**ЗМІСТ**

[ВСТУП 2](#_Toc485208180)

[РОЗДІЛ 1. ОБЗОР МЕТОДІВ МАРШРУТИЗАЦІЇ В МЕРЕЖІ MPLS 4](#_Toc485208181)

[1.1. Мережа MPLS 4](#_Toc485208183)

[1.1.1. Базові поняття мережі MPLS 5](#_Toc485208184)

[1.1.2. Стек міток MPLS 7](#_Toc485208185)

[1.1.3. Механізм функціонування мережі MPLS 8](#_Toc485208186)

[1.1.4. Протоколи розподілу міток MPLS 9](#_Toc485208187)

[1.2. Поняття маршрутизації в мережах та її способи 12](#_Toc485208188)

[Висновки до розділу 1 21](#_Toc485208189)

[РОЗДІЛ 2. АЛГОРИТМ МАРШРУТИЗАЦІЇ МЕТОДОМ ЗУСТРІЧНОЇ ХВИЛІ 22](#_Toc485208190)

[2.1. Визначення кількості непересічних шляхів при маршрутизації 22](#_Toc485208192)

[2.2. Формування множини непересічних шляхів на основі хвильового алгоритму 24](#_Toc485208193)

[2.3. Приклад роботи однобічного хвильового алгоритму 24](#_Toc485208194)

[2.4. Затримка при збірці повідомлень 30](#_Toc485208195)

[2.5. Формування множини непересічних шляхів на основі методу зустрічної хвилі 31](#_Toc485208196)

[Висновки до розділу 2 38](#_Toc485208197)

[РОЗДІЛ 3. МОДЕЛЮВАННЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ 39](#_Toc485208198)

[3.1. Опис логіки програми 40](#_Toc485208200)

[3.2. Опис програмного інтерфейсу 43](#_Toc485208201)

[3.3. Розгляд прикладу роботи програми 49](#_Toc485208202)

[3.4. Порівняння результатів запропонованого алгоритму з традиційними 52](#_Toc485208203)

[Висновки до розділу 3 57](#_Toc485208204)

[ВИСНОВКИ 58](#_Toc485208205)

[СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ: 60](#_Toc485208206)

ВСТУП

У наш час через значне зростання кількості користувачів комп'ютерних мереж ускладнюється структура мереж та взаємодія між ними. Тому ускладнюється завдання маршрутизації, а саме пошук оптимальних шляхів в мережі для швидкої доставки повідомлень користувачів мережі, у зв’язку з чим є потреба в швидкому та надійному методі передачі даних.

Одним із варіантів покращення якості та підвищення надійності передачі даних є використання на мережевому рівні багатошляхової маршрутизації. При багатошляховій маршрутизації для кожного віртуального канала ставиться у відповідність декілька шляхів, що дозволяє підвищити ефективність використання каналів зв’язку. Оскільки характеристики комп’ютерної мережі змінюється динамічно під час роботи, необхідно швидко виконувати ремаршрутизацію, багатошляхова маршрутизація дозволяє виконати це завдання швидко, мінімально навантажуючи мережу службовими пакетами та забезпечуючи рівномірне навантаження каналів зв’язку. Окрім цього, багатошляхова маршрутизація дозволяє покращити надійність передачі даних за рахунок передачі декількох частин повідомлення паралельно, резервування шляхів та їх використання в разі відмови чи перенавантаження критичних ділянок мережі або відправка декількох копій повідомлення задля гарантування доставки без помилок.

Наступним ефективним методом підвищення надійності передачі слід вважати використання технології багатопротокольної комутації по міткам (Multi-Protocol Label Switching, MPLS), що дозволяє підвищити інтелектуальність мережевих механізмів. Віртуальна структура мережі MPLS дозволяє відправляти дані паралельними шляхами для пришвидшення передачі та кращої надійності.

Для підвищення ефективності маршрутизації постає питання дослідження існуючих та розробка нових алгоритмів маршрутизації, оскільки через високу складність структур сучасних мереж завдання маршрутизації вирішується на в повному обсязі та має високу часову складність.

РОЗДІЛ 1

ОБЗОР МЕТОДІВ МАРШРУТИЗАЦІЇ В МЕРЕЖІ MPLS

1.1. Мережа MPLS

MPLS (multiprotocol label switching) – це метод передачі даних у високопродуктивних мережах за допомогою міток [1]. Головними особливостями MPLS є масштабування та можливість роботи від будь-яких проколів передачі даних. Технологію розроблено у компанії Cisco у 2001 році.

MPLS посідає проміжну ланку між другим (канальним) та третім (мережевим) рівнями еталонної моделі OSI [2].

За видом комутації відноситься до мереж з комутацією пакетів.

Головною перевагою є повна незалежність від особливостей технології канального рівня: MPLS можна використовувати для передачі різного роду трафіку: комірки ATM, IP-пакети, фрейми SONET/SDH та кадри Ethernet [3]. Не є необхідною підтримка декількох технологій нижчого рівня.

MPLS було розроблено, враховуючи слабкі та сильні сторони технології ATM. З одного боку, інженерами було усвідомлено, що на рівні ядра сучасних мереж немає необхідності у комірках ATM малого фіксованого розміру. Оскільки сучасні оптичні мережі мають достатньо високу швидкість передачі даних, щоб навіть пакет із максимальною для ATM довжиною 1500 байт мав невелику затримку в буферах комутаційного забезпечення (що було суттєвим при передачі голосових даних у ATM). З іншого боку, збережено переваги ATM щодо механізмів керування трафіком (TE, teletraffic engineering) і управління окремо від даних, що передаються, що було великою перевагою ATM при використанні у великих мережах передачі даних.

Також, значно зменшується обсяг аналізованої при маршрутизації інформації: аналізуються лише мітки, при цьому можна використовувати попередньо обчислені результати (на відміну від звичайної маршрутизації, де рішення приймає кожен маршрутизатор) [4].

MPLS надає додаткові можливості для надання різних рівнів якості обслуговування (QoS, quality of service), керування трафіком (TE) та створення приватних віртуальних мереж (VPN, virtual private network).

1.1.1. Базові поняття мережі MPLS

Комітет розробників мережі MPLS визначив три базові поняття: клас еквівалентності пересилки (FEC), мітка (label), комутуємий за мітками шлях (LSP) [5].

Клас еквівалентності пересилки (FEC, forwarding equivalence class) – це множина пакетів, що передаються однаковим чином (наприклад, задля забезпечення заданого QoS).

Мітка (label) – короткий ідентифікатор фіксованої довжини, що визначає приналежність пакета до того чи іншого FEC.

Комутуємий за мітками шлях (LSP, label switched path) – шлях, що проходить через один чи кілька LSR, за яким передаються пакети одного й того самого FEC.

Маршрутизатор комутації за мітками (LSR, label switch router) – маршрутизатор, що здатен передавати пакети за технологією MPLS.

Прикінцевий маршрутизатор (LER – MPLS edge router) – прикінцевий маршрутизатор мережі MPLS, з’єднує мережу MPLS з вузлом, що не належить до даної мережі.

При звичайній маршрутизації у заголовку IP-пакету міститься набагато більше інформації, ніж того потребує вибір наступного маршрутизатора. Цей вибір можна зробити шляхом виконання наступних двох функцій: співставлення пакету до конкретного класу FEC; співставлення кожному FEC наступного кроку маршрутизації. Таким чином клас FEC ставиться у відповідність одного разу на вході у мережу MPLS, у наступних маршрутизаторах заголовок не аналізується [6]. При співставленні пакетів із різними FEC велику роль грають IP-адреси, пріоритети обслуговування та інші параметри трафіку. Кожен FEC оброблюється окремо, що дозволяє отримати необхідну якість обслуговування у мережі MPLS.

Розмір мітки у MPLS складає 32 біти, причому є можливість для кожного пакету призначити набір міток – стек [7]. Мітка у MPLS має наступну структуру (рис. 1.1): 20 бітів відводиться на значення мітки, за яким власне відбувається маршрутизація (від 0 до 1048575), 3 біти відповідають за QoS (класи якості обслуговування), 1 біт є флагом (S), шо говорить, чи є ця мітка кінцевою у стеку міток, 8 бітів відповідають за час життя пакету (TTL).



Рис. 1.1. Структура мітки MPLS

Варто зазначити, що мітки мають локальне значення, оскільки конкретна прив’язка мітки до FEC має місце лише для конкретної пари маршрутизаторів. Мітка використовується при передачі пакету від верхнього до нижнього, де є вхідною, замінюється на нову висхідну, що також має локальне значення на наступній ділянці шляху.

Комутуємий за мітками шлях LSP – це послідовність MPLS-маршрутизаторів. Набір пакетів, що передається по LSP, відноситься до одного FEC, і кожний маршрутизатор LSR в LSP-тунелі визначає для нього свою мітку. LSP-тунель створюється всередині LSP-шляху. У одному LSP-щляху може бути створено декілька LSP-тунелів із різними точками входу та передачі, причому у кожному тунелі можуть бути створено тунелі нижчого рівня. Вищезгадане говорить про ієрархічність будови мережі MPLS.

Існують два варіанти створення тунелів. У першому випадку використовуються принцип hop-by-hop, де кожен маршрутизатор обирає наступну ділянку шляху пакету самостійно. У другому випадку при маршрутизації використовується вказівка від маршрутизатору, що є вхідним для даного маршрутизатора на даному LSP-шляху. Таким чином, у першому випадку шлях пакетів визначається довільним чином, у другому випадку він відомий наперед.

1.1.2. Стек міток MPLS

Пакет, що передається мережею MPLS, зазвичай, містить не одну, а декілька міток, що утворюють стек. Головна мета утворення стеку міток – це можливість утворення деревовидної структури шляхів LSR, що починаються у спільному вхідному маршрутизаторі, а також створення LSP-тунелів. Специфікація кодування стеку міток MPLS визначена у документі RFC3032 "MPLS Label Stack Encoding" [8].

MPLS може виконувати зі стеком наступні операції: додавати мітку у стек, видаляти мітку зі стеку та замінювати мітку. Ці операції також можуть бути використані для зливання та розгалуження потоків даних. Стек працює за принципом LIFO: «останнім прийшов, першим пішов». При додаванні мітки у стек (push operation) додається нова мітка у голову стеку, при видаленні мітки зі стеку (pop operation) видаляється верхня мітка стеку.

MPLS надає можливість за допомогою стеку об’єднати декілька LSP у один. До стеку міток кожного з цих LSP додається спільна мітка, у результаті утворюється спільний шлях MPLS. При закінченні спільної ділянки шляху виконується його розгалуження на окремі складові LSP. Таким чином виконується ієрархічна пересилка даних у MPLS, що наразі є широко використовуваною функцією. При її використанні не треба дублювати глобальну маршрутну інформацію, і це робить мережу MPLS більш стабільною та масштабованою, ніж традиційний аналог.

У загальному вигляді стек міток MPLS інкапсулюється між заголовками канального та мережевого рівнів (рис. 1.2). Мітка MPLS розміщується у існуючому форматі заголовку рівня 2 у випадку ATM та Frame Relay, або розміщується у спеціально створеному для MPLS заголовці у випадку Ethernet та PPP.

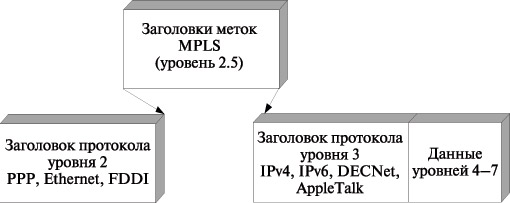


Рис. 1.2. Принцип інкапсуляції заголовку MPLS

1.1.3. Механізм функціонування мережі MPLS

Роботу мережі MPLS можна розглянути у 5 стадій (рис. 1.3).

Етап 1. У мережі автоматично створюються таблиці маршрутизації, для чого використовуються маршрутизатори/ комутатори мережі (ATM або IP) та внутрішні протоколи маршрутизації (OSPF, IS-IS тощо).

Етап 2. Створені вище таблиці маршрутизації використовуються у якості топології мережі при генерації значень міток протокол розподілу міток (LDP, label distribution protocol), при цьому відбувається посилання на сусідні пристрої. На цьому етапі визначаються шляхи з комутацією по мітках, причому на відміну від технології приватних віртуальних каналів АТМ PVC, це відбувається автоматично, а не вручну.

Етап 3. Прикінцевий маршрутизатор міток LER отримує вхідний пакет даних, визначає для нього набір послуг третього рівня, що необхідні цьому пакету (зазначений клас якості обслуговування, зазначена швидкість тощо). Використовуючи вказану інформацію та правила топології мережі прикінцевий LER генерує мітку, що записується у голову стека, та передає далі пакет.

