УДК 621.391

ЛАБОРАТОРНЕ ДОСЛІДЖЕННЯ МЕТОДУ ОБСЛУГОВУВАННЯ ЧЕРГ, ЗАСНОВАНОГО НА КЛАСАХ, НА МАРШРУТИЗАТОРАХ IP-МЕРЕЖ

Я.І. Базуков, В.Х. Чакрян, А.А. Акулинічев, О.О. Мартинчук,
Харківський національний університет радіоелектроніки

Abstract – It has been established that packet queue organization and servicing mechanisms are among the key technological tools for ensuring quality of service, traffic management, and allocation of link and buffer resources. However, a significant drawback of most existing queue servicing mechanisms is the considerable amount of administrative configuration required, which increases sharply as the number of organized queues grows. Therefore, automating the configuration of communication equipment using network programming tools in the Python environment is a viable solution to this problem. This approach enables the reduction of configuration time and the decrease in the likelihood of administrative errors. The study focuses on an optimization-based method for queue servicing. This method is based on optimizing flow-aggregation processes and on the balanced allocation of bandwidth among class-based queues. The article presents a methodology for a laboratory experiment. During the experiment, automated configuration functions were delegated to a network server running a Python environment. In the experiment, this server was responsible for collecting network-state information and calculating control parameters that determined the servicing order of class-based queues. As an example, the study implemented automated configuration for class-based queues created using the CBWFQ mechanism. The experimental results confirmed the adequacy of the computational model and the selected queue-servicing method. The server ensured the correct solution of optimization problems using appropriate Python libraries, and the resulting calculations were subsequently applied via network programming to remotely and automatically configure the corresponding router interface. Verification of the router state and its interfaces confirmed the correctness of the conducted experiment. In future work, the laboratory experiment methodology may be extended and supplemented with network testing tasks focused on analyzing current values of key packet quality-of-service indicators.

Анотація – Встановлено, що одними з ключових технологічних засобів забезпечення якості обслуговування, управління трафіком та розподілу канального та буферного ресурсу є механізми організації та обслуговування черг пакетів. Однак, суттєвим недоліком більшості існуючих механізмів обслуговування черг є значні адміністративні налаштування, об'єм яких різко збільшується при зростанні кількості організованих черг. Тому автоматизація налаштувань комунікаційного обладнання засобами мережного програмування з використанням середовища Python є певним виходом із ситуації. Це дозволить скоротити час налаштування та зменшити ймовірність помилки адміністратора. В роботі обрано для дослідження оптимізаційний метод обслуговування черг. Він ґрунтується на оптимізації процесів агрегування потоків та збалансованого розподілу пропускної здатності між класовими чергами. В статті наведено методику лабораторного експерименту. В межах експерименту функції автоматизованого налаштування були делеговані на сервер мережі, на якому було розгорнуто середовище Python. Цей сервер в експерименті відповідав за збір інформації про стан мережі та розрахунок керуючих параметрів, які визначали порядок обслуговування класових черг. Як приклад, у процесі дослідження здійснювалась автоматизація налаштувань класових черг, створених за допомогою механізму CBWFQ. Результати експерименту підтвердили адекватність використаної моделі розрахунків та обраного для дослідження методу обслуговування черг. На сервері забезпечувалось коректне розв'язання оптимізаційних задач за допомогою відповідних бібліотек Python, а результати розрахунків засобами мережного програмування надалі віддалено автоматизовано налаштовувалися на відповідному інтерфейсі маршрутизатора. Шляхом перевірки стану маршрутизатора та його інтерфейсів підтверджено коректність проведеного експерименту. В подальшому методика лабораторного експерименту може бути розширена та доповнена задачами тестування мережі з погляду аналізу поточних значень основних показників якості обслуговування пакетів.

Вступ

Сучасний етап розвитку інформаційних та комунікаційних технологій пов'язаний як із значним розширенням переліку та типів сервісів, так і з підвищенням вимог до рівня якості обслуговування (Quality of Service, QoS) [1]. Рівень якості обслуговування, як правило, оцінюється через відповідні показники (метрики), до яких можуть

належати показники мережної продуктивності (пропускна здатність, середня затримка, джитер та рівень втрат пакетів), а також через показники якості, які сприймаються на рівні користувача – R-фактор, MMq (Multimedia Quality) [2-4]. В свою чергу числові значення перерахованих показників QoS традиційно залежить як від об'єму доступного мережного (канального, буферного) ресурсу, так і від ефективності технологічних засобів (протоколів, механізмів) управління тим чи іншим типом ресурсу.

В інфокомунікаційних мережах (ІКМ) пакетної комутації, які підтримують, наприклад, технології IP або MPLS, до основних засобів управління мережними ресурсами, трафіком та забезпечення QoS відносять наступні (рис. 1) [2]:

- засоби класифікації та маркування (пріоритизації) трафіка (пакетів);
- механізми створення та обслуговування черг на маршрутизаторах ІКМ;
- механізми активного управління чергами;
- засоби профілювання (обмеження або вирівнювання) трафіка;
- протоколи маршрутизації;
- протоколи резервування ресурсів.

Рис. 1. Основні засоби управління трафіком в ІКМ

Засоби класифікації та маркування трафіка відповідають за призначення пакетам відповідного пріоритету, який у подальшому визначає на маршрутизаторах ІКМ порядок та швидкість їхньої обробки. Механізми створення, обслуговування черг та активного управління ними забезпечують успішну боротьбу з перевантаженням та розподіл пропускної здатності інтерфейсів маршрутизаторів (каналів зв'язку ІКМ) між потоками пакетів, які мають різні характеристики (довжину, пріоритет, клас

тощо). Засоби профілювання трафіка також відповідають за боротьбу з перевантаженням мережі шляхом обмеження інтенсивності потоків пакетів, що надходять в мережу або на окремі її елементи (маршрутизатори, інтерфейси або навіть черги). Протоколи маршрутизації повинні визначати шляхи передачі пакетів в ІКМ, які є оптимальними з точку зору тих чи інших QoS-метрик. У свою чергу, протоколи резервування мають здійснювати резервування пропускної здатності інтерфейсів та буферного простору черг під запити сервісних потоків [1-3].

І. Огляд технологічних та теоретичних рішень щодо обслуговування черг на маршрутизаторах ІКМ

Серед перерахованих у вступі технологічних засобів управління трафіком важливе місце займають саме механізми створення та обслуговування черг на інтерфейсах маршрутизаторів ІКМ. Саме ці засоби є алгоритмічно-програмною основою таких двох архітектурних моделей забезпечення якості обслуговування, як DiffServ (Differentiated Services) та IntServ (Integrated Services) [1, 2].

