

Міністерство освіти і науки України
Державний заклад
«Луганський національний університет імені Тараса Шевченка»

Навчально-науковий інститут математики та інформаційних технологій

Кафедра інформаційних технологій та систем

Коцюба Микола Дмитрович

РОЗРОБКА ДОДАТКА ДЛЯ МОДЕЛЮВАННЯ ЛОКАЛЬНОЇ МЕРЕЖІ

кваліфікаційна робота

здобувача вищої освіти другого (магістерського) рівня

освітньої програми «Мультимедійні системи»

за спеціальністю 121 Інженерія програмного забезпечення

Особистий підпис _____ Микола КОЦЮБА

Науковий керівник _____ Геннадій МОГИЛЬНИЙ,
кандидат технічних наук, доцент
кафедри інформаційних технологій
та систем

Завідувач кафедри _____ Микола СЕМЕНОВ,
кандидат педагогічних наук, доцент
кафедри інформаційних технологій
та систем

Полтава – 2025

АНОТАЦІЯ

Коцюба М. Д.

Тема: Розробка додатка для моделювання локальної мережі.

Спеціальність: 121 «Інженерія програмного забезпечення».

Установа: ЛНУ імені Тараса Шевченка, 2025р.

Магістерська робота містить: 72 с., 26 рис., 3 табл., 23 джерел.

Об'єкт дослідження – модель роботи IP мереж.

Предмет дослідження – підходи до автоматизації процесу моделювання IP мереж.

Мета роботи - провести аналіз загальних підходів до створення моделі IP мереж та розробити її програмну реалізацію для технології Ethernet.

Результати роботи. У роботі розглянуті наступні системи моделювання: BONEs, Netmaker, Optimal Perfomance, Prophecy, CANE, COMNET. Проведено аналіз цих систем. Розроблені наступні алгоритми роботи системи: роботи ПК, роботи роутера, роботи хаба та відстеження колізій. Також виконано тестування всієї системи. У роботі покроково змодельована мережа за заданою топологією, та проаналізована робота кожного вузла, та всієї системи в цілому. Розглядається програмна реалізація додатку для моделювання локальної мережі.

Висновок. В результаті розробки було розроблено додаток для моделювання локальної мережі.

Ключові слова: СИСТЕМИ МОДЕЛЮВАННЯ, ТЕХНОЛОГІЇ ETHERNET, РОУТЕР, IP АДРЕС, QT SCRIPT.

ANNOTATION

Kotsiuba Mykola

Theme: Development of an application for local network modeling.

Speciality: 121 "Software Engineering".

Institution: Luhansk Taras Shevchenko National University (LTSNU), 2025 year.

Master's work of: 72 p., 26 im, 3 table, 23 sources.

A research object is IP network operating model.

The article of research is approaches to automating the process of modeling IP networks.

An aim of work is to analyze general approaches to creating an IP network model and develop its software implementation for Ethernet technology.

Job performances. The following modeling systems are considered in the work: BONeS, Netmaker, Optimal Performance, Prophecy, CANE, COMNET. An analysis of these systems is carried out. The following algorithms for the system operation are developed: PC operation, router operation, hub operation and collision tracking. The entire system is also tested. The work step-by-step simulates the network according to the given topology, and analyzes the operation of each node and the entire system as a whole. The software implementation of the application for modeling a local network is considered.

Conclusion. As a result of the development, an application for modeling a local network was developed.