Етап 4. Наступний проміжний LSR, приймає пакет, аналізує мітку пакету, приймає рішення щодо нової мітки (за локальною таблицею цього LSR) та передає отриманий пакет далі. Процедура повторюється, доки пакет на дістанеться прикінцевого маршрутизатору.

Етап 5. Прикінцевий LER отримує пакет, видаляє мітку, за прочитаним заголовком визначає місце призначення пакета та передає його далі.

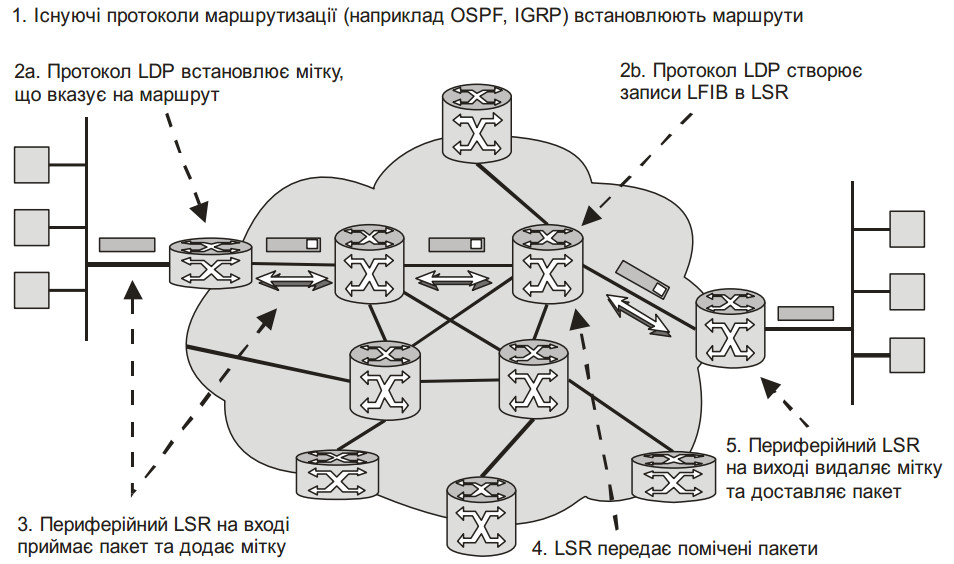


Рис. 1.3. Передача пакету даних по мережі MPLS

При маршрутизації мітка MPLS порівнюється з наперед розрахованими таблицями комутації (що зберігається у LSR-маршрутизаторах) та містить необхідну інформацію третього рівня. Завдяки цьому пристрої LSR можуть автоматично надавати пакетам необхідно IP-послуги. Оскільки таблиці маршрутизації розраховано наперед, знімається необхідність повторної обробки пакетів у кожній точці передачі, тому MPLS є добре масштабованим.

1.1.4. Протоколи розподілу міток MPLS

Будова MPLS жорстко не регламентує використання конкретного протоколу розподілу міток; у одній мережі MPLS можуть використовуватись протоколи:

1) спеціальний протокол розподілу міток LDP (label distribution protocol), перетворює у мітки IP-адреси отримувача;

2) протоколи сигналізації CD-LDP і RSVP; використовуються для оптимізації розподілу трафіку в мережі та для резервування ресурсів;

3) розширення можливостей протоколів маршрутизації, наприклад, протокола міждоменевої маршрутизації (ВGР, border gateway protocol); працює з зовнішніми мітками VPN.

Архітектура MPLS в усіх вищезгаданих випадках передбачає, що співставлення мітки та конкретного FEC виконує прикінцевий LER, що є кінцевим на цьому маршруті (рис. 1.4). Таким чином, призначення міток завжди відбувається знизу, у бік, протилежний напряму трафіка. Таким чином, нижній LSR інформує сусідні верхні LSR про прив’язку міток та пакетів. Цей процес і називається розподілом міток, і гарантується протоколом розподілу міток.

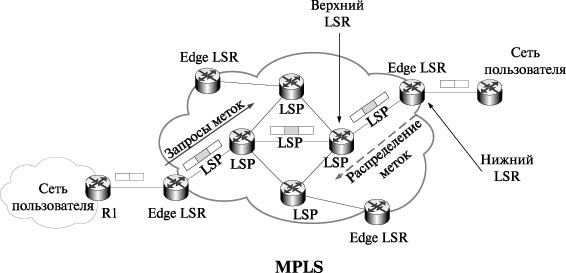


Рис. 1.4. Початок розподілу міток

У документації наразі визначається два методи, за якими визначається FEC у протоколі LDP: Address Prefix – адресний префікс будь-якої довжини від нуля до повної адреси; Host Address – повна адреса отримувача [9].

Рішення про призначення мітки може орієнтуватися на певні критерії передачі, такі як гарантування якості обслуговування, керування трафіком у мережі, створення віртуальної приватної мережі, багатоадресної розсилки тощо.

Хоча розподіленням міток зажди займається нижній маршрутизатор, він не обов’язково є ініціатором розподілення міток. Такий процес може розпочати і верхній LSR, якщо відправить до нижнього LSR відповідний запит (downstream on-demand). У мережі можливі варіанти, що використовують тільки запити знизу (unsolicited downstream), чи тільки запити зверху, чи комбінований варіант.

Процедура розподілу міток може бути незалежною чи впорядкованою. У першому випадку LSR може повідомити свого верхнього LSR ще до того, як отримав повідомлення про призначення мітки до конкретного FEC від свого нижнього LSR. У другому випадку повідомити верхнього сусіда дозволяється лише після отримання інформації від нижнього сусіда.

Також однією з важливих функцій протоколу LDP є знайдення петель. Для цього можуть бути використані поля у повідомленнях label request та label mapping, а саме path vector (містить перелік LSR-ідентифікаторів) та hop count (містить лічильник LSR, що було пройдено).

Оскільки протокол LDP при маршрутизації може користуватися лише таблицями маршрутизації IP, для подолання цього обмеження було запропоноване розширення протоколу LDP, що має назву CR-LDP.

CR-LDP містить механізми, що дозволяють створити та використати шляхи LSP, що задаються явним чином. При створенні шляху у протоколі CR-LDP використовується більше інформації, ніж може бути отримано від звичайних протоколів внутрішньої маршрутизації. Тому протокол використовується у тих сферах, де важливим є можливості керування трафіком та забезпечення якості обслуговування (що вимагають додаткової інформації про мережу). У цьому випадку певні ділянки шляху можна задати явним чином. При цьому при маршрутизації не використовуються таблиці пересилок, а запит виконується у відповідності з інструкціями, що містяться у повідомленні.

CR-LDP не підтримує обчислення явно заданих шляхів, тому інформацію про динамічне резервування пропускної здатності каналів зв’язку передають за допомогою додаткових протоколів OSPF та IS-IS або сповіщень про стан каналів LSA. За допомогою цих механізмів CR-LDP має можливість резервувати пропускну здатність.

1.2. Поняття маршрутизації в мережах та її способи

Під маршрутизацією в комп’ютерних мережах розуміють процедуру визначення шляху в мережах передачі даних [10]. Маршрутизація є характерною для глобальних та регіональних мереж, для яких розглядається відповідна група протоколів. Важливим завданням маршрутизації є знаходження якомога коротшого шляху, що матиме найменшу затримку, буде використовувати менше вузлів та зв’язків, внаслідок чого мережею можна буде передати більше інформації. Крім цього, важливим аспектом є питання гарантування надійності та безпеки передачі даних.

За еталонною моделлю OSI протоколи маршрутизації належать до третього (мережевого) рівня. Завдання маршрутизації належить до класу NP-повних задач, оскільки додавання нових вузлів значно прискорює зростання кількості зв’язків між вузлами. Окрім власне обчислювальної складності при маршрутизації також існує декілька вагомих проблем. При маршрутизації необхідно опитати всі вузли системи, що вимагає високої степені координації роботи вузлів мережі. Також необхідно забезпечити працездатність системи при відмові певних вузлів чи зв’язків; зменшити ризик утворення перенавантаження мережі та розробити механізм керування перенавантаженням.

Способи маршрутизації вирізняються широким різноманіттям та мають розвинену класифікацію [11]. Класифікацію способів маршрутизації наведено на рис. 1.5.



Рис. 1.5. Класифікація способів маршрутизації в комп’ютерних мережах

За наявністю головного центру керування, що керує процесом маршрутизації виділяють розподілені та централізовані способи. Відповідно, централізовані способи маршрутизації передбачають наявність головного центру керування, що робить можливим полегшення структури окремого вузла мережі (оскільки вони звільняються від процесу визначення наступного вузла маршруту). Проблемою такої будови мережі є зростання навантаження на головний центр керування, що призводить до зменшення надійності роботи маршрутизації (треба обробити значну кількість інформації, при відмові головного вузла в мережі неможливо провести процес маршрутизації).

Розподілені способи маршрутизації не передбачають наявність головного центру керування. Це призводить до ускладнення конкретних вузлів мережі, що при такому способі мають визначати наступний шлях руху пакетів. Перевагою таких способів є безвідмовна робота мережі при відмові конкретного вузла.

Маршрутизацію поділяють на просту та табличну.

Проста маршрутизація не враховує особливості топології мережі, через що вирізняється невисокою продуктивністю та високою ймовірністю відказу при перенавантаженні мережі. Таблична маршрутизація надає можливість кожному вузлу побудувати спеціальну таблицю, в якій зазначається відповідність між конкретною адресою доставки повідомлення та шляхом, яким цю адресу можна буде досягти.

Серед способів простої маршрутизації вирізняють випадкову, лавинну та по попередньому досвіду [12]. Випадкова маршрутизація передає пакет у випадково згенерованому напрямі (за виключенням шляху, яким прийшов пакет). Через те, що мережа є обмеженою, за обмежену кількість кроків пакет надійде до кінцевої точки маршруту. Через вкрай неефективне використання пропускних каналів мережі та значну затримку при передачі випадкова маршрутизація рідко використовується, хоча із певними модифікаціями може бути використана при незначній інтенсивності передачі даних у мережі разом із невеликим розміром мережі.

Лавинна маршрутизація передбачає розмноження пакетів по вихідних напрямах при отриманні наступним вузлом, проте завантажує мережу зайвими пакетами.

При маршрутизації по попередньому досвіду відбувається корекція наперед заданих шляхів, якщо цього вимагає ситуація, проте «адаптація» мережі відбувається повільно та неефективно.

В залежності від часу, у який створюються таблиці маршрутизації, табличну маршрутизацію можна поділити на статичну та динамічну. Статична маршрутизація передбачає утворення таблиці маршрутів у момент створення мережі, зміна маршрутів може відбутися, як правило, лише при відмові якогось вузла.

Серед способів статичної маршрутизації виділяють фіксовану маршрутизацію та маршрутизацію методом найкоротшої черги.

У випадку фіксованої маршрутизації ставиться відповідність між початковим і кінцевим вузлом маршрутизації та віртуальним каналом передачі даних. Якщо такий віртуальний канал містить один шлях, говорять про одношляхову маршрутизацію. Цей спосіб маршрутизації є найбільш простим серед табличних способів, проте має недолік схильності до нерівномірного навантаження мережі, оскільки для нього не є можливим врахування реального трафіку окремих зв’язків між вузлами та відмов окремих вузлів/зв’язків.

Одним із методів вирішення вищезгаданої проблеми є утворення відповідності між початковою та кінцевої вершиною маршрутизації із набором декількох шляхів. Це дозволяє вирішити проблему перенавантаження мережі декількома способами. По-перше, можна розбити повідомлення, що потребує передачі, на декілька частин, і ці частини відправляти паралельно шляхами сформованого віртуального каналу. По-друге, можна встановити декілька резервних шляхів, які будуть використовуватись у раз появи критичних вузлів, що не можуть нормально працювати у умовах високого об’єму трафіку, чи при відмові певних маршрутів.

Перевагою при багатошляховій маршрутизації є утворення непересічних шляхів, де кожний є незалежним від іншого, оскільки при відмові одного шляху інші зможуть передавати інформацію без втрат для отримувача.

Сьогодні багатошляхову маршрутизацію реалізують декількома способами [13]. Найперспективнішими є використання мережі MPLS; розширення протоколу MPTCP та протоколу BGP-XM; або побудова специфічних мереж на кшталт мереж із топологією FatTree.

Робоча група IETF (Internet Engineering Task Force) розробила стандарт багатошляхового протоколу для транспортного рівня шляхом розширення протоколу TCP до багатошляхового MPTCP, що дозволяє розділити транспортне з’єднання на декілька потоків. Протокол дозволяє вирішити проблеми підвищення пропускної здатності, надійності та зменшення навантажень.