На практиці в мережах IP/MPLS зараз використовується досить широкий спектр механізмів обслуговування черг, основними з яких є такі [2]:

- FIFO (First In, First Out);
- PQ (Priority Queuing);
- CQ (Custom Queuing);
- WFQ (Weighted Fair Queueing);
- CBQ (Class based Queuing);
- CBWFQ (Class-Based Weighted Fair Queuing);
- LLQ (Low Latency Queuing).

До ключових функцій перерахованих механізмів відносяться наступні:

- організація та підтримка множини черг на інтерфейсі;
- розподіл пакетів між створеними чергами;
- розподіл пропускної здатності інтерфейсу між чергами та визначення порядку обслуговування пакетів відповідно до їхньої довжини, пріоритету, класу тощо.

З метою підвищення рівня диференціації обслуговування на інтерфейсі може збільшуватися кількість черг. В ідеалі кожному окремому потоку пакетів варто було б надати окрему чергу з виділеною пропускною здатністю інтерфейсу, яка є необхідною саме цьому потоку. На практиці кількість підтримуваних черг може досить сильно коливатися: від 4 (для PQ) та 16 (для CQ) – до 256÷4096 (для WFQ) або десятків тисяч черг для CBWFQ або LLQ [2]. Механізм FIFO організовує лише одну чергу. Через обмежене число кількості черг потоки пакетів, як правило, агрегуються, тобто об'єднуються за певними спільними ознаками і направляються до однієї черги. Підвищення рівня диференціації обслуговування та збільшення кількості черг на інтерфейсі значно підвищує обчислювальні витрати маршрутизатора. Тому адміністратори мережі намагаються обрати компромісний варіант при налаштуванні маршрутизатора.

Ще одним важливим моментом при обґрунтуванні вибору механізму обслуговування черг та оптимізації його налаштування є рівень автоматизації його роботи. На жаль, переважна більшість існуючих механізмів обслуговування черг (CQ, PQ, CBQ, CBWFQ, LLQ) все ще потребують досить значних адміністративних (ручних) налаштувань. Ці налаштування, як правило, охоплюють задачі класифікації пакетів, їхнього розподілу між чергами та виділення тій чи іншій черзі певної частки пропускної здатності інтерфейсу. По-перше, це забирає багато часу, а, по-друге, значно підвищує вимоги щодо рівня кваліфікації адміністратора мережі, який повинен у реальному часі не тільки встигати проводили налаштування, але ще й обґрунтовувати кількісні значення чисельних керуючих параметрів. Зараз тільки механізм WFQ забезпечує автоматичне розв'язання ключових задач щодо обслуговування черг, але через евристичність закладених у його основу алгоритмів не варто очікувати оптимальність мережних рішень. Тому досить актуальною науковою та прикладною задачею є розробка математичних моделей та методів, які б стали алгоритмічно-програмною основою перспективних засобів обслуговування черг та забезпечували високий рівень автоматизації та ефективності покладених на них задач.

За останні роки науковцями світу запропоновано досить багато теоретичних рішень щодо підвищення ефективності обслуговування черг на маршрутизаторах ІКМ. Вибірковий аналіз основних рішень щодо обслуговування черг [5-11] дозволив зробити висновок, що основними трендами в цій області є такі:

- перехід до оптимізаційних рішень, використання яких оптимізує використання канального та буферного ресурсу [6-9];
- використання потокових моделей, коли поруч з параметрами окремих пакетів також використовуються у розрахунках і характеристики потоків;
- орієнтованість на класові черги, коли окремі потоки пакетів групуються та агрегуються у спільні черги на основі певних схожих характеристик, наприклад, вимог до рівня QoS;
- використання ієрархічних (багаторівневих) черг для підвищення масштабованості мережних рішень [9-11].

Основним критерієм, за яким визначається ефективність того чи іншого теоретичного або технологічного рішення, є практика, тобто лабораторний або фізичний експеримент. Саме експеримент дозволяє перевірити адекватність використаних математичних моделей, обґрунтувати діапазони зміни тих чи інших керуючих параметрів, визначити область застосування запропонованих рішень. Тому у даній роботі буде запропоновано методику лабораторного експерименту із дослідження методу обслуговування черг, заснованого на класах, на маршрутизаторах IP-мереж, запропонованого у роботах [6, 7] та вдосконаленого для ієрархічних черг у [9, 11].

II. Метод обслуговування черг, заснований на класах

У основу методу покладено розв'язання двох оптимізаційних задач:

- оптимального агрегування та розподілу потоків пакетів між чергами інтерфейсів маршрутизаторів ІКМ;

- збалансованого розподілу пропускної здатності інтерфейсу між сформованими класовими чергами.

Вирішення першої задачі було зведено у роботах [6, 7] до розв'язання оптимізаційної задачі лінійного цілочисельного програмування з критерієм оптимальності:

$$\min_x F,\ F = \sum_{i=1}^{N} \sum_{j=1}^{M} h_{i,j}^{x}\, x_{i,j}, \tag{1}$$

де

$$x_{i,j} \in \{0; 1\} \tag{2}$$

– множина керуючих змінних, кожна з яких характеризує частку i-го потоку пакетів, спрямованого на обслуговування до j-ї черги; N – загальна кількість потоків пакетів, що надходять на обраний інтерфейс маршрутизатора ІКМ; M – число організованих на інтерфейсі маршрутизатора черг; $h_{i,j}^{x}$ – метрика обслуговування пакетів i-го потоку j-ю чергою.

Значення метрики у виразі (1) розраховується відповідно до наступної формули:

$$h_{i,j}^{x} = \left(k_i^f - k_j^q\right)^2 + 1, \tag{3}$$

де k_i^f – значення класу i-го потоку; k_j^q – значення класу j-ї черги.

Таким чином, метрика $h_{i,j}^{x}$ буде приймати своє мінімальне значення у випадку співпадіння класів k_i^f та k_j^q. Це сприятиме тому, що потоки будуть напрямлятися у чергу із близьким значенням класу. З іншого боку, чим більшою буде різниця у значеннях класів потоків і черг, тим вищою стане метрика (3). Це стримуватиме направлення потоків різних класів до однієї спільної класової черги.