Keywords. MODELING SYSTEMS, ETHERNET TECHNOLOGIES, ROUTER, IP ADDRESS, QT SCRIPT.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. ОГЛЯД СИСТЕМ МОДЕЛЮВАННЯ КОМП'ЮТЕРНИХ МЕРЕЖ.....	9
1.1. Загальні відомості про системи моделювання комп'ютерних мереж	9
1.2. Методи моделювання комп'ютерних мереж.....	11
1.3. Системи імітаційного моделювання комп'ютерних мереж.....	13
Висновки до розділу	20
РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНИХ РЕАЛІЗАЦІЙ АЛГОРИТМІВ	22
2.1. Технологія Gigabit Ethernet	22
2.1.1. Загальні положення.....	22
2.1.2. Архітектура стандарту Gigabit Ethernet	23
2.1.3. Інтерфейс 1000Base -X	25
2.1.4. Інтерфейс 1000Base-T	27
2.1.5. Рівень MAC	28
2.1.6. Специфікації фізичного середовища стандарту 802.3z.....	33
2.1.7. 10 Gigabit Ethernet	34
2.1.8. Сімейство 40GBASE-R	35
2.1.9. 100-Gigabit Ethernet (100-GE)	37
2.1.10. Використання технології Ethernet для побудови мультисервісних мереж	41
2.2. Огляд та обґрунтування вибору середовища розробки	48
2.3. Елементи програми	53
2.4. Модулі програми	55
Висновки до розділу	59
РОЗДІЛ 3. РОЗРОБКА ТА АНАЛІЗ СИСТЕМИ МОДЕЛЮВАННЯ ЛОКАЛЬНОЇ МЕРЕЖІ.....	60

3.1. Розробка системи моделювання	60
3.2. Аналіз роботи системи моделювання	64
Висновки до розділу	68
ВИСНОВКИ	69
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	71
ДОДАТОК А	73
ДОДАТОК Б	110

ВСТУП

З розвитком мережевої інфраструктури та з ускладненням додатків, що застосовуються в мережі, збільшуються вимоги до пропускної здатності, надійності і захисту мережі, її керованості, зниженню вартості експлуатації.

Для задоволення сучасних вимог до комп'ютерної мережі вона повинна підтримувати такі мережеві програми та сервіси:

- інтегровану передачу голосових, відео - і цифрових даних;
- створення віртуальних локальних і приватних мереж;
- управління мережею на основі правил;
- використання угод про рівень послуг, що надаються;
- облік використовуваних ресурсів;
- управління користувачами;
- передачу багатоадресного трафіку;
- побудову мереж Internet, Intranet, Extranet.

Таким чином, комп'ютерні мережі надають широкий вибір послуг і механізмів для забезпечення роботи необхідних додатків за допомогою реалізації служби каталогів, різноманітних мережевих сервісів (комутація, віртуальні локальні та приватні мережі, мережі Frame Relay і АТМ, правила доступу, захист інформації, якість обслуговування, облік використання ресурсів) та постійно ускладнюються. У зв'язку з цим, процес проєктування комп'ютерних мереж також суттєво ускладнюється. В теперішній час найбільш розповсюдженим підходом до їх проєктування є експертні оцінки. Спеціалісти в галузі інформаційних систем проєктують комп'ютерну мережу у відповідності до поставлених їй задач. Але рішення цих спеціалістів має суб'єктивний характер і не завжди є оптимальним.

Застосування сучасних методів обробки і передачі інформації, перш за все, в режимі реального часу, суттєво змінює вимоги до архітектури комп'ютерних мереж. Збільшується кількість і якість вузлів такої мережі, підвищуються вимоги

до послуг, що надаються, ускладняється характер запитів, зростає значимість моніторингу і змінюється характер адміністрування цих мереж.

Для успішного створення таких мереж на етапі проектування виникає необхідність їх моделювання.

Вибір того або іншого засобу моделювання комп'ютерної мережі визначається задачами, які вирішуються за допомогою моделювання, типом та архітектурою мережі, що моделюється, а також бюджетом, виділеним на проектування цієї мережі. На даний момент не існує універсального середовища моделювання. Виходячи з цього, створення таких середовищ є актуальною науковою задачею.

Актуальність теми цієї роботи обумовлюється необхідністю ефективної побудови і використання корпоративних інформаційних систем, особливо в умовах недостатнього фінансування інформаційних технологій на підприємствах [2]. Практична значимість роботи визначається розробкою реального програмного засобу, що служитиме для автоматизації створення моделей мереж.