Також використовується розширення протоколу BGP (BGP-XM, BGP eXtended Multipath), яке об’єднує декілька шляхів спільним агрегованим атрибутом [14].

Класифікацію багатошляхової маршрутизації наведено на рис. 1.6.



Рис. 1.6. Класифікація способів багатошляхової маршрутизації

За математичним методом можна виокремити комбінаторні та потокові алгоритми побудови багатошляхових маршрутів.

Комбінаторні базується на рекурсивному обході вершин, кожного етапу обираються вершини із мінімальним значенням обраної метрики. Комбінаторні алгоритми мають більшу часову складність та не враховують тип каналу зв’язку [15].

У потокових алгоритмах задача маршрутизації моделюється за рахунок моделювання технології розподілення потоку на графах.

За використанням резерву алгоритми багатошляхової маршрутизації поділяються на холодного резерву та без резервування.

При маршрутизації без резервування резервні шляхи використовуються при відповідному запиті, при маршрутизації без резервування передача даних відбувається по всім шляхам.

З точки зору процесу побудови маршруту вирізняють маршрутизацію від відправника та покрокову маршрутизацію з обробкою маршрутної інформації на кожному кроці. Перший варіант гарантує знаходження непересічних шляхів, другий варіант є масштабованим.

До наступного типу статичної маршрутизації відносять маршрутизацію методом найкоротшої черги. У базовому варіанті у методі при маршрутизації для кожного вузла існує таблиця маршрутів, для кожного вузла виокремлюють декілька можливих напрямків передачі пакетів, серед яких наступний обирається випадково. Через низьку ефективність такого методу використовують його певну модифікацію із використанням пріорітетів напрямків передачі пакетів. На кожному вузлі, коли необхідно, вибір наступного напряку передачі виконується за зменшенням пріорітетів можливих напрямків, що є вільними. Модифікований алгоритм є доволі поширеним у мережах із низькою надійністю комутаційних систем.

Динамічна маршрутизація обумовлює зберігання інформації, що знаходиться у таблицях маршрутизації, таким чином, що вона може змінюватись під час маршрутизації враховуючи навантаження та стан вузлів та каналів передачі даних. Оновлення стану таблиць відбувається за рахунок обміну спеціальними пакетам між вузлами мережі, що вказують на стан мережі і містять інформацію про її топологію, навантаження каналів та черги у вузлах. Суттєвим чином на маршрутизацію впливає, наскільки миттєвим буде відгук мережі на зміну її стану. Проте прагнення до оптимальності керування трафіком має наслідок у значно більшій кількості службових пакетів, додатковому навантаженні каналів та, зрештою, зниження ефективності роботи мережі.

Динамічна маршрутизація містить декілька важливих механізмів:

* власне формування маршрутів за таблицями маршрутизації у кожному вузлі;
* керування пакетами при передачі: визначення наступного вузла за маршрутами;
* корекція маршрутів;
* контроль стану мережі;
* обмін повідомленнями про стан мережі між вузлами для корекції маршрутів.

Серед видів динамічної маршрутизації можна навести централізовану, розподілену, локальну та гібридну маршрутизацію в залежності від методів корекції маршрутів.

При централізованій маршрутизації кожним вузлом виконується аналіз та передача інформації про стан навантаження цього вузла. Керівний вузол виконує аналіз про стан мережі загалом, та визначає стратегії створення окремих маршрутів. Формування маршрутів відбувається використовуючи критерій мінімального часу затримки пакетів. Після обрахування шляхів, керівний вузол складає таблицю маршрутів для кожного вузла мережі, які потім надсилаються до кожного вузла. До недоліку централізованої маршрутизації є залежність від керівного вузла, причому навантаження на цей вузол зростає зі збільшенням вузлів у мережі; затримка на обробку інформації про стан мережі, який постійно змінюється.

Цих недоліків позбавлені методи розподіленого керування маршрутизацією, які найбільш широко застосовуються в сучасних глобальних комп'ютерних мережах.

Розподілена динамічна маршрутизація вирішує ці недоліки. Кожний вузол містить власну таблицю маршрутів, яку сам корегує, одержуючи інформацію від суміжних вузлів про навантаження каналів, черги пакетів та часові затримки. Оскільки для маршрутизації враховується час відгуку після відправки пакетів, при значній розбіжності часу реакції з планованим буде виконано корекцію маршрутів суміжних вузлів.

При локальній динамічній маршрутизації у кожному вузлі міститься попередньо завантажена інформація; кожен вузол самостійно обирає наступний на шляху, не використовуючи інформацію від суміжних вузлів; виконується аналіз черг на виході з вузлу і на підставі цього виконується корекція шляхів. Ця маршрутизація є ефективною, проте складною при формуванні та обробці маршрутів, є можливим зникнення пакетів при обробці.

Гібридна маршрутизація є поєднанням рис локальної та централізованої динамічної маршрутизації. Зокрема, дельта-маршрутизація містить керівний центр, що з деякою затримкою реагує на ситуацію у мережі в цілому, а окремі вузли швидко реагують на зміну стану в околі.

Всі вказані алгоритми використовують одну з трьох математичних моделей – Дейкстри, Белмана-Форда, Флойда-Уоршела. Статичні алгоритми використовують їх при плануванні на мережі глобально, у той час як динамічні тільки локально, де треба беручі до уваги задані критерії оптимальності.

Зі статичних методів найбільш широке застосування через високу швидкість та надійність має алгоритм заливки [16]. Принцип функціонування полягає в розсилці пакету, що прийшов у всі лінії, окрім тієї, за якою він прийшов. Але його єдиний і головний мінус – неприпустимо велике значення трафіку. Даний алгоритм використовується при тестуванні для оцінки ефективності нових розробок і все ще використовується в спеціалізованих мережах (наприклад, військових).

Алгоритм маршрутизації на підставі потоку ґрунтується на припущенні про те, що трафік усередині мережі можна описати статистичним законом, на підставі якого і вибираються оптимальні схеми маршруту.

Динамічні алгоритми для оцінки оптимальності шляху використовують механізм метрик. Метрикою для дистанційно-векторної маршрутизації є число відрізків мережі між відправником і одержувачем. На підставі даної метрики вибирається оптимальний маршрут, локально використовуючи алгоритм Дейкстри. Цей алгоритм має ряд недоліків, головним з яких є проблема нескінченних обчислень. Практична реалізація алгоритму виконана у вигляді протоколу RIP [17]. Зараз цей метод поступився місцем більш досконалим, але його ще підтримує переважний відсоток устаткування, що випускається та операційних систем (MVS, Unix, сімейство MS Windows Server).

Одним з найбільш досконалих на сьогоднішній момент алгоритмів в маршрутизації є маршрутизація з обліків стану ліній. Метрикою для такого алгоритму є середня величина затримки для тестового пакета, що відображає не тільки довжину маршруту, але і завантаження каналу. Практичною реалізацією цього алгоритму є протокол OSPF [18].

Висновки до розділу 1

MPLS є масштабованою та незалежною від будь-яких інших протоколів мережею. При маршрутизації у MPLS кожному пакету даних присвоюється унікальна мітка, подальша передача пакету до інших вузлів відбувається лише на підставі значення цієї мітки, що робить можливим створення віртуального каналу, незалежного від будь-яких середи та протоколу передачі даних. Розглянуто переваги використання мережі MPLS для формування множини непересічних шляхів.

Розглянуто основні поняття маршрутизації у комп’ютерних мережах, описано класифікацію методів маршрутизації у сучасних мережах. Виокремлено переваги та сферу застосування багатошляхової маршрутизації.

Показано, що існуючі алгоритми формування множини непересічних шляхів мають високу часову складність, тому вирішення задачі зменшення часу формування множини непересічних шляхів є актуальним.

РОЗДІЛ 2

АЛГОРИТМ МАРШРУТИЗАЦІЇ МЕТОДОМ ЗУСТРІЧНОЇ

ХВИЛІ

2.1. Визначення кількості непересічних шляхів при маршрутизації

Більшість відомих методів формування множини непересічних шляхів побудовано на методі направленого перебору можливих шляхів. При цьому попереднє визначення кількості непересічних шляхів дозволяє скоротити кількість операцій перебору, тому дозволяє зменшити часову складність формування шляхів.

Кількість непересічних шляхів залежить від топології графу мережі, зокрема, від характеру області зчленування графу, кількості вершин та їх ступеня [19]. У загальному випадку максимальна кількість непересічних шляхів між вершинами *вi* і *вj* задається мінімальними значеннями ступеня вершин *вi*, *вj* та мінімальною множиною області зчленування графу *G*=(*В,E*). Під мінімальною множиною області зчленування *ВS* = *В*\(*В1*∪*В2*) розуміємо множину вершин, видалення яких призводить до поділу графу *G*=(*В,E*) на два підграфа *G1*=(*В1,E1*) і *G2*=(*В2,E2*) за умови , що *вi*∈*В1*, а *вj*∈*В2*.

Причому під ступенем зв’язності розуміємо внутрішню зв’язність: для вершин з множини *В1*(*В2*) це кількість ребер, якими вона пов’язується з вершинами з множини *В1*(*В2*).

Розглянемо початкову та кінцеву вершини *вi* та *вj*, що мають ступені зв’язності *Si* та *Sj*. При *Si* і *Sj* ≥ min|*Вs*|*n* максимальна кількість непересічних шляхів визначається значенням min|*Вs*|*n*. У цьому випадку кожна з вершин *вi*∈min *Вs* обов’язково знаходиться на шляху між вершинами *в0* і *вn*.

Відомі комбінаторні та графічні методи формування шляхів на основі різних метрик не гарантують множини непересічних шляхів. Наприклад, для графу, зображеного на рис. 2.1, при формуванні шляхів на основі метрики, що оптимізує мінімальну кількість переходів, формується лише один мінімальний шлях *P1* = {*V1-V6-V7-V13-V21-V27-V31*}. Усі інші шляхи матимуть спільні ділянки із цим маршрутом.



Рис 2.1. Формування мінімального шляху із оптимізацією кількості переходів

Слід зазначити, що для графу, зображеного на рис. 2.2, можна побудувати чотири непересічних шляхи:

*P1* = {*V1-V6-V11-V16-V17-V23-V28-V31*};

*P2* = {*V1-V3-V8-V7-V12-V18-V19-V25-V29-V31*};

*P3* = {*V1-V4-V9-V13-V20-V26-V30-V31*};

*P4* = {*V1-V5-V10-V15-V22-V21-V27-V31*}.

Ситуація, подібна до зазначеної на рис. 2.1, виникає, якщо сформований шлях містить переріз графу системи, тому існує необхідність розробки такого алгоритму, що унеможливить виникнення подібної проблеми.

Треба скористатись наступним правилом: створення нового непересічного шляху у графі є подібним до видалення відповідних вузлів та ребер, що є інцидентними до розглянутих вузлів [20]. Тому задля гарантування більшої кількості зв’язків між підграфами, утвореними в результаті видалення непересічного маршруту, на кожному етапі слід обирати вершину з найменшою зв’язністю.

Можна визначити наступний критерій для формування множини непересічних шляхів:

1) Обираємо шлях *Pi* з мінімальним значення зовнішнього ступеня для поточної вершини *вk*.

2) При рівних значеннях зовнішнього ступеня обираємо шлях *Pi* з мінімальною вагою *Wi* =.

3) Наступною вершиною для шляху *Pi* обираємо вершину *вl*, що є суміжною до поточної вершини *вk* та має найменше значення зовнішнього ступеня.

2.2. Формування множини непересічних шляхів на основі хвильового алгоритму

Часова складність формування багатоканального віртуального шляху, що складається з *q* фізичних каналів з допомогою комбінаторних алгоритмів, наприклад, алгоритма Дейкстри дорівнює О(*qN*2) [21]. Одним із шляхів зменшення обчислювальної складності є, наприклад, виключення повного перебору із алгоритму маршрутизації методом «гілок та границь», проте даний алгоритм не враховує довжину шляху при формуванні декількох непересічних шляхів.

Тому у якості базового алгоритму обираємо хвильовий алгоритм, вводимо метрику найменшої довжини шляху, враховуємо критерій із пункту 2.1. Отримуємо алгоритм, зображений на рис. 2.2.

2.3. Приклад роботи однобічного хвильового алгоритму

Розглянемо приклад модифікованого хвильового алгоритму, що формує непересічні шляхи, взявши для розгляду граф, зображений на рис. 2.3. Непересічні шляхи будемо формувати між вершинами *V1* та *V31*.