Вирішення другої задачі щодо збалансованого розподілу пропускної здатності інтерфейсу між сформованими класовими чергами здійснюється також у процесі розв'язання оптимізаційної задачі, але лінійного програмування. В цій задачі потрібно визначити множину керуючих змінних b_j, кожна з яких визначає частку пропускної здатності інтерфейсу, виділену для обслуговування j-ї черги [6, 7]. Для адекватного розподілу загальної пропускної здатності інтерфейсу b між створеними на ньому чергами на введені керуючі змінні накладаються такі обмеження:

$$b_j \geq 0, \quad \sum_{j=1}^{M} b_j \leq b, \quad \left(j = \overline{1, M}\right). \tag{4}$$

Для забезпечення збалансованого розподілу пропускної здатності інтерфейсу між сформованими на ньому класовими чергами необхідно виконати такі умови:

$$\alpha^* \sum_{i=1}^{N} a_i\, x_{i,j} \leq h_j^{\alpha} b_j,\ (j = \overline{1, M}), \tag{5}$$

де a_i $(i = \overline{1,N})$ – середня інтенсивність i-го потоку, яка вимірюється в пакетах за секунду (1/с); α^* – додатково введена керуюча змінна, яка обернено пропорційна до верхнього порогу завантаженості черг інтерфейсу (α).

Коефіцієнт завантаження кожної черги визначався за допомогою наступного виразу:

$$\rho_j = \frac{\sum_{i=1}^{N} a_i x_{i,j}}{b_j}, \ (j = \overline{1,M}).$$

На змінну α^* накладаються такі обмеження:

$$\alpha^* > 1. \tag{6}$$

У виразі (5) h_j^{α} – класовий коефіцієнт, який введено з метою врахування класів черг при збалансованому розподілі пропускної здатності інтерфейсу між чергами:

$$h_j^{\alpha} = 1 + \frac{k_j^q}{K \cdot D}, \ \left(j = \overline{1,M}\right),$$

де K – кількість класів потоків; $D \geq 1$ – коефіцієнт нормування, за допомогою якого здійснюється диференціація розподілу пропускної здатності інтерфейсу маршрутизатора між класовими чергами.

Тоді критерієм оптимальності рішень щодо збалансованого розподілу пропускної здатності інтерфейсу між сформованими класовими чергами є такий вираз:

$$\max_{b,\alpha*} \alpha^* \tag{7}$$

при наявності обмежень (4)-(6).

III. Опис лабораторного експерименту

Нехай топологія мережі, яка використана в процесі експерименту, складається з двох підмереж, які з'єднані між собою через маршрутизатор R1 (рис. 2).

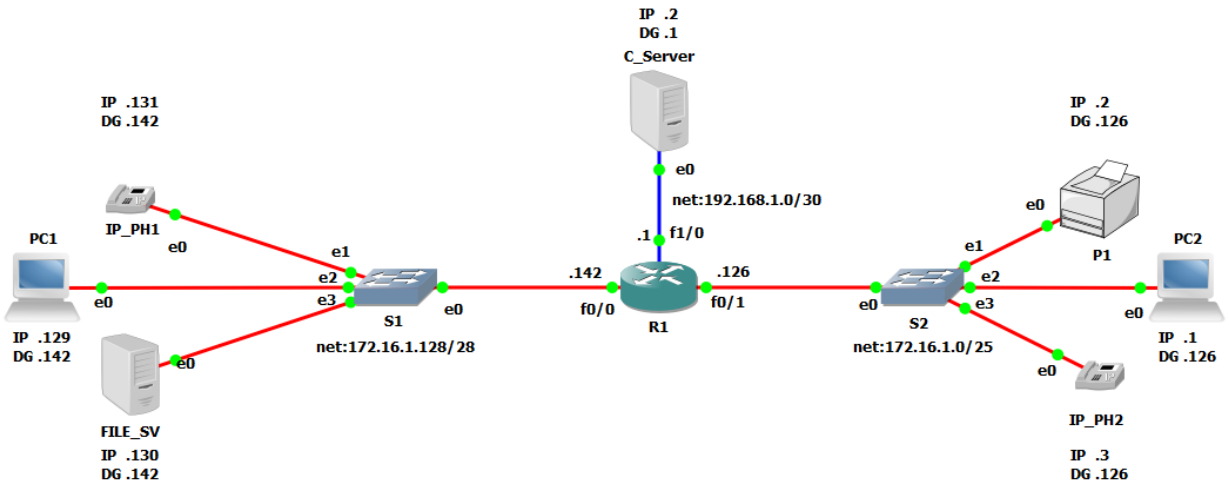

Рис. 2. Загальна структура мережі

Маршрутизатор виконує функції маршрутизації та управління трафіком. Для моделювання та перевірки працездатності мережі використано середовище GNS3, що дозволяє відтворити роботу реального мережного обладнання, провести тестування конфігурацій та здійснити налаштування обладнання та протоколів.

Перша підмережа розрахована на одинадцять кінцевих пристроїв, серед яких персональні комп'ютери, файловий сервер та IP-телефон. Вона підключається до маршрутизатора через інтерфейс FastEthernet0/0 та комутатор S1. Друга підмережа може включати шістдесят шість пристроїв, зокрема персональні комп'ютери, принтер та IP-телефон. Вона з'єднана з маршрутизатором через інтерфейс FastEthernet0/1 та комутатор S2. Окрім цього, маршрутизатор R1 має окреме з'єднання з сервером C_Server через виділений канал у мережі 192.168.1.0/30. Сервер виконує обчислювальні функції та бере участь у налаштуванні політик якості обслуговування. Зокрема, за його допомогою реалізується механізм CBWFQ.

Для організації IP-адресації використовується адресний простір 172.16.1.0/24. Застосування методу змінної довжини маски підмереж (VLSM) дозволяє оптимально розподілити адреси відповідно до потреб кожного мережного сегмента. Друга підмережа, яка потребує більшої кількості адрес, отримала діапазон 172.16.1.0/25, що включає 128 адрес, з яких 126 є доступними для використання хостами. Перша підмережа отримала діапазон 172.16.1.128/28, що містить 16 адрес, з яких 14 можуть бути використані кінцевими пристроями.

Таким чином, обрана схема дозволяє ефективно використовувати доступний адресний простір, забезпечує кожному мережному сегменту необхідну кількість IP-адрес та гарантує взаємодію між усіма пристроями мережі. Маршрутизатор R1 виконує не лише базову маршрутизацію, але й функції управління трафіком із підтримкою механізмів QoS, що дозволяє забезпечити пріоритизацію потоків та якісний зв'язок між підмережами.