На даний момент розроблюється велика кількість програм моделювання мереж. Але ця система розробляється для впровадження її у сферу освіти, а точніше для залучення її до освітнього процесу, тому ця система є унікальною, чим обумовлюється її інноваційна цінність.

Об'єкт дослідження – модель роботи IP мереж.

Предмет дослідження – підходи до автоматизації процесу моделювання IP мереж.

Мета роботи - провести аналіз загальних підходів до створення моделі IP мереж та розробити її програмну реалізацію для технології Ethernet.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

1. Провести огляд існуючих систем моделювання комп'ютерних мереж.
2. Провести аналіз основних методів моделювання комп'ютерних мереж.

3. Розробити систему моделювання локальної мережі.

Методи дослідження - техніко-економічний з використанням комп'ютерних технологій, технічний аналіз, методи моделювання інформаційних процесів.

В першому розділі докладно розглянуто наступні системи моделювання: BONEs, Netmaker, Optimal Perfomance, Prophecy, CANE, COMNET. Проведено аналіз цих систем.

У другому та третьому розділі були розроблені наступні алгоритми роботи системи: роботи ПК, роботи роутера, роботи хаба та відстеження колізій. Також виконано тестування всієї системи. У цьому розділі покроково змодельована мережа за заданою топологією, та проаналізована робота кожного вузла, та всієї системи в цілому.

РОЗДІЛ 1. ОГЛЯД СИСТЕМ МОДЕЛЮВАННЯ КОМП'ЮТЕРНИХ МЕРЕЖ

1.1. Загальні відомості про системи моделювання комп'ютерних мереж

Жоден проєкт великої мережі зі складною топологією, в даний час, не обходиться без вичерпного моделювання майбутньої мережі. Програми, що виконують це завдання, досить складні і дороги. Метою моделювання є визначення оптимальної топології, адекватний вибір мережевого устаткування, визначення робочих характеристик мережі і можливих етапів майбутнього розвитку. Адже мережа, дуже точно оптимізована для рішень завдань поточного моменту, може зажадати серйозних переробок в майбутньому. Моделювання використовується для дослідження характеристик нових модифікацій протоколів, методи роботи з чергами, для оптимізації параметрів якості обслуговування.

На моделі можна випробувати вплив сплесків завантаження, вплив великого потоку широкомовних запитів або реалізувати режим колапсу (для Ethernet), що навряд чи хтось може собі дозволити в працюючій мережі [15]. У процесі моделювання з'ясовуються наступні параметри:

- граничні пропускні спроможності різних фрагментів мережі і залежності втрат пакетів від завантаження окремих станцій і зовнішніх каналів;
- час відгуку основних серверів в самих різних режимах, у тому числі таких, які в реальній мережі укрив небажані. Оптимізація конфігурації, наприклад, поштового або WEB-сервера;
- вплив установки нових серверів на перерозподіл інформаційних потоків (Proxy, Firewall і т.д.);
- рішення оптимізації топології при виникненні вузьких місць в мережі (розміщення серверів, DNS, зовнішніх шлюзів, організація опорних каналів тощо);

- вибір того чи іншого типу мережного обладнання (наприклад, 10BaseTX, 100BaseFX, GE або 10GE) або режиму його роботи (наприклад, cut-through, store-and-forward для мостів і перемикачів і т.д.);
- вибір внутрішнього протоколу маршрутизації і його параметрів (наприклад, метрики);
- визначення гранично допустимого числа користувачів того або іншого сервера;
- оцінка необхідної смуги пропускання зовнішнього каналу для забезпечення необхідного рівня QOS. Вибір і оптимізація параметрів системи обслуговування черг;
- оцінка впливу мультимедійного трафіку на роботу локальної мережі, наприклад, при підготовці відеоконференцій;
- вибір протоколів та схеми опорної мережі сервіс-провайдера і т.д.