Рис. 2.2. Блок-схема модифікованого хвильового алгоритму

Рис. 2.3. Граф мережі

Проміжний вигляд шляхів позначаємо жирною лінією, хвилі вершин – виділеною областю.

На першому кроці починаємо побудову множини непересічних шляхів із розгляду суміжних вершин для початкової вершини V1, та будуємо перші ланки шляхів: *P1* = {*V1-V5*}; *P2* = {*V1-V4*}; *P3* = {*V1-V3*}; *P4* = {*V1-V6*}; *P5* = {*V1-V2*}.

На другому кроці для усіх шляхів обраховуємо значення зовнішнього ступеня для останньої вершини у шляху. Маємо: для шляху *P1* для вершини *V5* значення ступеня дорівнює 1, для шляху *P2* для вершини *V4* значення ступеня дорівнює 2, для шляху *P3* для вершини *V3* значення ступеня дорівнює 1, для шляху *P4* для вершини *V6* значення ступеня дорівнює 3, для шляху *P5* для вершини *V2* значення ступеня дорівнює 0. Шлях *P5* продовжити неможливо, інші шляхи сортуємо за значенням ступеня за зростанням: *P1*, *P3*, *P2*, *P4*. Серед суміжних до вершин першої хвилі виділяємо вершини *V7*, *V8*, *V9*, *V10*, *V11*. Продовжуємо шляхи *P1* та *P3*: *P1* = {*V1-V5-V10*}, *P3* = {*V1-V3-V8*}, оскільки іншого альтернативного вибору немає. Після прокладання шляху *P3* ступінь зв’язності вершини *V4* шляху *P2* стає рівним 1, і єдиним можливим варіантом є прокладання шляху до вершини *V9*, тому шлях має вигляд *P2* = {*V1-V4-V9*}.

Для шляху *P4* є дві можливі опції: вершини *V7* та *V11*. Оскільки перша має зовнішній 4, а друга 3, обираємо вершину *V11* як наступну. Шлях має наступний вигляд: *P4* = {*V1-V6-V11*}. Ситуація після другої хвилі зображена на рис. 2.4.

Під час третьої хвилі у якості наступних розглядаються вершини *V12*, *V13*, *V14*, *V15*, *V16*. Для шляху обираємо вершину *V14* як наступну, оскільки вона має зв’язність 2, а вершина *V13* – 4. Для інших шляхів альтернативного вибору наступних вершин не маємо.

Проміжну ситуацію після четвертого кроку наведено на рис. 2.5.



Рис. 2.4. Проміжна ситуація після другого кроку

Наступні кроки виконуємо аналогічним чином. Повну хронологію маршрутизації наведено у табл. 2.1.

У результаті маршрутизації маємо ситуацію, зображену на рис. 2.6.

Сформовані шляхи мають наступний вигляд:

*P1* = {*V1-V5-V10-V15-V22-V21-V27-V31*};

*P2* = {*V1-V4-V9-V14-V13-V20-V26-V30-V31*};

*P3* = {*V1-V3-V8-V7-V12-V18-V19-V25-V29-V31*};

*P4* = {*V1-V6-V11-V16-V17-V23-V28-V31*}.

Побудову шляхів *P1* та *P4* виконано за 7 кроків, шляху *P2* – за 8 кроків, шляху *P3* - за 9 кроків.

 Рис. 2.5. Проміжна ситуація після четвертого кроку

Рис. 2.6. Остаточні результати прокладання непересічних шляхів

Після прокладання шляхів є можливість балансування шляху *P1* з вершини *V15* на вершину *V14*, а шляху *P3* – з вершини *V25* на вершину *V24*.

Оскільки ступінь зв’язності початкової вершини дорівнює 5, а кінцевої вершини – 4, максимальна кількість непересічних шляхів дорівнює 4, що і було отримано із застосуванням алгоритму.

Таблиця 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| T | *P1* | *P2* | *P3* | *P4* |
| 1 | *V1-V5* | *V1-V4* | *V1-V3* | *V1-V6* |
| 2 | *V1-V5-V10* | *V1-V4-V9* | *V1-V3-V8* | *V1-V6-V11* |
| 3 | *V1-V5-V10-V15* | *V1-V4-V9-V14* | *V1-V3-V8-V7* | *V1-V6-V11-V16* |
| 4 | *V1-V5-V10-V15-V22* | *V1-V4-V9-V14-V13* | *V1-V3-V8-V7-V12* | *V1-V6-V11-V16-V17* |
| 5 | *V1-V5-V10-V15-V22-V21* | *V1-V4-V9-V14-V13-V20* | *V1-V3-V8-V7-V12-V18* | *V1-V6-V11-V16-V17-V23* |
| 6 | *V1-V5-V10-V15-V22-V21-V27* | *V1-V4-V9-V14-V13-V20-V26* | *V1-V3-V8-V7-V12-V18-V19* | *V1-V6-V11-V16-V17-V23-V28* |
| 7 | *V1-V5-V10-V15-V22-V21-V27-V31*  Шлях побудовано | *V1-V4-V9-V14-V13-V20-V26-V30* | *V1-V3-V8-V7-V12-V18-V19-V25* | *V1-V6-V11-V16-V17-V23-V28-V31*  Шлях побудовано |
| 8 | – | *V1-V4-V9-V14-V13-V20-V26-V30-V31*  Шлях побудовано | *V1-V3-V8-V7-V12-V18-V19-V25-V29* | – |
| 9 | – | – | *V1-V3-V8-V7-V12-V18-V19-V25-V29-V31*  Шлях побудовано | – |

2.4. Затримка при збірці повідомлень

Розглянемо збірку повідомлення, що складається з трьох частин, кожна з яких має рівну за довжиною затримку передачі *Ti* = *const*. Під часом *Td* розуміємо час запису частини повідомлення (рис. 2.7).



Рис. 2.7. Збірка частин повідомлення

У даному випадку час збірки усього повідомлення Тсб=(*t2*-*t0*) є мінімальним і дорівнює 2*Td*. При затримці передачі кожної частини повідомлення на значення величину Δ*t*=(*t1*-*t0*) відносно попередньої частини повідомлення час збірки є мінімальним та дорівнює *Т*зб=(*N*-1)*Td* [22].

При затримці *τi*>*Td* передачі *i*-ї частини повідомлення час збірки повідомлення стає більшим на значення *tz*=*τi*-*Td*.

Тому при значній потужності множини непересічних шляхів для віртуального каналу постає проблема існування затримки при процесі збірки повідомлення із декількох частин, що передаються різними маршрутами. Для гарантування мінімізації ускладнення процесу конструювання трафіку слід формувати віртуальні багатошляхові канали враховуючи задані параметри QoS до трафіку. У цьому разі створення декількох непересічних маршрутів дозволить оптимізувати процес передачі повідомлень у комп’ютерних мережах великої розмірності.

2.5. Формування множини непересічних шляхів на основі методу зустрічної хвилі

При виконанні процесу маршрутизації однією з головних є проблема великої області пошуку для прокладання нових маршрутів. Є актуальною вона і для хвильового алгоритму. Якщо розглянути комп’ютерну мережу з вершинами *V0* та *Vk* та діаметром мережі між цими вершинами D, то область пошуку буде пропорційною до квадрату діаметру мережі: *S =πD2* (рис. 2.8а).

Щоб оптимізувати алгоритм та зменшити область пошуку можна запропонувати будувати дерева шляхів паралельно з початкової та кінцевої вершини. У цьому випадку області пошуку від початкової вершини та кінцевої вершини до вершини, де дерева шляхів перетинаються є пропорційною до квадрату півдіаметра: *S0=Sk=πD2/4*, а загальна область пошуку зменшується у два рази: *S1+S2=πD2/2* (рис. 2.8б).

Головною проблемою при вказаній оптимізації може бути пошук тих вершин, де дерева шляхів, що будуються від початкової та кінцевої вершин, перетинаються.

Доведемо, що точки перетину дерев існують, що зробить можливим виконання маршрутизації зазначеним методом.



а) область пошуку для хвильового алгоритму; б) область пошуку для алгоритму зустрічної хвилі

Рис. 2.8.Область пошуку шляхів між *V0* та *Vk*

Дерева *Ti* (*Bi*,*Ei*) та *Ti* (*Bi*,*Ei*) мають точки сполучення, якщо множини перетину їх вершин та ребер не є пустими, при цьому вершини *E0*=*Ei* ∩ *Ej* є точками сполучень цих дерев.

Оскільки пошук відбувається у обмеженому графі, завдання буде виконано за зліченну послідовність кроків.

Таким чином, задача пошуку шляхів зводиться до знаходження точок сполучення дерев від початкової та кінцевої вершин. При цьому не є принципіальним, який саме алгоритм вибору вершин використовувати, його можна варіювати в залежності від задачі маршрутизації.

При маршрутизації аналогічно із модифікованим хвильовим алгоритмом у якості наступної для певного шляху обираємо вершину, що має менший зовнішній ступінь. Таким чином гарантується формування непересічних шляхів.

Недоліком методу можна вважати необхідність виділення пам’яті для шляхів від кінцевої вершини, проте цей мінус нівелюється пришвидшенням пошуку множини непересічних шляхів в ідеальному випадку в два рази.

Дерева шляхів будуються, доки не буде збудовано усі непересічні шляхи між початковою та кінцевою вершинами. Блок-схему алгоритму маршрутизації методом зустрічної хвилі наведено на рис. 2.9.

Розглянемо приклад роботи алгоритму формування непересічних шляхів методом зустрічної хвилі, взявши для розгляду граф, зображений на рис. 2.10. Непересічні шляхи будемо формувати між вершинами *V1* та *V40*.

Проміжний вигляд шляхів від початкової та кінцевої вершин позначаємо жирною та пунктирною лінією, хвилі вершин – виділеною областю.

На першому такті відбувається утворення маршрутів від початкової вершини до суміжних вершин: *P01* = {*V1-V2*}; *P02* = {*V1-V3*}; *P03* = {*V1-V4*}; *P04* = {*V1-V12*}; *P05* = {*V1-V6*}. На другому такті відбувається утворення маршрутів від початкової вершини до суміжних вершин: *Pк1* = { *V40-V37*}; *Pк2* = { *V40-V32*}; *Pк3* = {*V40-V33*}; *Pк4* = {*V40-V38*}; *Pк5* = {*V40-V39*}.



Рис. 2.9.Алгоритм формування множини непересічних шляхів

Рис. 2.10.Граф для моделювання

На третьому такті (пошук від початкової вершини): згідно ступеня зв’язності останньої вершини подовжуємо шляхи у порядку: *P01*, *P03*, *P05*, *P02*, *P04*. Отримуємо: *P01* = {*V1-V2-V7*}; *P02* = {*V1-V3-V9*}; *P03* = {*V1-V4-V11*}; *P04* = {*V1-V12-V9*}; *P05* = {*V1-V6-V13*}. На четвертому такті (пошук від кінцевої вершини): згідно ступеня зв’язності останньої вершини подовжуємо шляхи у порядку: *Pк1*, *Pк2*, *Pк5*, *Pк4*, *Pк3*. Отримуємо: *Pк1* = {*V40-V37-V31*}; *Pк2* = { *V40-V32-V25*}; *Pк3* = {*V40-V33-V34*}; *Pк4* = {*V40-V38-V35*}; *Pк5* = {*V40-V39-V36*}.

За викладеним алгоритмом виконуємо обчислення шляхів на наступних тактах. Маємо наступні результати після п’ятого такту (пошук від початкової вершини): *P01* = {*V1-V2-V7-V8*}; *P02* = {*V1-V3-V9-V15*}; *P03* = {*V1-V4-V11-V10*}; *P04* = {*V1-V12-V9-V22*}; *P05* = {*V1-V6-V13-V18*}. Маємо наступні результати після шостого такту (пошук від кінцевої вершини): *Pк1* = {*V40-V37-V31-V24*}; *Pк2* = {*V40-V32-V25-V20*}; *Pк3* = {*V40-V33-V34-V27*}; *Pк4* = {*V40-V38-V35-V28*}; *Pк5* = {*V40-V39-V36-V30*}.

Сформовані на сьомому такті шляхи мають вигляд (пошук від початкової вершини): *P01* = { *V1-V2-V7-V8-V14*}; *P02* = { *V1-V3-V9-V15-V20*}; *P03* = { *V1-V4-V11-V10-V16*}; *P04* = { *V1-V12-V9-V22-V28*}; *P05* = { *V1-V6-V13-V18-V23*}.

