.github.io/flatbuffers/flatbuffers_internals.html.

16. Apache Software Foundation. Apache Thrift. URL: https://thrift.apache.org.

17. L. Peter Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951. URL: https://tools.ietf.org/html/rfc1951.

18. Mamta Sharma. Compression Using Huffman Coding. *IJCSNS International Journal of Computer Science and Network Security*. 2010. Vol. 10, No. 5. P. 133–140.

19. David Huffman. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*. 1952. Vol. 40, Issue 9, P. 1098–1952.

20. Ziv J., Lempel A. A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on information theory*. 1977. Vol. IT-23, No. 3. P. 337–343.

References

1. *Cisco Visual Networking Index: Forecast and Methodology*: *2016-2021*. Cisco Public, 2017.

2. Andrew M. Odlyzko (2003). Internet traffic growth: Sources and implications. *Proceedings of SPIE: The International Society for Optical Engineering*, University of Minnesota, USA.

3. Kazuaki Maeda (2012). Performance evaluation of object serialization libraries in XML, JSON and binary formats. *Second International Conference on Digital Information and Communication Technology and it's Applications* (DICTAP) (pp. 177–182).

4. S. Kami Makki (2012). A comparison of data serialization formats for optimal efficiency on a mobile platform. *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication*, 48. DOI: 10.1145/2184751.2184810.

5. Rama Chawla (2013). Object Serialization Formats and Techniques a review. *Global Journal of Computer Science and Technology Software, Global Journals Inc*., 13 (6).

6. David Carrera (2018). Optimizing Binary Serialization with an Independent Data Definition Format / A.Gustavo, J. Rosales / *International Journal of Computer Applications*, 180 (28).

7. Canggih Puspo Wibowo (2011). Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication. *International Conference on Informatics for Development* (pp. 40–43), At Yogyakarta, Indonesia.

8. Carrera, D., Gustavo, A., Rosales, J. (2017). A detailed analysis of data consistency concepts in data exchange formats (JSON & XML). *International Conference on Computing, Communication and Automation* (ICCCA) (pp. 72–77).

9. *Extensible Markup Language P World Wide Web Consorcium*. Retrieved from https://www.w3.org/XML.

10. *ECMA-404 The JSON Data Interchange Standard, 2017*. Retrieved from http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf.

11. *BSON Spec. BSON (Binary JSON): Specification*. Retrieved from http://bsonspec.org/ spec.html.

12. *FasterXML. Smile Data Format*. Retrieved from: https://github.com/FasterXML/smile-format-specification.

13. Sadayuki Furuhashi. *MessagePack*. Retrieved from: https://msgpack.org.

14. *Google Developers. Protocol Buffers: Developer Guide*. Retrieved from https://developers.google.com/protocol-buffers/docs/overview.

15. *Flat Buffers Internals*. Retrieved from https://google.github.io/flatbuffers /flatbuffers_internals.html.

16. *Apache Software Foundation. Apache Thrift*. Retrieved from: https://thrift.apache.org/

17. L. Peter Deutsch. *DEFLATE Compressed Data Format Specification version 1.3.* RFC 1951. Retrieved from: https://tools.ietf.org/html/rfc1951

18. Mamta Sharma (2010). Compression Using Huffman Coding. *IJCSNS International Journal of Computer Science and Network Security,* 10, 5, 133–140.

19. David Huffman (1952). A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE,* 40 (9), 1098–1952.

20. Ziv, J., Lempel, A. (1977). A Universal Algorithm for Sequential Data Compression. *IEEE Transactions on information theory,* IT-23, 3, 337–343.

UDC 004.7

Roman Andrushchenko

COMPARATIVE ANALYSIS OF THE PERFORMANCE CHARACTERISTICS OF DATA SERIALIZATION METHODS IN COMPUTER NETWORKS

Urgency of the research. *Nowadays there is a high growth of traffic on the Internet and in local networks. Therefore, it's necessary to pay attention to the performance of reception/transmission processes in the network software. Software that somehow transmits data over a computer network performs serialization/deserialization of internal structures in a byte stream. These processes are necessary and can directly affect the mechanisms of communication between hosts. In the article a comparative analysis of the serialization of structured data has been done, and the influence of compression on the application layer of the OSI model is investigated. The study will improve the data transfer processes within the OSI model, taking into account the high-level structure of the transmitted data.*

Target setting. *The processes of transforming the internal data structures into a form acceptable for transmission over the network can affect the speed and reliability of the software. The problem is that existing software performs a significant, and sometimes redundant work while transmitting data. Also, protocols of different layers of the OSI model usually don't take into account the particulars of the data itself, considering it only as a stream of bytes, which leads to less effective results. One way to improve this situation is to determine the structural features of the data being transmitted and analyze how they affect the process of serialization and deserialization.*

Actual scientific researches and issues analysis. *The publications, materials of conferences in the field of information technologies on the topic of research, as well as official documentation of popular data formats and Internet standards (RFC) has been considered. The analysis of existing researches of the work of protocols of application layer and data serialization formats has been done.*

Uninvestigated parts of general matters defining. *Investigating the impact of the internal structure and the format of transmitted data on the performance of data transfer, taking into account standard compression methods at the application layer of the OSI model (GZIP).*

The research objective. *To make a comparative analysis of the performance characteristics of the serialization processes for text and binary data formats; to investigate the effectiveness of their work.*

The statement of basic materials. *Analyzed and tested the work of realization of text and binary data serialization formats on message sets of different sizes and different structures by the experiment.*

Conclusions. *The article presents the results of comparative analysis of text and binary data serialization formats. The advantages and disadvantages of using standard compression mechanisms on the application layer of the OSI model in combination with different serialization mechanisms are formulated.*

Keywords. *HTTP; data serialization; data compression; computer network; encoding; network software.*

Fig.: 10. Tables: 1. References: 20.

Андрущенко Роман Богданович – аспірант, Чернігівський національний технологічний університет (вул. Шевченка, 95, м. Чернігів, 14035, Україна).

Andrushchenko Roman – PhD student, Chernihiv National University of Technology (95 Shevchenka Str., 14035 Chernihiv, Ukraine).

E-mail: arbamor@ukr.net

Андрущенко Р. Порівняльний аналіз показників ефективності методів серіалізації даних у комп'ютерних мережах. *Технічні науки та технології*. 2019. № 1 (15). С. 115–126.