Перераховані завдання пред'являють різні вимоги до програм. В одних випадках достатньо провести моделювання на фізичному (MAC) рівні, в інших потрібний вже рівень транспортних протоколів (наприклад, UDP і TCP), а для найбільш складних завдань потрібно відтворити поведінку прикладних програм. Все це повинно прийматися до уваги при виборі або розробці моделюючої програми. Адже потрібно врахувати, що ваша машина повинна в тій чи іншій мірі відтворити дії всіх машин в модельованій мережі. Таким чином, машина ця повинна бути досить швидкодіючою, і, незважаючи на це, моделювання однієї секунди роботи мережі може зайняти при певних умовах не одну годину.

Задачі моделювання обробки черг у віртуальному каналі з метою оцінки параметрів якості обслуговування вимагають не настільки значних ресурсів.

Але з урахуванням перебору можливих значень параметрів конфігурації це також може зайняти багато годин.

Ефективність побудови і використання корпоративних інформаційних систем стала надзвичайно актуальним завданням, особливо в умовах недостатнього фінансування інформаційних технологій на підприємствах.

Критеріями оцінки ефективності можуть служити зниження вартості реалізації інформаційної системи, відповідність поточним вимогам і вимогам найближчого часу, можливість і вартість подальшого розвитку і переходу до нових технологій.

Аналізатори протоколів незамінні для дослідження реальних мереж, але вони не дозволяють отримувати кількісні оцінки характеристик, для ще не існуючих мереж, що знаходяться в стадії проєктування. У цих випадках проєктувальники можуть використовувати засоби моделювання, за допомогою яких розробляються моделі, які відтворюють інформаційні процеси, що протікають в мережах [14].

1.2. Методи моделювання комп'ютерних мереж

Моделювання являє собою потужний метод наукового пізнання, при використанні якого досліджуваний об'єкт замінюється більш простим об'єктом, званим моделлю. Основними різновидами процесу моделювання можна вважати два його види - математичне та фізичне моделювання. При фізичному (натурному) моделюванні досліджувана система замінюється відповідної їй іншої матеріальної системою, яка відтворює властивості, що вивчається системи із збереженням їх фізичної природи. Прикладом цього виду моделювання може служити пілотна мережа, за допомогою якої вивчається принципова можливість побудови мережі на основі тих чи інших комп'ютерів, комунікаційних пристроїв, операційних систем і додатків [4].

Можливості фізичного моделювання досить обмежені. Воно дозволяє вирішувати окремі завдання при завданні невеликої кількості сполучень досліджуваних параметрів системи. Дійсно, при натурному моделюванні обчислювальної мережі практично неможливо перевірити її роботу для варіантів з використанням різних типів комунікаційних пристроїв - маршрутизаторів,

комутаторів і т.п. Перевірка на практиці близько десятка різних типів маршрутизаторів пов'язана не тільки з великими зусиллями і тимчасовими витратами, але і з чималими матеріальними витратами.

Але навіть і в тих випадках, коли при оптимізації мережі змінюються не типи пристроїв та операційних систем, а лише їх параметри, проведення експериментів у реальному масштабі часу для величезної кількості всіляких поєднань цих параметрів практично неможливо за осяжне час. Навіть проста зміна максимального розміру пакета в будь-якому протоколі вимагає переконфігурування операційної системи в сотнях комп'ютерів мережі, що вимагає від адміністратора мережі проведення дуже великої роботи.

Тому, при оптимізації мереж у багатьох випадках кращим виявляється використання математичного моделювання. Математична модель являє собою сукупність співвідношень (формул, рівнянь, нерівностей, логічних умов), що визначають процес зміни стану системи в залежності від її параметрів, вхідних сигналів, початкових умов і часу.