Після виконання сьомого такту маємо наступні перетини шляхів: шляхи *P02* та *Pк2* перетинаються у вершині *V20*, маємо сформований шлях *P2* = {*V1-V3-V9-V15-****V20****-V25-V32-V40*}; шляхи *P04* та *Pк4* перетинаються у вершині *V28*, маємо сформований шлях *P4* = {*V1-V12-V9-V22-****V28****-V35-V38-V40*}.

Сформовані на восьмому такті шляхи мають вигляд (пошук від кінцевої вершини): *Pк1* = { *V40-V37-V31-V24-V19*}; *Pк3* = { *V40-V33-V34-V27-V21*}; *Pк5* = { *V40-V39-V36-V30-V23*}.

Після виконання восьмого такту маємо наступний перетин шляхів: шлях *P05* та *Pк5* перетинаються у вершині *V23*, маємо сформований шлях *P5* = {*V1-V6-V13-V18-****V23****-V30-V36-V39-V40*}.

Сформовані на дев’ятому такті шляхи мають вигляд (пошук від початкової вершини): *P01* = {*V1-V2-V7-V8-V14-V19*}; *P03* = {*V1-V4-V11-V10-V16-V21*}.

Після виконання дев’ятого такту маємо наступні перетини шляхів: шлях *P01* та *Pк1* перетинаються у вершині *V19*, маємо сформований шлях *P1* = {*V1-V2-V7-V8-V14-****V19****-V24-V31-V37-V40*}; шляхи *P03* та *Pк3* перетинаються у вершині *V21*, маємо сформований шлях *P3* = {*V1-V4-V11-V10-V16-****V21****-V27-V34-V33-V40*}.

У результаті роботи алгоритму утворилося п’ять непересічних шляхів (точки зустрічі виділено жирним):

*P1* = {*V1-V2-V7-V8-V14-****V19****-V24-V31-V37-V40*};

*P2* = {*V1-V3-V9-V15-****V20****-V25-V32-V40*};

*P3* = {*V1-V4-V11-V10-V16-****V21****-V27-V34-V33-V40*};

*P4* = {*V1-V12-V9-V22-****V28****-V35-V38-V40*};

*P5* = {*V1-V6-V13-V18-****V23****-V30-V36-V39-V40*}.

Початкова та кінцева вершина мають ступені зв’язності, що дорівнюють п’ятьом, максимальна кількість непересічних шляхів дорівнює 5, цієї кількості вдалося досягти використавши алгоритм зустрічної хвилі.

У таблицях 2.2 і 2.3 наведено формування шляхів за тактами від початкової та кінцевої вершин відповідно, на рис. 2.11 наведено візуалізацію результатів.

Таблиця 2.2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T | *P01* | *P02* | *P03* | *P04* | *P05* |
| 1 | *V1-V2* | *V1-V3* | *V1-V4* | *V1-V12* | *V1-V6* |
| 3 | *V1-V2-V7* | *V1-V3-V9* | *V1-V4-V11* | *V1-V12-V9* | *V1-V6-V13* |
| 5 | *V1-V2-V7-V8* | *V1-V3-V9-V15* | *V1-V4-V11-V10* | *V1-V12-V9-V22* | *V1-V6-V13-V18* |
| 7 | *V1-V2-V7-V8-V14* | *V1-V3-V9-V15-V20*  Шлях побудовано | *V1-V4-V11-V10-V16* | *V1-V12-V9-V22-V28*  Шлях побудовано | *V1-V6-V13-V18-V23*  Шлях побудовано |
| 9 | *V1-V2-V7-V8-V14-V19*  Шлях побудовано |  | *V1-V4-V11-V10-V16-V21*  Шлях побудовано |  |  |

Таблиця 2.3

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T | *Pк1* | *Pк2* | *Pк3* | *Pк4* | *Pк5* |
| 2 | *V40-V37* | *V40-V32* | *V40-V33* | *V40-V38* | *V40-V39* |
| 4 | *V40-V37-V31* | *V40-V32-V25* | *V40-V33-V34* | *V40-V38-V35* | *V40-V39-V36* |
| 6 | *V40-V37-V31-V24* | *V40-V32-V25-V20* | *V40-V33-V34-V27* | *V40-V38-V35-V28* | *V40-V39-V36-V30* |
| 8 | *V40-V37-V31-V24-V19* | Шлях побудовано | *V40-V33-V34-V27-V21* | Шлях побудовано | *V40-V39-V36-V30-V23*  Шлях побудовано |
| 5 | Шлях побудовано |  | Шлях побудовано |  |  |



Рис. 2.11.Результат моделювання методом зустрічної хвилі

Висновки до розділу 2

Значення максимально можливої кількості непересічних шляхів між двома кінцевими вершинами залежить від потужності множини перерізу графу та ступеня зв’язності вершин.

Було запропоновано модифікований спосіб формування множини непересічних шляхів на основі хвильового алгоритму та запропоновано його оптимізацію за рахунок використання методу зустрічної хвилі, що дозволяє удвічі зменшити область пошуку непересічних шляхів. Запропонований алгоритм суттєво зменшив часову складність формування множини непересічних шляхів.

РОЗДІЛ 3

МОДЕЛЮВАННЯ ЗАПРОПОНОВАНОГО АЛГОРИТМУ

Для моделювання запропонованого алгоритму було обрано об’єктно-орієнтовану мову програмування Java. Такий вибір слід обґрунтувати наступними чинниками:

* мова Java є крос-платформною, тобто є можливим виконання вже зібраних програм, написаних на Java, незалежно від платформи/ операційної системи (зокрема, Linux, Unix, Mac OS, Windows, Android, Web);
* великий вибір інструментів та бібліотек, що можна застосувати для програмування необхідного рішення;
* наявність зручних при розробці програмних рішень інтегрованих середовищ розробки (зокрема, Intelij IDEA від компанії JetBrains);
* обрана мова є сучасною та популярною, її застосування є актуальним та корисним для покращення навичок програмування.

При розробці програми використовується інтегроване середовище розробки Intelij IDEA, встановлено Java JRE 8 та Java SDK 8.

Для візуалізації використовується бібліотека JavaFX, що входить до Java JDK8.

При роботі з програмою бажано використовувати наступні технічні характеристики комп’ютерної системи:

* процесор з тактовою частотою не менше ніж 2 ГГц;
* оперативна пам’ять об’ємом не менше ніж 1 Гб;
* встановлена віртуальна машина JVM не нижче 8 версії.

Інші характеристики не мають вагомого впливу для коректної роботи з програмою.

3.1. Опис логіки програми

Програмний продукт має надавати можливості для роботи з наступними функціями:

* створення, збереження, редагування графу мережі;
* моделювання обраного алгоритму багатошляхової маршрутизації;
* перегляд та аналіз результатів моделювання;
* можливість розширення можливостей програми за рахунок додавання інших алгоритмів маршрутизації.

При розробці рішення було зауважено, програму слід розбити на 3 шари:

* модель, що зберігає граф системи;
* візуальне відображення моделі;
* шар, що моделює алгоритм маршрутизації.

Отже, розглянемо структуру програми. Класи *Model*, *Node* та *Link* відповідають за шар моделі, клас *View* відповідає за візуальне відображення моделі, клас *Algorithm* відповідає за моделювання алгоритму.

Клас *Node* відповідає за окремий вузол графу мережі. Містить поля *id* ідентифікатору вузла, координати *x* та *y* центру вузла відображення вузла у *View* та поточне значення кольору вузла *color*, містить методи, що читають та модифікують значення цих полів.

Клас *Link* відповідає за окремий зв’язок між двома вузлами графу, містить ідентифікатори *id1* початкового та *id2* кінцевого вузлу графу, координати точок початку *x1*, *y1* та кінця *x2*, *y2* відображення зв’язку у *View* і поточне значення кольору зв’язка *color*, містить методи, що читають та модифікують значення цих полів.

Модель графу системи зберігається у класі *Model*. Має поля *modelNodes* та *modelLinks*, що є колекціями екземплярів класів *Node* та *Link* відповідно. Значення ідентифікатора наступного доданого вузла будується за автоінкрементом поля *maxIndex*, таким чином гарантується унікальність ідентифікатора.

Клас *Model* містить методи *addNode*, r*emoveNode*, *moveNode*, *addLink*, *removeLink*, що відповідають за додавання вузла, видалення вузла, пересунення вузла, додавання зв'язку між двома вузлами графу, видалення зв’язку між двома вузлами графу відповідно у моделі графу мережі. Також клас *Model* містить службові методи:

* *getNode* повертає вузол за ідентифікатором;
* *getModelLinks* повертає колекцію зв’язків моделі;
* *getModelNodes* повертає колекцію вузлів моделі;
* *getNodeSize* повертає розмір вузла на відображенні графу колекцію зв’язків моделі;
* *findLink* повертає зв’язок між двома вузлами графу, якщо такий існує;
* *getLinks* повертає колекцію зв’язків за ідентифікатором вузла, що може бути початковим або кінцевим у цих зв’язках;
* *generateLinkBounds* повертає координати точок початку та кінця зв’язків на відображенні графу.

Клас *File* містить методи *new*, *save*, *open*, що дозволяють створити новий файл, зберегти модель у файл та прочитати модель з файлу відповідно. Клас *Model* є наслідником класу *File.*

Клас *View* містить наступні контролери:

* *addNodeHandler* – відповідає за додавання нового вузла у граф;
* *removeNodeHandler* – відповідає за видалення вузла у графі;
* *pressNodeHandler*, *moveNodeHandler* – відповідають за пересування вузла у графі (реагують на натиснення на вузол та перенесення вузла відповідно);
* *addLinkHandler* – відповідає за додавання нового зв’язку між двома вузлами у графі;
* *removeLinkHandler* – відповідає за видалення зв’язку між двома вузлами у графі.

При запуску програми викликається метод start, методом *initiateMenu* ініціалізується меню програми, за що відповідають компоненти *menuBar*.

Повідомлення від контролерів передаються до моделі графу, після внесення змін у модель відбувається перемалювання графу методом *repaint*: читаються колекції *modelNodes* та *modelLinks* класу *Model*, відбувається будування вузлів та зв’язків графу.

Оскільки для додавання та видалення зв’язку між двома вузлами графу необхідно почергово натиснути на початковий та кінцевий вузол, для зберігання ідентифікаторів вузлів використовуються змінні *startNodeId* та *finishNodeId*.

Клас *Algorithm* відповідає за моделювання власне алгоритму.

Для покрокового моделювання використовуються таблиці *leftTable* та *rightTable*, де зберігаються метрики довжин шляху.

Архів Поточні значення кольорів вузлів та використання зв’язків зберігається у колекціях *nodeColor* та *linkColor*.

На кожному кроці розглядаються таблиці *leftTable* та *rightTable,* у тимчасові колекції *leftTempNode* та *rightTempNode* додається наступна хвиля вершин, серед них на основі критерію, визначеному у пункті 2.2, обираються потрібні вершини.

Клас *Algorithm* містить наступні специфічні методи:

* *importModel* – імпортує модель графу із класу *Model*;
* *step* – моделює один крок роботи алгоритму;
* *allModel* – моделює повну роботу алгоритму;
* *isCrossingReached* – визначає, чи було досягнуто перетину дерев графу для всіх шляхів;
* *getModelTimeline* – повертає хронологію моделювання;
* *getPaths* – повертає змодельовані шляхи;
* *getModelNodeColors* – повертає промодельовані кольори вузлів;
* *getModelLinkColors* – повертає промодельовані кольори зв’язків;
* *getOperationQuantity* – повертає кількість операцій моделювання;
* *setDiscipline* – змінює дисципліну моделювання.

3.2. Опис програмного інтерфейсу

Програма призначена для моделювання розробленого у розділі два алгоритму багатошляхової маршрутизації. Програмний продукт надає можливості редагування графу та власне моделювання алгоритму.

Для кращого розуміння роботи програми необхідно описати особливості інтерфейсу програми. Інтерфейс програмного застосунку наведено на рисунку 3.1.