Для проведення експерименту визначено вхідні дані для методу, описаному у другому розділі. Нехай загальна кількість черг становить 5, а кількість потоків – 9. Для кожного потоку задано інтенсивність у мегабітах за секунду та клас на основі DSCP (Differentiated Services Code Point). Характеристики потоків пакетів представлені в табл. 1.

Таблиця 1. Характеристики потоків трафіку

№ потоку	1	2	3	4	5	6	7	8	9
Інтенсивність, Мбіт/с	6	4	12	10	18	6	8	5	3
Політика DSCP	CS0	CS5	AF22	AF41	AF43	AF32	CS2	EF	CS1

У даному дослідженні під класом потоку вважався DSCP-пріоритет його пакетів. Для організації ефективного управління трафіком у системі було використано підхід рівномірного розподілу 64 (K=64) значень пріоритетів політики DSCP між п'ятьма чергами. Це дозволяє забезпечити оптимальне використання ресурсів для кожної черги залежно від її класу. Розподіл класів виконувався шляхом поділу діапазону 64 значень

на п’ять частин. Оскільки результат ділення (64 / 5 = 12,8) не є цілим числом, перші чотири черги отримали по 13 значень, а остання – 12. Таким чином, приблизні діапазони класів розподілилися наступним чином: для першої черги: 0–12, для другої: 13–25, для третьої: 26–38, для четвертої: 39–51, для п’ятої: 52–63. Середнє значення діапазону визначало значення класу черги: 6 для першої черги, 19 для другої, 32 для третьої, 45 для четвертої та 56,5 для п’ятої. Графічний розподіл пріоритетів черг наведено на рис. 3.

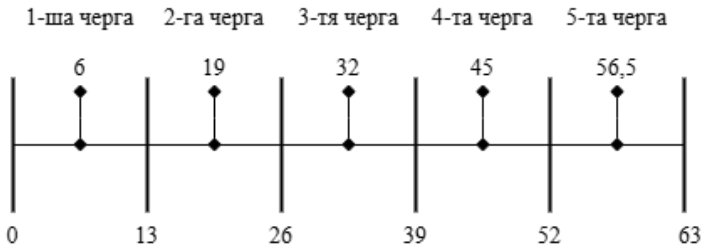

Рис. 3. Визначення класів п’яти черг

У межах експерименту (рис. 2) всі необхідні розрахунки здійснювалися на віддаленому сервері (C_Server). Сервер передавав готові налаштування до маршрутизатора через Telnet-з’єднання. Для цього на маршрутизаторі було активовано Telnet-сервер, що забезпечувало можливість віддаленого управління й автоматизацію процесу конфігурації. Таким чином, маршрутизатор застосовував сформовані параметри без необхідності ручного втручання.

У віртуальному середовищі кінцеві пристрої підключаються до мережевої топології GNS3 через адаптери, які створюють тунельний зв’язок між віртуальними машинами та маршрутизатором. Це забезпечувало інтеграцію тестового трафіку у змодельовану мережу та дозволяло перевіряти роботу політик QoS у реалістичних умовах. Для коректної роботи тестового стенда була налаштована мережна конфігурація віртуальних машин. Віртуальні машини були підключені через адаптер Generic Driver, який дозволяє створювати UDP-тунель для передачі трафіку до GNS3. Основні параметри мережевого адаптера показані на рис. 4.

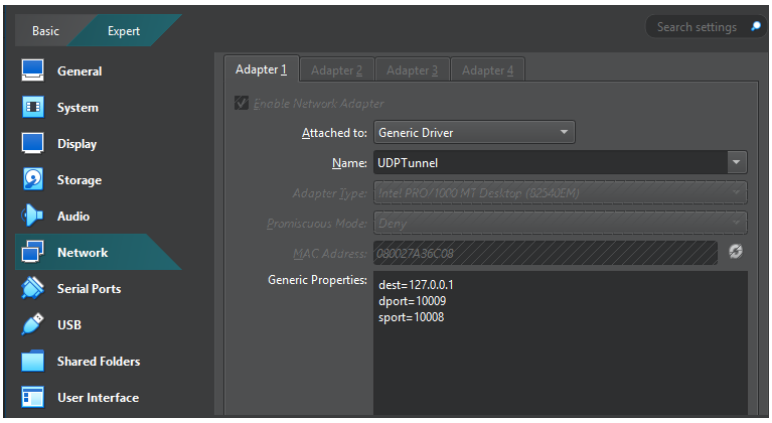

Рис. 4. Налаштування мережного адаптера віртуальної машини

Ці параметри дозволяють встановити зв'язок між віртуальними машинами та маршрутизатором через тунель, що необхідно для тестування мережі та її характеристик.Програмна реалізація обраного методу обслуговування черг (1)-(7) була здійснена мовою Python. Для розв'язання задачі агрегування потоків у середовищі Python застосовувалась функція milp з пакету SciPy (рис. 5).

```
# ==== ОПТИМІЗАЦІЙНА ЗАДАЧА ====
# Формування матриці Aeq та вектора beq
Aeq = np.zeros((M, M * N))
for i in range(M):
    Aeq[i, i * N:(i + 1) * N] = 1
beq = np.ones(M)

# Функція вартості: мінімізація різниці між DSCP та пріоритетами черг
hx = np.array([(kf[i] - kq[j])**2 for i in range(M) for j in range(N)])
for i in range(M):
    for j in range(N):
        hx[i * N + j] = (kf[i] - kq[j])**2 + 1

# Цілочислова оптимізація
integrality = np.ones(M * N)
bounds = Bounds(np.zeros(M * N), np.ones(M * N))
constraint_eq = LinearConstraint(Aeq, beq, beq)

# Розв'язання задачі MILP
res = milp(c=hx, integrality=integrality, bounds=bounds, constraints=[constraint_eq])

# Перевірка успішності
if not res.success:
    raise ValueError("MILP не вдалося знайти рішення. Перевірте вхідні дані.")

x = res.x
x2 = np.zeros((N, M))
for i in range(M):
    x2[:, i] = x[i*N:(i+1)*N]

print ("\n\n=================================================\n\n")
print("Результат агрегування потоків за чергами:")
for i, row in enumerate(x2):
    s = [j+1 for j, val in enumerate(row) if val == 1]
    print(f"Черга {i+1}: потоки {s if s else '— (немає)'}")
```