Особливим класом математичних моделей є імітаційні моделі. Такі моделі являють собою комп'ютерну програму, яка крок за кроком відтворює події, що відбуваються в реальній системі. Стосовно до обчислювальних мереж їх імітаційні моделі відтворюють процеси генерації повідомлень додатками, розбиття повідомлень на пакети і кадри певних протоколів, затримки, пов'язані з обробкою повідомлень, пакетів і кадрів усередині операційної системи, процес отримання доступу комп'ютером до поділюваного мережевому середовищі, процес обробки вступників пакетів маршрутизатором і т.д. При імітаційному моделюванні мережі відсутня потреба в придбанні дорогого обладнання, оскільки його функціональність імітується програмним забезпеченням, яке точно відтворює основні характеристики і параметри такого обладнання.

Однією з головних переваг імітаційного моделювання є можливість замінити реальний процес зміни подій у системі на прискорений, синхронізований із роботою програми. Завдяки цьому всього за кілька хвилин

можна змодельовати функціонування мережі, що у реальності тривало б кілька днів. Це дозволяє оцінити роботу мережі в широкому діапазоні змін параметрів.

Результатом роботи імітаційної моделі є статистичні дані, зібрані під час спостереження за подіями. До них належать такі важливі характеристики мережі, як час реакції, коефіцієнти використання каналів і вузлів, імовірність втрати пакетів тощо.

Для полегшення процесу створення програмної моделі використовуються спеціалізовані мови імітаційного моделювання, які є зручнішими порівняно з універсальними мовами програмування. Прикладами мов імітаційного моделювання можуть служити такі мови, як SIMULA, GPSS, SIMDIS. Існують також системи імітаційного моделювання, які орієнтуються на вузький клас систем, що вивчаються і дозволяють будувати моделі без програмування. Подібні системи для обчислювальних мереж розглядаються нижче.

1.3. Системи імітаційного моделювання комп'ютерних мереж

Процес створення моделей комп'ютерних мереж можна автоматизувати за допомогою спеціальних програм. Ці програми беруть на себе рутинну роботу з побудови моделі, аналізуючи вхідні дані про топологію мережі, протоколи, навантаження та інше обладнання. Завдяки цьому користувачеві залишається лише налаштувати основні параметри моделі [4].

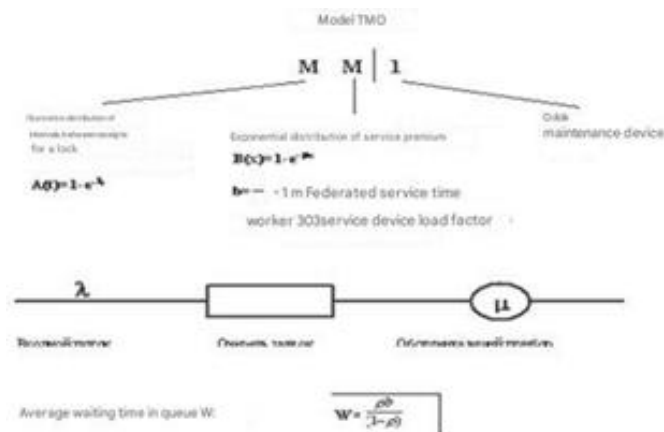


Рис. 1.1 - Застосування моделі теорії масового обслуговування М/М/1 для аналізу трафіку в мережі Ethernet

Якість результатів моделювання значною мірою залежить від точності вихідних даних про мережу, переданих у систему імітаційного моделювання.

Програмні засоби моделювання мереж є корисним інструментом для адміністраторів корпоративних мереж, особливо на етапі проєктування нової мережі або внесення значних змін до вже існуючої. Ці програми дозволяють оцінити наслідки впровадження різних рішень ще до придбання обладнання, що мінімізує ризики помилок.

Хоча більшість таких програм досить дорогі, потенційна економія від їх використання може бути суттєвою. Імітаційні моделі мереж використовують дані про просторову структуру мережі, кількість вузлів, конфігурацію з'єднань, швидкість передачі даних, протоколи, типи обладнання, а також про виконувані у мережі додатки.