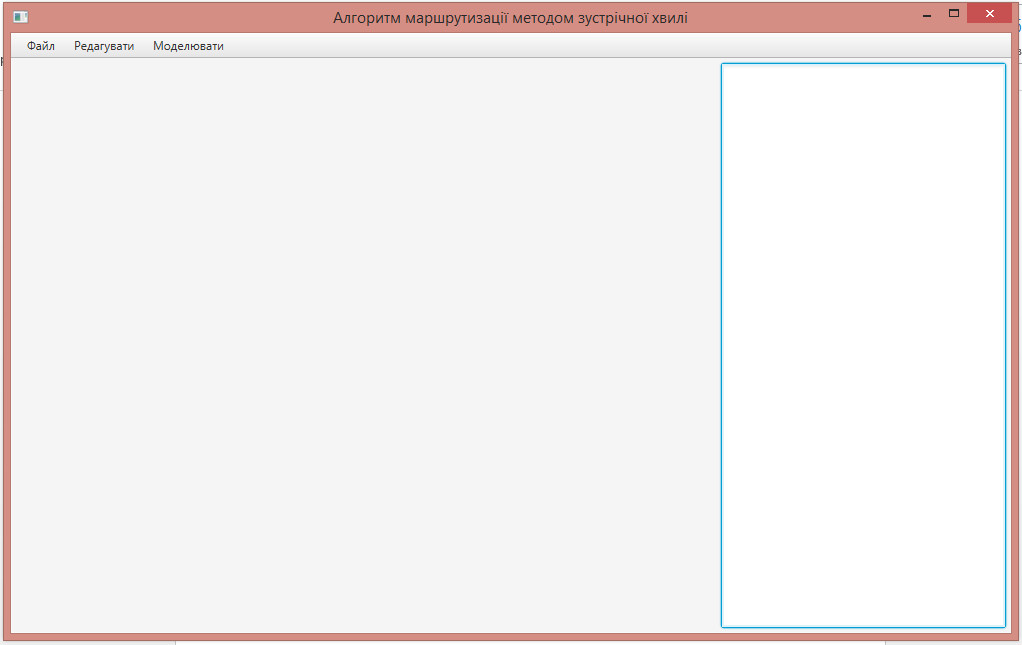


Рис. 3.1. Інтерфейс програми

Меню розробленої програми знаходиться у верхній частини головного вікна програми та складається з трьох елементів: «Файл», «Редагувати» та «Моделювати». Нижня частина головного вікна програми містить дві області: ліва область (область графу мережі) використовується для створення графу мережі та візуального відображення результатів моделювання, права область (область виводу результату) використовується для виведення результатів моделювання.

Меню «Файл» використовується для загального керування програмою та містить наступні елементи: «Створити новий», «Зберегти», «Відкрити» та «Вийти», що для зручності мають наступні гарячі клавіши: *Ctrl+N, Ctrl+S, Ctrl+O, Ctrl+E* відповідно. Гарячі клавіші обрано з огляду на простоту запам’ятовування та використання, використано одну спільну клавішу для цього меню: *Ctrl*. Елементи «Створити новий», «Зберегти», «Відкрити» та «Вийти» використовуються для створення нового графу, збереження поточного стану графу у файл, читання графу зі збереженого файлу та виходу з програми відповідно. Вигляд меню «Файл» наведено на рисунку 3.2.

Меню «Редагувати» використовується при створенні та редагуванні графу мережі і містить наступні елементи: «Додати вузол», «Видалити вузол», «Пересунути вузол», «Додати зв’язок», «Видалити зв’язок» та «Перемалювати». Ці елементи мають наступні гарячі клавіші для зручності: «1», «2», «3», «4», «5» та «6» відповідно. Вигляд меню «Редагувати» наведено на рисунку 3.3.

Редагування графу мережі виконується наступним чином.

Для розміщення нового вузла необхідно натиснути на елемент «Додати вузол» меню «Редагувати», після чого натиснути на область графу мережі. У області з’явиться вузол з центром у місці натискання. Ідентифікатор вузла генерується автоматично за інкрементом.

Для видалення необхідно натиснути на елемент «Видалити вузол» меню «Редагувати», після чого натиснути вузол на області графу мережі. При видаленні вузлу автоматично зникають і всі зв’язки, що містять цей вузол.

Для пересунення вузла необхідно натиснути на елемент «Пересунути вузол» меню «Редагувати», після чого натиснути на вузол графу та пересунути його на нове місце. Після перенесення слід натиснути на елемент «Перемалювати» меню «Редагувати», після чого перемалюються усі зв’язки перенесеного вузла.

Для додавання нового зв’язку між двома вузлами графу необхідно натиснути на елемент «Додати зв’язок» меню «Редагувати», після чого

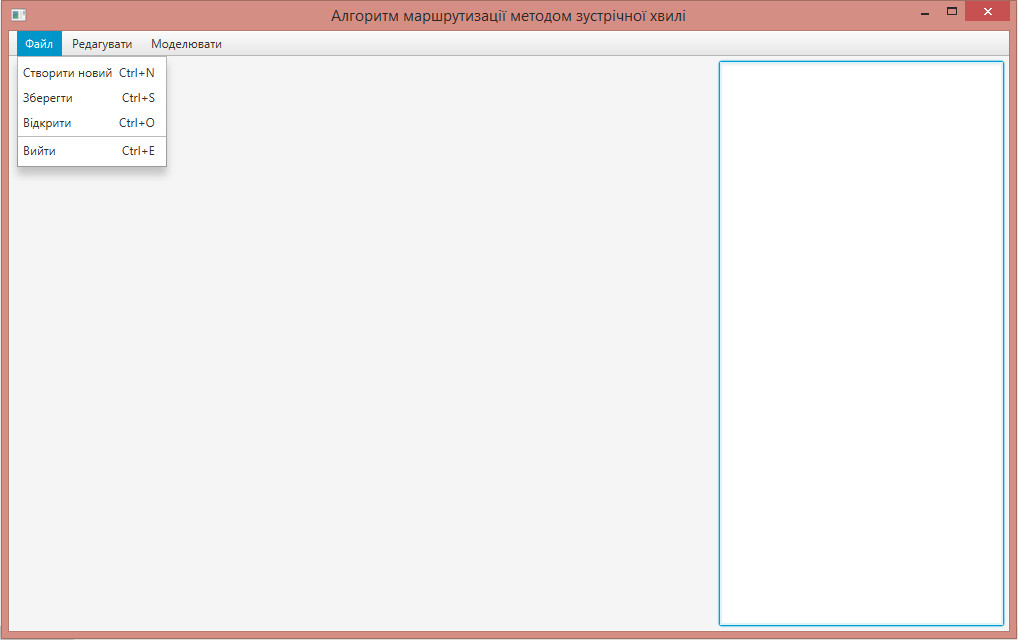


Рис. 3.2. Меню «Файл»

почергово натиснути на два вузли графу. Якщо між вузлами до цього моменту не було зв’язку, він додастся до моделі та відобразиться у області графу.

Для додавання нового зв’язку між двома вузлами графу необхідно натиснути на елемент «Додати зв’язок» меню «Редагувати», після чого почергово натиснути на два вузли графу. Якщо між вузлами до цього моменту існував зв’язок, він видалиться з моделі та зникне з області графу.

Обраний елемент, що вибрано у меню «Редагувати» є активним до вибору нового елементу або переходу до іншого меню. Таким чином, можливо додати декілька вузлів, зв’язків чи пересунути декілька вузлів тощо не змінюючи режим при редагування графу.

Меню «Моделювати» використовується для керування моделюванням алгоритму моделювання та містить наступні елементи: «Перевірити граф», «Ввести початковий та кінцевий вузли», «Зробити повне моделювання» та «Зробити крок», що для зручності мають наступні гарячі клавіши: *Shift+C*,

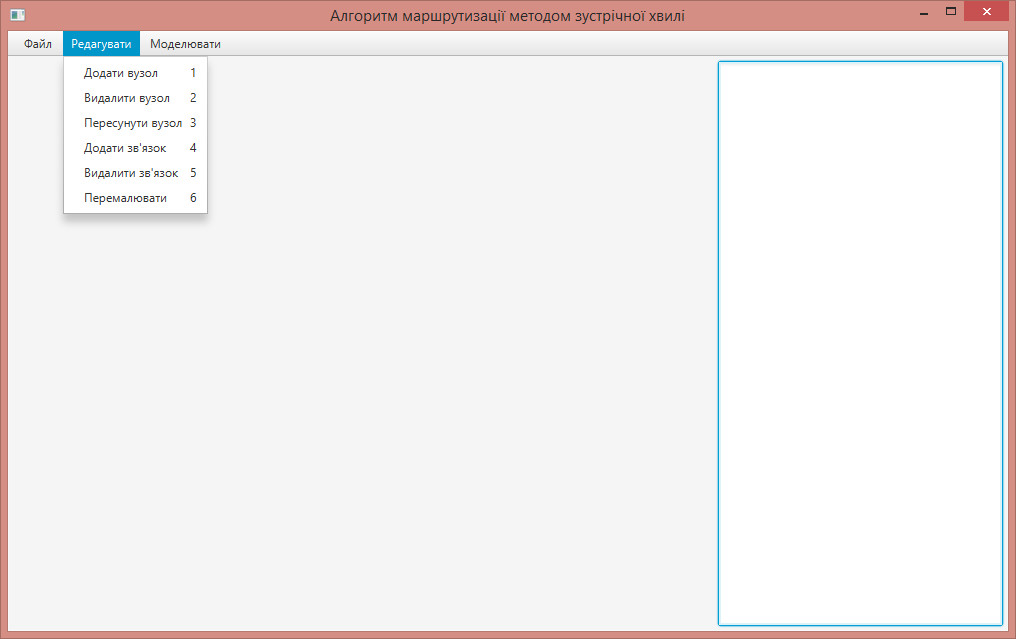


Рис. 3.3. Меню «Редагувати»

*Shift+E, Shift+W, Shift+S* відповідно. Використано одну спільну клавішу для цього меню: *Shift*. Вигляд меню «Моделювати» наведено на рисунку 3.4.

Елемент «Перевірити граф» використовується для переходу до режиму моделювання. Програма перевіряє граф на відсутність вузлів, що не пов’язані з жодним іншим вузлом графу та наявність інших помилок побудови графу.

Елемент «Ввести початковий та кінцевий вузли» використовується для вводу початкового та кінцевого вузлів маршрутизації. Після натиснення на цей елемент у випливаючому вікні потрібно буде вказати початковий вузол та вузол, до якого буде необхідно побудувати маршрут. Випливаюче вікно зображено на рис. 3.5.

При натисканні на елемент «Зробити крок» буде виконано наступний крок моделювання. На області графу мережі будуть відображені зміни, що відбулися у результаті цього кроку (зміна кольору вузла, зміна вигляду зв’язку чи додавання у область виводу результатів нового текстового рядка з інформацією. Проміжний результат роботи програми наведено на рис. 3.6.

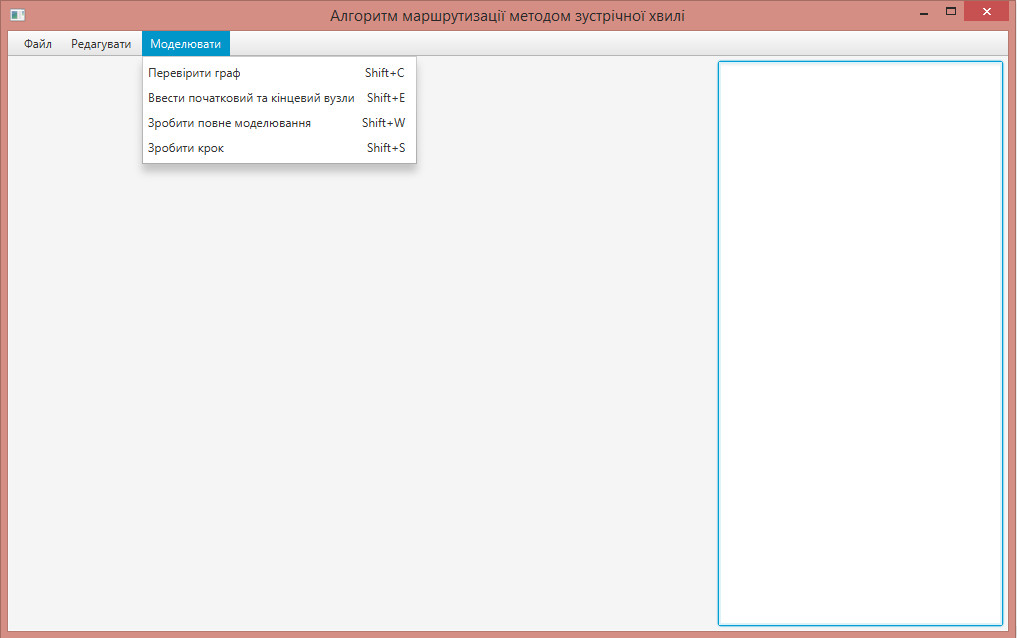


Рис. 3.4. Меню «Моделювати»

При натисканні на елемент «Зробити повне моделювання» виконається автоматичне моделювання алгоритму маршрутизації. Після закінчення роботи алгоритму у область виводу результату програми буде виведено результат роботи програми.

Є можливим на певному кроці моделювання застосувати автоматичне моделювання.