Рис. 5. Програмна реалізація моделі агрегування потоків за чергами

Друга задача – задача збалансованого розподілу пропускної здатності інтерфейсу між класовими чергами – розв'язувалася як задача лінійного програмування у середовищі Python за допомогою функції linprog (рис. 6).

```
    # Побудова задачі LP
    A = np.hstack([-np.eye(N), (h * aa).reshape(-1, 1)])
    b = np.zeros(N)
    Aeq_lp = np.hstack([np.ones(N), 0])
    beq_lp = np.array([B])

    f_lp = np.hstack([np.zeros(N), -1])
    lb = np.zeros(N + 1)
    ub = np.hstack([np.full(N, B), np.inf])

    res_lp = linprog(f_lp, A_ub=A, b_ub=b, A_eq=[Aeq_lp], b_eq=beq_lp, bounds=list(zip(lb, ub)), method='highs')

    # Перевірка успішності
    if res_lp.status != 0:
        raise ValueError("LP не вдалося знайти рішення. Перевірте вхідні дані.")

    x3 = res_lp.x

    po = []
    for j in range(N):
        if x3[j] != 0:
            po.append(aa[j] / x3[j])
        else:
            po.append(0)
    poo.append(po)

poo = np.array(poo)
print ("\n\n=====================================================\n\n")
print (f"Результат розподілу пропускної здатності між чергами:")
for i, val in enumerate(x3[:N], 1):
    print(f"  Черга {i}: {float(val):.4f}")
```

Рис. 6. Програмна реалізація моделі розподілу пропускної здатності

У процесі дослідження визначалось мінімальне значення коефіцієнта нормування (D_{min}) для забезпечення максимальної диференціації обслуговування класових черг та пакетів в них (рис. 7).

```
88   # Пошук мінімального Dmin
89   Dmin = 11  # Початкове значення
90   max_Dmin = 150  # Максимальне значення для перевірки
91   found = False
92
93   while Dmin <= max_Dmin:
94       try:
95           # Аналіз завантаження при поточному Dmin
96           D_range = np.arange(Dmin, 151, 5)
97           poo = []
98
99           for D in D_range:
100              h = 1 + kq / (K * D)  # класовий коефіцієнт
101
```

Рис. 7. Програмна реалізація з автоматизованого обирання значення Dmin

Заключним етапом лабораторного дослідження є автоматичне застосування розрахованої конфігурації на маршрутизаторі мережі. Після виконання агрегування потоків та визначення оптимального розподілу пропускної здатності інтерфейсу програма формує набір команд конфігурації, необхідних для реалізації механізму CBWFQ. До них належать команди створення *class-map* для класифікації трафіку, *policy-map* для призначення ресурсів чергам, а також команди налаштування інтерфейсів, на яких застосовується політика. Приклад автоматичної генерації цих команд подано на рис 8.

```python
# Генерація команд class-map
for j in range(N):
    if formatted_bandwidths[j] > 0:  # Перевіряємо, чи є трафік у черзі
        class_name = f"q{j+1}"
        if queue_dscp_map[j]:  # Генеруємо class-map лише якщо є DSCP
            commands.append(f"class-map match-any {class_name}") #R1(config)
            for dscp in set(queue_dscp_map[j]):
                dscp_name = dscp_reverse_map.get(dscp, f"dscp {dscp}")
                commands.append(f"match dscp {dscp_name}") #R1(config-cmap)
            commands.append(f"exit") #R1(config-cmap)

# Генерація команд policy-map
commands.append(f"policy-map pyauto") #R1(config)
for j in range(N):
    if formatted_bandwidths[j] > 0:  # Перевіряємо, чи є трафік у черзі
        class_name = f"q{j+1}"
        bw = max(1, int(formatted_bandwidths[j])) # R1(config-pmap)
        commands.append(f"class {class_name}") #R1(config-pmap)
        commands.append(f"bandwidth {bw}") #R1(config-pmap-c)
        commands.append(f"exit")    #R1(config-pmap-c)
commands.append(f"")    #R1(config-pmap)
```

Рис. 8. Генерація команд для налаштування механізму чергування CBWFQ

Для застосування згенерованої конфігурації програма використовувала бібліотеку telnetlib, яка забезпечує підключення до маршрутизатора за протоколом Telnet. Автоматизоване виконання цих дій значно зменшує ризик помилок, що виникають під час ручного введення, та прискорює процес налаштування. Відповідний фрагмент роботи програми зображено на рис. 9.

```python
    host = input("Введіть IP-адресу пристрою: ").strip()
    username = input("Введіть ім'я користувача: ").strip()
    password = input("Введіть пароль: ").strip()

    try:
        # Підключення до пристрою через Telnet
        tn = telnetlib.Telnet(host)
        tn.read_until(b"Username: ")
        tn.write(username.encode('ascii') + b"\n")
        tn.read_until(b"Password: ")
        tn.write(password.encode('ascii') + b"\n")

        # Виконання команд
        tn.write(b"enable\n")
        tn.read_until(b"Password: ")
        tn.write(password.encode('ascii') + b"\n")

        tn.write(b"conf t\n")
        output = tn.read_until(b"(config)#", timeout=2).decode('ascii')
        print(output)

        for cmd in commands:
            tn.write(cmd.encode('ascii') + b"\n")
            time.sleep(0.3)
            response = tn.read_very_eager().decode('ascii', errors='ignore')
            print(f"==> {cmd}")
            print(response)

```

Рис. 9. Автоматизоване введення згенерованих команд на пристрій

На основі проведених розрахунків було отримано результати щодо агрегування потоків у черги, розподілу пропускної здатності інтерфейсу між ними та оцінки їхнього завантаження при різних значеннях коефіцієнта нормування *D*.

Результати агрегування потоків подано на рис. 10. З аналізу даних видно, що розподіл є нерівномірним, що узгоджується з відмінностями у класах потоків та черг. У

п’яту чергу, яка мала найвище значення класу (56,5), не направлено жодного потоку, що пояснюється відсутністю вхідних даних, які б відповідали її діапазону.

```
Результат агрегування потоків за чергами:
Черга 1: потоки [1, 9]
Черга 2: потоки [3, 7]
Черга 3: потоки [4, 5, 6]
Черга 4: потоки [2, 8]
Черга 5: потоки — (немає)
```

Рис. 10. Результат агрегування потоків за чергами

Проаналізовано впливу коефіцієнта нормування D на завантаження черг (рис. 11). З рис. 11 видно, що при значеннях $D < 11$ метод не працює, бо не забезпечує виконання умови (5), а починаючи з $D \geq 11$, метод надає адекватні результати. При мінімальному значенні $D = 11$ забезпечується максимальна диференціяція в обслуговуванні черг різних класів – в цій точці значення коефіцієнтів завантаження різних класових черг максимально відрізняються. Масштабований варіант рис. 11 наведено на рис. 12, на якому відсутня інформація про завантаження порожньої п’ятої черги.