У області результатів відображається хронологія виконання маршрутизації.

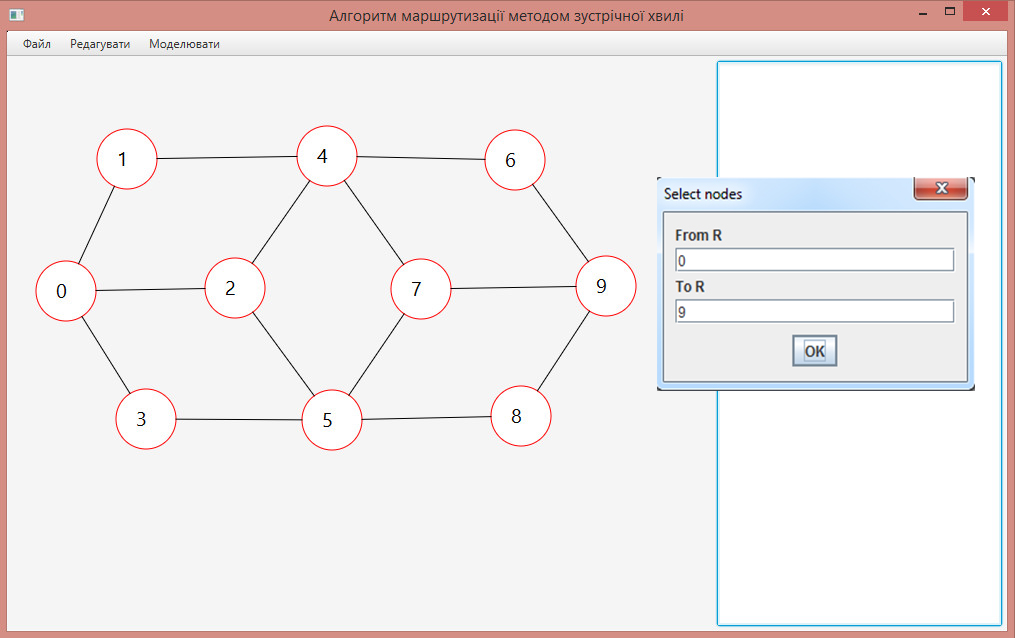


Рис. 3.5. Випливаюче вікно для вибору вершин

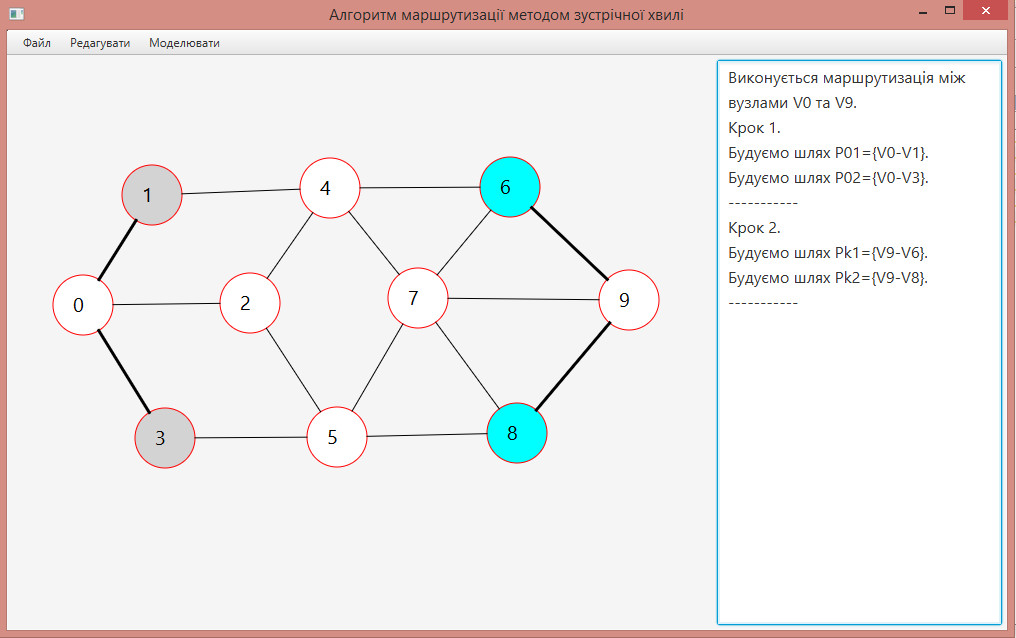


Рис. 3.6. Проміжні результати роботи програми

3.3. Розгляд прикладу роботи програми

Розглянемо роботу програми на прикладі графу, що зображено на рис. 3.7.

Спочатку треба створити граф мережі у програмі. Натискаємо елемент «Новий» у меню «Файл» для створення нового графу.

Далі за допомогою режимів «Додати вузол», «Видалити вузол», «Пересунути вузол», «Додати зв’язок», «Видалити зв’язок» та «Перемалювати» будуємо граф, зображений на рис. 3.7.

Результат введення усіх вузлів мережі та зв’язків між ними можна побачити на рис. 3.8.

Для перевірки помилок, можливих при створенні графу, використовуємо елемент «Перевірити граф» у меню «Моделювати».

Для введення завдання маршрутизації натискаємо на елемент «Ввести початковий та кінцевий вузли» у меню «Моделювати». У нашому випадку задаємо шляхи від вузла 0 до вузла 19. Випливаюче вікно можна побачити на рис. 3.9.

Далі для проведення автоматичного моделювання натискаємо на елемент «Зробити повне моделювання» у меню «Моделювати».



Рис. 3.7. Граф системи

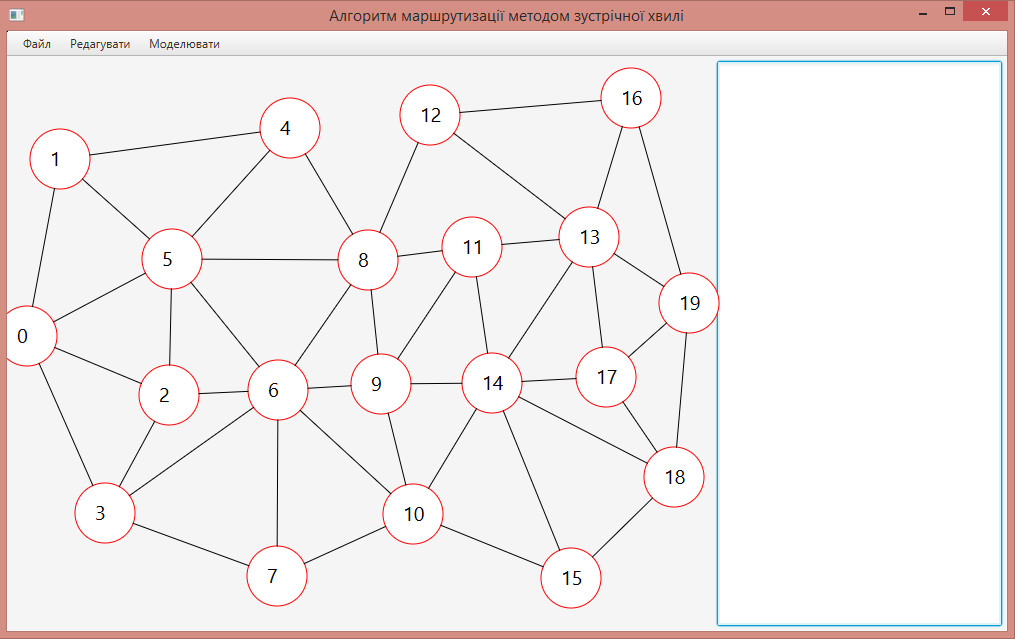


Рис. 3.8. Результати внесення графу мережі у програму

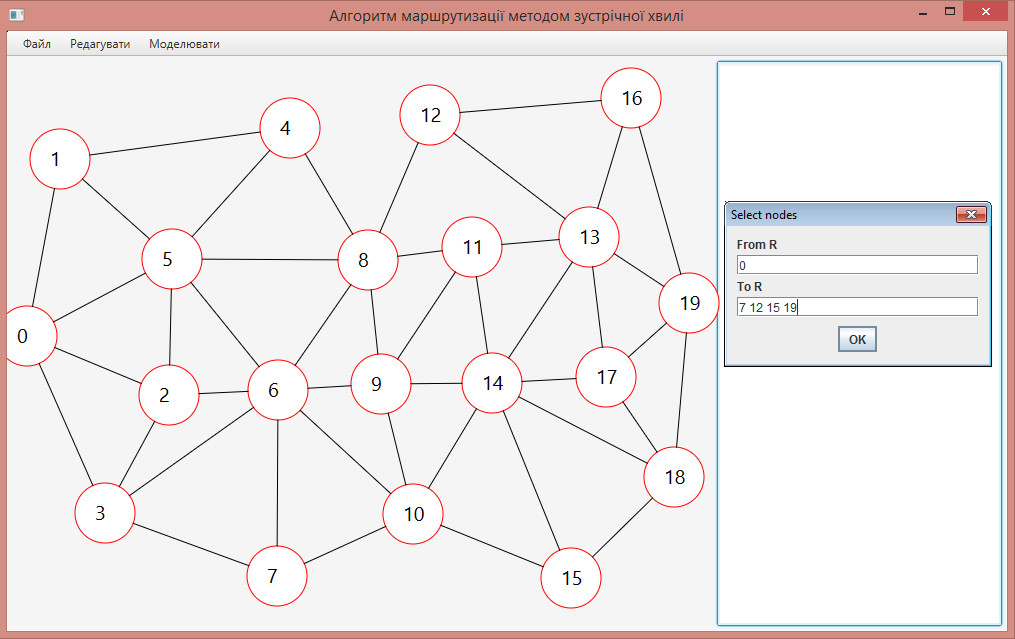


Рис. 3.9. Випливаюче вікно означення початкової та кінцевої вершини маршрутизації

Після введення всіх необхідних даних, програма виконує моделювання і виводить результат у праву частину вікна (рис. 3.10).

Розглянемо детально результат моделювання (рис. 3.11).

Маємо три сформованих шляхи до вершини R19. У полі «Кількість операцій» можна побачити кількість кроків, необхідних для проведення моделювання.

Якщо вказати конкретну пару вершин, при відображенні можна побачити зміну кольору вершин, відповідно до хвиль: сірий, червоний, зелений та синій для хвилі від початкової вершини, та блакитний, жовтий та коричневий від кінцевої вершини. Сформовані шляхи виділено жирним. Оскільки потужність перетину графу складає три, для усіх необхідних вершин маємо три шляхи. У правій області можна побачити повну хронологію моделювання, зробивши промотку елементу.

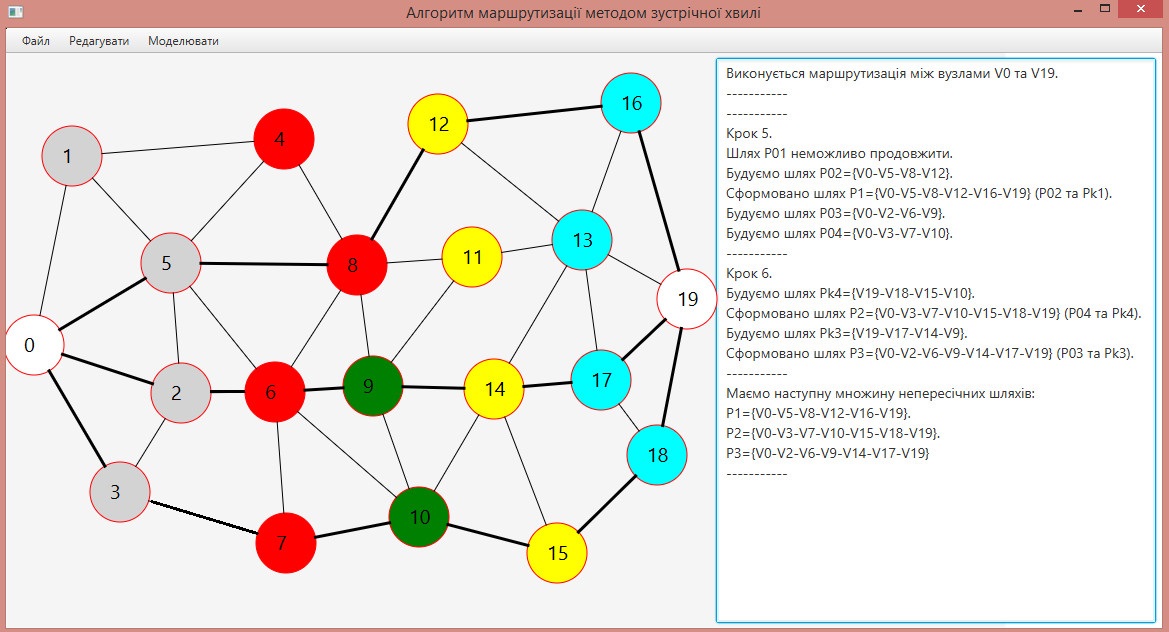


Рис. 3.10. Результати повного моделювання для заданого графу

Рис. 3.11. Результати моделювання для заданого графа у текстовій формі

Крок 6.

Будуємо шлях Pk4={V19-V18-V15-V10}.

Сформовано шлях P2={V0-V3-V7-V10-V15-V18-V19} (P04 та Pk4).

Будуємо шлях Pk3={V19-V17-V14-V9}.

Сформовано шлях P3={V0-V2-V6-V9-V14-V17-V19} (P03 та Pk3).

-----------

Маємо наступну множину непересічних шляхів:

P1={V0-V5-V8-V12-V16-V19}.

P2={V0-V3-V7-V10-V15-V18-V19}.

P3={V0-V2-V6-V9-V14-V17-V19}

-----------

3.4. Порівняння результатів запропонованого алгоритму з традиційними

Для проведення аналізу ефективності запропонованого алгоритму порівняно його результати із роботою комбінаторних алгоритмів. У якості комбінаторних алгоритмів обрано алгоритм Дейкстри та модифікований алгоритм Дейкстри [23].

У алгоритмі Дейкстри пошук нового шляху виконується методом направленого перебору, відповідно складність алгоритму є пропорційною до квадрату кількості вузлів, кожен наступний шлях визначається після видалення вузлів сформованого шляху з графу та інцидентнтих їм ребер. У якості модифікації використовується поділ графу на підграфи після знайдення чергового маршруту із наступним зменшенням області пошуку.

Для тестування роботи алгоритмів використовується метод *genGraph(number)*, що генерує граф із заданою кількістю вузлів та випадковим розміщенням ребер між вузлами без наявності дублюючих ребер.

Результати експериментів наведено на рис. 3.12 – 3.16. Результати класичного алгоритму Дейкстри наведено на графіках у білих стовпчиках, модифікованого алгоритму Дейкстри у світло-сірих, алгоритму зустрічної хвилі у темно-сірих стовпчиках. Під час кожної серії експериментів проведено 10 дослідів, наведено усереднені дані.

На рис. 3.12. наведено результати роботи алгоритмів щодо знайденої кількості непересічних шляхів. Бачимо, що для алгоритму зустрічної хвилі кількість знайдених шляхів більша, що підтверджує ефективність використання запропонованого алгоритму.

Рис. 3.12. Графік залежності кількості знайдених непересічних шляхів від розмірності графу

На рис. 3.13 та 3.14 наведено результати роботи алгоритмів щодо виконаної кількості операцій на графах невеликої та великої розмірності відповідно. Бачимо, що кількість операцій для алгоритмів Дейкстри суттєво перевищує кількість операцій алгоритму зустрічної хвилі, що пояснюється наявністю перебору при побудові нових шляхів. Результати свідчать про доречність використання запропонованого алгоритму для побудови множин непересічних шляхів.

Рис. 3.13. Графік залежності кількості операцій на знайдення непересічних шляхів від розмірності графу (для графів невеликої розмірності)

Рис. 3.14. Графік залежності кількості операцій на знайдення непересічних шляхів від розмірності графу (для графів великої розмірності)

На рис. 3.15 наведено результати роботи алгоритмів щодо максимальної довжини непересічних шляхів. Результати свідчать, що шляхи, запропоновані

алгоритмом зустрічної хвилі є довшими за шляхи, запропоновані класичним та модифікованим алгоритмами Дейкстри. Це пояснюється процедуру вибору наступної вершини для запропонованого алгоритму: наступною обирається вершина із найменшим значенням зовнішнього ступеня. Тому постає проблема наявності невеликої затримки при доставці повідомлень після маршрутизації, проведеної методом зустрічної хвилі.

Цей недолік можливо трохи зменшити, корегуючи знайдені маршрути після маршрутизації у бік знайдення більш коротких ділянок, де це є можливим. У якості оптимізації запропоноваго алгоритму на рис. 3.16 наведено результати роботи подібної модифікації.

Рис. 3.15. Графік залежності максимальної довжини шляху від розмірності графу

Рис. 3.16. Графік залежності максимальної довжини шляху для алгоритму зустрічної хвилі від розмірності графу

Висновки до розділу 3

Було розроблено програмну реалізацію запропонованого у попередньому розділі алгоритму формування множини непересічних шляхів методом зустрічної хвилі між вершинами графа мережі за допомогою інструментів та бібліотек мови програмування Java.

Програма надає можливість створення та редагування графів мережі, зберігання та читання графів з пам’яті. Графічне відображення графа покращую сприйняття роботи алгоритму за рахунок вдалої візуалізації.

Програма підтримує можливість моделювання множини непересічних шляхів між зазначеними вершинами, реалізовано можливість моделювання в автоматичному та покроковому режимі.

Було наведено детальний опис логіки та інтерфейсу програми, розглянуто детальний приклад роботи програми для заданого графу. В результаті роботи отримано усі можливі непересічні шляхи між початковою та кінцевою вершинами графа.

Було проведено порівняльний аналіз ефективності використання запропонованого алгоритму маршрутизації методом зустрічної хвилі в порівнянні з класичним та модифікованим алгоритмами Дейкстри.

Показано, що запропонований алгоритм дозволяє істотно скоротити часову складність формування множини непересічних шляхів мережі, що є суттєвою перевагою порівняно з існуючими способами.

ВИСНОВКИ

Роботу присвячено вирішенню задачі багатошляхової маршрутизації в мережі MPLS на основі формування багатоканальних віртуальних шляхів, що містять множину непересічних шляхів.

Було проведено теоретичний огляд та аналіз особливостей технології MPLS. Виявлено наступні переваги використання технології MPLS: масштабованість мережі; можливість роботи незалежно від структур другого та третього рівня моделі OSI; наявність стеку міток; можливість ізолювання трафіку один від одного на другому рівні моделі OSI.

Також було розглянуто існуючі протоколи маршрутизації, виокремлено основні переваги використання багатошляхової маршрутизації як одного з ефективних методів підвищення надійності та швидкості передачі даних в комп’ютерних мережах. Підтверджено необхідність розробки нового підходу до вирішення питання маршрутизації через неефективну роботу існуючих алгоритмів.

У якості базового алгоритму формування множини непересічних шляхів обрано хвильовий алгоритм через високу часову складність комбінаторних алгоритмів багатошляхової маршрутизації. У якості критеріїв обрано довжину шляху та зовнішній ступінь зв’язності вершин. За рахунок виключення процедури перебору та за рахунок використання принципу зустрічної хвилі було суттєво зменшено часову складність обчислення множини непересічних шляхів. Використання запропонованого рішення дозволяє рівномірно завантажити канали передачі даних та покращити надійність доставки повідомлень.

Для моделювання запропонованого алгоритму було розроблено програму. Програма надає можливості моделювання та візуалізації алгоритму, реалізовано можливість проведення моделювання в автоматичному та покроковому режимі.

При проведенні порівняльного аналізу запропонованого алгоритма із класичним та модифікованим алгоритмами Дейкстри отримано підтвердження, що запропонований алгоритм має кращу ефективність та має значно меншу часову складність формування множини непересічних шляхів ніж комбінаторні алгоритми.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ:

1. Luc De Ghein. MPLS Fundamentals [Електронний ресурс] / Luc De Ghein // Cisco Press. – 2006. – Режим доступу до ресурсу: http://www.ciscopress.com/store/mpls-fundamentals-9781587051975.

2. Кулаков Ю. О. Комп'ютерні мережі. Підручник / Ю. О. Кулаков, Г. М. Луцький. – Київ: Юніор, 2005.

3. Multiprotocol Label Switching [Електронний ресурс] // Wikipedia – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/ Multiprotocol\_Label\_Switching.

4. Преимущества и недостатки технологии MPLS [Електронний ресурс] // АНС «СибАК». – 2014. – Режим доступу до ресурсу: https://sibac.info/studconf/tech/xix/37782.

5. Лекция 9: Технология MPLS [Електронний ресурс] // «Национальный Открытый Университет «ИНТУИТ». – 2011. – Режим доступу до ресурсу: http://www.intuit.ru/studies/courses/1150/157/lecture/28716.

6. Label Advertisement Discipline for LDP Forwarding Equivalence Classes (FECs) [Електронний ресурс] // The Internet Engineering Task Force (IETF®). – 2014. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc7358.

7. The Use of Entropy Labels in MPLS Forwarding [Електронний ресурс] // The Internet Engineering Task Force (IETF®). – 2012. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc6790.

8. MPLS Label Stack Encoding [Електронний ресурс] // The Internet Engineering Task Force (IETF®). – 2001. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc3032.

9. LDP Specification [Електронний ресурс] // The Internet Engineering Task Force (IETF®). – 2007. – Режим доступу до ресурсу: https://tools.ietf.org/html/rfc5036.

10. Кулаков Ю.О., Луцький Г.М. Комп’ютерні мережі: Підручник за редакцією Ю.С. Ковтанюка – Київ.: Видавництво «Юніор», 2005. – 397с., іл.

11. Маршрутизація: мета, основні задачі й протоколи – Електронні засоби навчання. [Електронний ресурс]. – Режим доступу: URL: http://www.znani-us.com/3820.html (Дата звернення 09.02.2016).

12. Олифер В.Г., Олифер Н.А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 3-е изд. – СПб: Питер, 2006. – 958 с.: ил.

13. Costin R. Practical Congestion Control for Multipath Transport Protocols / Costin Raiciu., 2009.

14. BGP-XM: BGP eXtended Multipath for transit Autonomous Systems [Електронний ресурс] // Universidad Carlos III de Madrid. – 2012. – Режим доступу до ресурсу: http://www.sciencedirect.com/science/article/pii/ S1389128612003957.

15. Шувалов В.П. Классификация методов многопутевой маршрутизации [Електронний ресурс] / Шувалов В.П // T-Comm. – 2014. – Режим доступу до ресурсу: http://openarchive.nure.ua/bitstream/document/2589/1/4745.pdf.

16. Бертсекас Д., Галлагер Р., Сети передачи данных: Пер.с англ. - М.:Мир 1989. - 544с.

17. Routing Information Protocol [Електронний ресурс]: RFC: 2453 November 1998. – Режим доступу: URL : – https://www.ietf.org/rfc/rfc2453.txt (дата звернення 23.02.2016)

18. Столлингс В. Современные компьютерные сети. 2-е изд. - СПб: Питер, 2003. - 783с.

19. Диброва М.А. Способ конструирования трафика в grid системах / Диброва М.А. Коган А.В. Куценко В. А. // Вісник НТУУ “КПІ”. Інформатика, управління та обчислювальна техніка: збірник наукових праць. – Київ: Век+, – 2014. – Реферується наукометричними базами даних: Російський індекс наукового цитування (РІНЦ), Directory of Open Access Journals (DOAJ), Google Scholar

20. Диброва М.А. Способ конструирования трафика в grid системах / Диброва М.А. Коган А.В. Куценко В. А. // Вісник НТУУ “КПІ”. Інформатика, управління та обчислювальна техніка: збірник наукових праць. – Київ: Век+, – 2014. – Реферується наукометричними базами даних: Російський індекс наукового цитування (РІНЦ), Directory of Open Access Journals (DOAJ), Google Scholar

21. Lemeshko A.V. Research on Tensor Model of Multipath Routing in Telecommunication Network with Support of Service Quality by Greate Number of Indices / Lemeshko A.V., Evseeva O.Yu., Garkusha S.V. // Telecommunications and Radio Engineering, Volume 73, No. 15, January 2014, p. 1339-1360.

22. Кулаков Ю.А. Разработка и моделирование процесса безопасной многопутевой передачи информации в мобильных сетях / Кулаков Ю.А., Коган А.В., Пирогов А. А. // Вісн. Національного техн. ун-ту України “КПИ”: Інформатика, управління та обчислювальна техніка, К.: ТОВ “ВЕК+”, Випуск 54, 2011, С. 145-149.

23. Кулаков Ю.А Формирование оптимальных маршрутов в мобильных сетях на основе модифицированного алгоритма Дейкстры / Кулаков Ю.А., Воротников В.В. // Вісн. Національного техн. ун-ту України “КПИ”: Інформатика, управління та обчислювальна техніка, К.: ТОВ “ВЕК+”, Випуск 56, 2012, С. 13-19.