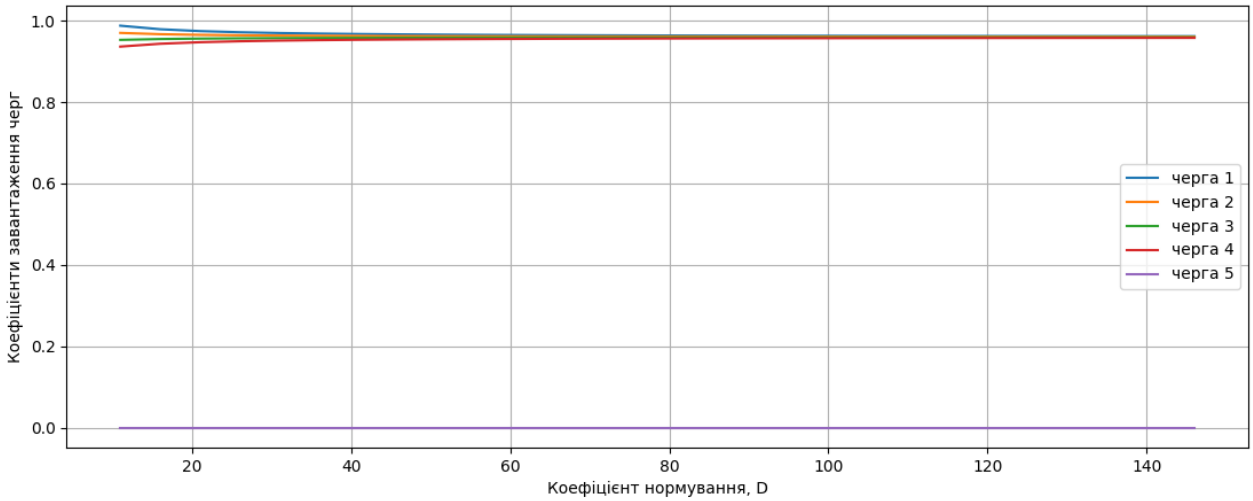

Рис. 11. Залежність коефіцієнтів завантаження черг від коефіцієнта нормування D

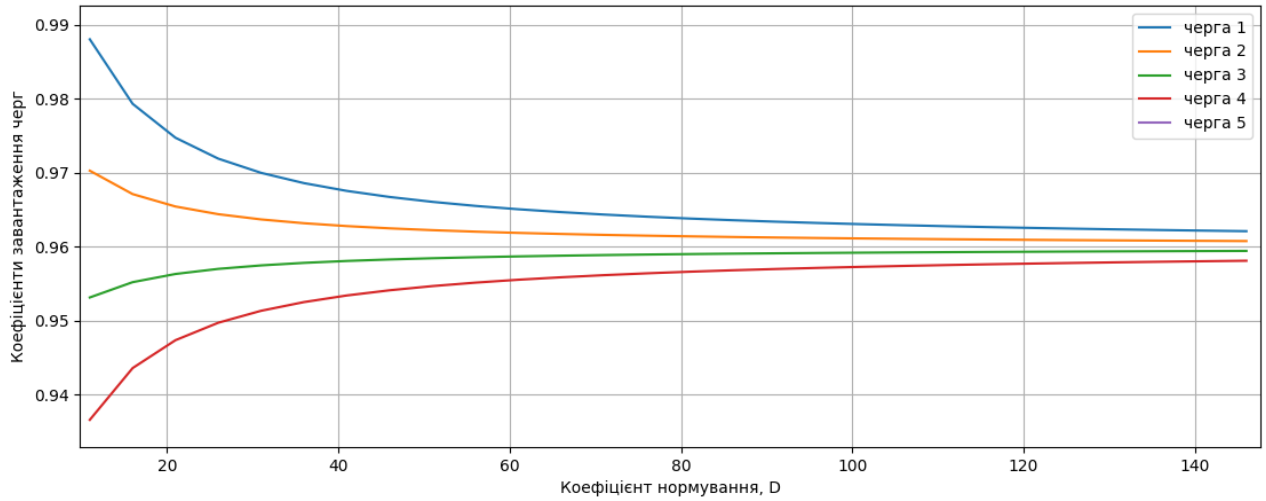

Рис. 12. Масштабований варіант результатів

Оптимізований та збалансований розподіл пропускної здатності інтерфейсу між класовими чергами при $D = 11$ здійснено на основі їхніх агрегованих інтенсивностей та класових коефіцієнтів h_j^{α} (рис. 13). Встановлена чітка закономірність: чим вище клас черги, тим нижче значення коефіцієнту її завантаження ρ_j, тим менші затримки та рівень втрат пакетів у цій черзі.

```
================================================


Результат розподілу пропускної здатності між чергами:
  Черга 1: 9.3542
  Черга 2: 20.8161
  Черга 3: 35.4365
  Черга 4: 9.3932
  Черга 5: 0.0000


================================================


Коефіцієнти завантаження для Dmin = 11:
  Черга 1: 0.9881
  Черга 2: 0.9703
  Черга 3: 0.9532
  Черга 4: 0.9366
  Черга 5: 0.0000


================================================


Сумарна інтенсивність: 72.0


================================================
```

Рис. 13. Результат розподілу пропускної здатності інтерфейсу між чергами та значення їхніх коефіцієнтів завантаження

Для наочного представлення результатів розрахунків щодо обслуговування черг розроблено схему рішення, наведену на рис. 14. Схема демонструє основні етапи обробки потоків, їхнє агрегування у черги, розподіл пропускної здатності інтерфейсу та вплив цих факторів коефіцієнти завантаження.

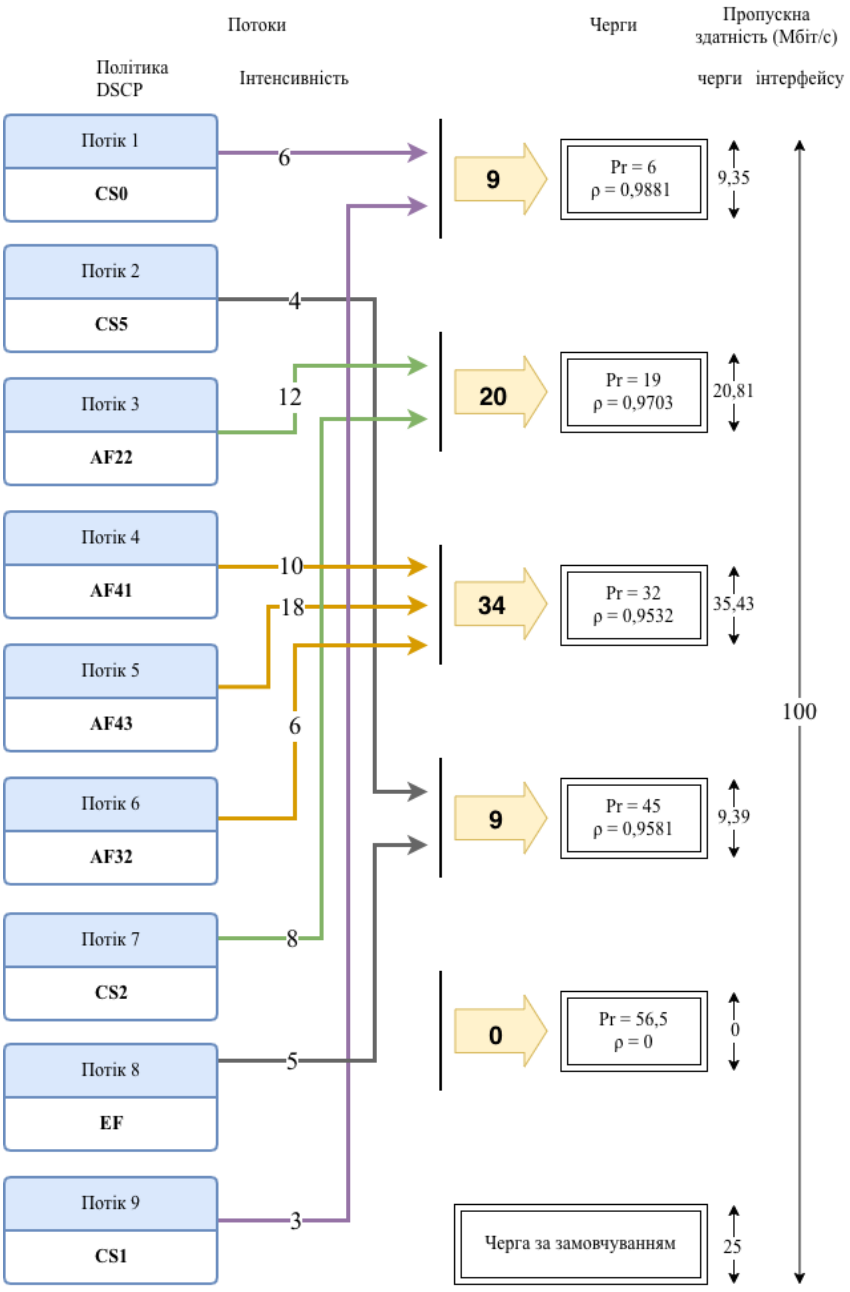

Рис. 14. Представлення результатів розрахунків щодо обслуговування черг (схема)

Після отримання результатів математичних розрахунків програма переходить до етапу налаштування обладнання. На цьому етапі користувачу пропонується ввести IP-адресу маршрутизатора, ім'я користувача та пароль для доступу через Telnet. Після введення даних програмний модуль самостійно встановлює з'єднання, формує необхідні команди відповідно до отриманих параметрів та виконує їх на пристрої (рис. 15).

```
Виконати налаштування через Telnet? (так(Y)/ні(N)): y
Введіть IP-адресу пристрою: 172.16.1.142
Введіть ім'я користувача: pyauto
Введіть пароль: pyaut0

R1#conf t
Enter configuration commands, one per line.  End with CNTL/Z.
R1(config)#
==> class-map match-any q1
```

Рис. 15. Запит авторизаційних даних у користувача та початок налаштування пристрою

Таким чином, реалізований підхід дозволяє повністю автоматизувати процес налаштування механізму CBWFQ, зменшити ризики помилок, що виникають при ручному конфігуруванні, та скоротити час, необхідний для впровадження QoS-рішень у мережне середовище.

Після завершення налаштувань механізму управління чергами CBWFQ було проведено серію перевірок для оцінки правильності застосованих параметрів та відповідності отриманих результатів попереднім розрахункам.

Першим кроком стала перевірка застосування політики обслуговування на інтерфейсі FastEthernet0/1 за допомогою команди *show policy-map interface f0/1*. Результати підтвердили, що політика pyauto, сформована автоматизованим засобом, була коректно активована на інтерфейсі. При цьому система відобразила статистику роботи кожної черги: кількість оброблених пакетів, рівень використання пропускної здатності та кількість відкинутих пакетів (рис. 16).

```
R1#show policy-map interface f0/1
 FastEthernet0/1

  Service-policy output: pyauto

    Class-map: q1 (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  dscp default (0)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match:  dscp cs1 (8)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
        Output Queue: Conversation 265
        Bandwidth 9354 (kbps) Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: q2 (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  dscp cs2 (16)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match:  dscp af22 (20)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
        Output Queue: Conversation 266
        Bandwidth 20816 (kbps) Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0
```

а)

```
    Class-map: q3 (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  dscp af41 (34)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match:  dscp af32 (28)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match:  dscp af43 (38)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
        Output Queue: Conversation 267
        Bandwidth 35436 (kbps) Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: q4 (match-any)
      0 packets, 0 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match:  dscp cs5 (40)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Match:  dscp ef (46)
        0 packets, 0 bytes
        5 minute rate 0 bps
      Queueing
        Output Queue: Conversation 268
        Bandwidth 9393 (kbps) Max Threshold 64 (packets)
        (pkts matched/bytes matched) 0/0
        (depth/total drops/no-buffer drops) 0/0/0

    Class-map: class-default (match-any)
      105 packets, 10995 bytes
      5 minute offered rate 0 bps, drop rate 0 bps
      Match: any
R1#
```

б)

Рис. 16. Перевірка налаштувань CBWFQ на інтерфейсі
а) статистика по чергам 1 та 2, б) статистика по чергам 3 та 4

Аналіз отриманих даних показав, що фактичне використання пропускної здатності кожної черги відповідає тим значенням, які були визначені під час розрахунків (рис. 13). Зокрема, найбільшу пропускну здатність отримали друга та третя черги, тоді як п'ята залишилася незадіяною. Це свідчить про повну відповідність конфігурації розрахованій моделі та підтверджує її адекватність.

Для отримання детальнішої інформації було використано команду *show policy-map*. Вона дозволила переглянути всі параметри політики ryauto, включаючи розподіл пропускної здатності між чергами відповідно до класів обслуговування. На рис. 17 наведено результати виконання цієї команди, які дозволяють провести глибший аналіз функціонування системи та перевірити дотримання визначених налаштувань.

```
R1#show policy-map
  Policy Map var2
    Class q1
      Bandwidth 9354 (kbps) Max Threshold 64 (packets)
    Class q2
      Bandwidth 20816 (kbps) Max Threshold 64 (packets)
    Class q3
      Bandwidth 35437 (kbps) Max Threshold 64 (packets)
    Class q4
      Bandwidth 9393 (kbps) Max Threshold 64 (packets)

R1#
```

Рис. 17. Детальна інформація про налаштування CBWFQ

Висновки

Технологічні механізми створення та обслуговування черг пакетів – є одним з основних засобів забезпечення якості обслуговування, управління трафіком та розподілу канального та буферного ресурсу. Суттєвим недоліком більшості існуючих механізмів обслуговування черг є необхідність у значних адміністративних налаштуваннях, об'єм яких різко збільшується при зростанні кількості організованих черг. Автоматизація мережних налаштувань засобами мережного програмування з використанням середовища Python є певною альтернативою, яка дозволить скоротити час налаштування та ймовірність помилки адміністратора.

У роботі на прикладі обраного методу обслуговування черг, який ґрунтується на оптимізації процесів агрегування потоків та збалансованого розподілу пропускної здатності між класовими чергами, наведено методику лабораторного експерименту. В межах експерименту функції автоматизованого налаштування були делеговані на сервер мережі (рис. 2), на якому було встановлено середовище Python. Цей сервер у експерименті відповідав за збір інформації про стан мережі та розрахунок керуючих параметрів, які визначали порядок обслуговування класових черг. Як приклад, в межах експерименту здійснювалась автоматизація налаштувань класових черг, створених за допомогою механізму CBWFQ.

Результати експерименту підтвердили адекватність використаної моделі розрахунків та обраного для дослідження методу обслуговування черг. На сервері забезпечувалось коректне розв'язання оптимізаційних задач за допомогою відповідних бібліотек Python, а результати розрахунків засобами мережного програмування надалі віддалено автоматизовано налаштовувалися на відповідному інтерфейсі маршрутизатора. Шляхом перевірки стану маршрутизатора та його інтерфейсів підтверджено коректність проведеного експерименту. В подальшому методика лабораторного експерименту може бути розширена та поповнена задачами тестування мережі з точки зору аналізу поточних значень основних показників якості обслуговування пакетів.

Список літератури

1. *Barreiros, M.; Lundqvist, P.* (2016), QoS-Enabled networks: Tools and foundations; John Wiley & Sons: Hoboken, NJ, USA, 256 p.

2. *Vegesna, S.* (2001), IP quality of service, Cisco press, 343 p.

3. QoS: Congestion Management Configuration Guide, Cisco IOS XE 17; Cisco Systems, Inc.: San Jose, CA, USA, 2019. **URL**: https://www.cisco.com/c/en/us/td/docs/ios-xml/ios/qos_conmgt/configuration/xe-17/qos-conmgt-xe-17-book/qos-conmgt-oview.html.

4. *Лемешко, О. В., Єременко, О. С., Невзорова, О. С.* (2020), Потокові моделі та методи маршрутизації в інфокомунікаційних мережах: відмовостійкість, безпека, масштабованість, Харків: ХНУРЕ, 308 с. **DOI**: https://doi.org/10.30837/978-966-659-282-1.

5. *Chahed, H., Kassler, A.* (2023), "TSN Network Scheduling—Challenges and Approaches", Network, No. 3, P. 585–624. **DOI**: https://doi.org/10.3390/network3040026.

6. *Lemeshko, O., Lebedenko, T., Holoveshko, M.* (2021), "Development and Research of Active Queue Management Method on Interfaces of Telecommunication Networks Routers", In Data-Centric Business and Applications; Lecture Notes on Data Engineering and Communications Technologies; Springer: Cham, Switzerland, No. 69, P. 1–20. **DOI**: https://doi.org/10.1007/978-3-030-71892-3_1.

7. *Лебеденко, Т.М., Головешко, М.В., Северілов, А.В.* (2019), "Результати експериментального дослідження методу активного управління чергами на інтерфейсах телекомунікаційних мереж", Проблеми телекомунікацій, No. 2(25), С. 37–55. **DOI**: https://doi.org/10.30837/pt.2019.2.03.

8. *Titarenko, L., Lemeshko, O., Yeremenko, O., Savchenko, R., Barkalov, A.* (2025), "Traffic Engineering Queue Optimization Models with Guaranteed Quality of Service Support", Electronics, No. 14, 4078. **DOI**: https://doi.org/10.3390/electronics14204078.

9. *Lemeshko, O., Yeremenko, O., Titarenko, L., Barkalov, A.* (2023), "Hierarchical Queue Management Priority and Balancing Based Method under the Interaction Prediction Principle", Electronics, No. 12, 675. **DOI**: https://doi.org/10.3390/electronics12030675

10. *You C., Zhao Y., Feng G., Quek T. Q. S., Li L.* (2023), "Hierarchical Multiresource Fair Queueing for Packet Processing", IEEE Transactions on Network and Service Management, No. 20(1), P. 726–740. **DOI**: https://doi.org/10.1109/TNSM.2022.3197747.

11. *Lemeshko, O., Persikov, A., Yeremenko, O., Yevdokymenko, M.* (2025), "Method of Hierarchical Queue Management on Network Routers Based on the Goal Coordination Principle", In: Ilchenko, M., Uryvsky, L., Globa, L. (eds) Advanced Smart Information and Communication Technology and Systems. MCT 2024. Lecture Notes in Networks and Systems, No. 1470, Springer, Cham, P. 200–214. **DOI**: https://doi.org/10.1007/978-3-031-94799-5_11.