Зазвичай моделі не створюються з нуля, адже існують готові імітаційні моделі для основних елементів мереж: популярних типів маршрутизаторів, каналів зв'язку, методів доступу, протоколів тощо. Ці моделі формуються на основі результатів тестування реальних пристроїв, аналізу їх роботи або теоретичних розрахунків. У результаті створюється бібліотека типових мережевих елементів, які можна налаштовувати за допомогою вбудованих параметрів.

Системи імітаційного моделювання також забезпечені інструментами для підготовки вихідних даних про мережу, такими як засоби попередньої обробки топології мережі та аналізу трафіку. Це особливо корисно, якщо модельована мережа є модифікацією існуючої, де можна провести реальні вимірювання. Крім того, системи зазвичай включають засоби статистичної обробки отриманих результатів моделювання.

Систем динамічного моделювання обчислювальної системи достатньо багато, вони розробляються в різних країнах. Вдалося знайти такі системи, вироблені в Румунії та інших країнах, які не є лідерами комп'ютерно-інформаційної індустрії. Крім того, часто розвинені системи діагностування

встановленої обчислювальної системи (інтелектуальні кабельні тестери, сканери, аналізатори протоколів) також зараховують до систем моделювання, що не відповідає дійсності. Класифікуємо системи за двома пов'язаним критеріями: ціна і функціональні можливості. Як і слід було очікувати, функціональні можливості систем моделювання жорстко пов'язані з їх ціною. Аналіз пропонованих на ринку систем показує, що динамічне моделювання обчислювальних систем - справа вельми дорога.

Щоб отримати реальну картину в обчислювальній системі - платіть гроші. Всі системи динамічного моделювання можуть бути розбиті на дві цінові категорії:

- Дешеві (сотні і тисячі доларів).
- High-end (десятки тисяч доларів, у повному варіанті - сто і більше тисяч доларів).

На жаль, знайти системи середнього цінового діапазону не вдалося, проте багато з них представляють собою набір пакетів і розкид у ціні однієї і тієї ж системи визначається комплектом поставки, тобто обсягом виконуваних функцій. Дешеві системи відрізняються від дорогих тим, наскільки детально вдається в них описати характеристики окремих частин модельованої системи. Вони дозволяють отримати лише "приблизні" результати, не дають статистичних характеристик і не надають можливості проведення докладного аналізу системи. Системи класу high-end дозволяють збирати вичерпну статистику по кожному з компонентів мережі при передачі даних по каналах зв'язку і проводити статистичну оцінку отриманих результатів. За функціональністю системи моделювання, що використовуються при дослідженні обчислювальних систем, можуть бути розбиті на два основні класи:

- Системи, що моделюють окремі елементи (компоненти) системи.
- Системи, що моделюють обчислювальну систему цілком.

Класифікувати програмні продукти для моделювання можна за наступними критеріями:

- вартість:
- платне програмне забезпечення;
- вільне програмне забезпечення;
- можливості програм:
- високо-функціональні;
- середньо-функціональні;
- низько-функціональні;
- типи мереж, що моделюються:
- глобальні;
- локальні;
- локальні і глобальні
- принцип роботи:
- програми, що моделюють;
- програми, що оцінюють.

Найбільш розповсюджені програмні продукти для моделювання комп'ютерних мереж наведені у табл. 1.1.

Такі програмні продукти, як COMNET III, NetMaker XA, MIND, AutoNet/Designer, AutoNet/MeshNET, BONES та Opnet є високо-функціональними засобами моделювання комп'ютерних мереж, що мають можливість моделювання як локальних, так і глобальних мереж. Але вони мають дуже велику вартість.

Ряд програм, таких як Prophecy, StressMagic, Performance-1, Performance-3, Netcracker мають меншу вартість, тобто відносяться до дешевих засобів моделювання, але вони мають середню функціональність, хоча також можуть моделювати і локальні, і глобальні мережі.

Крім комерційних програм існують засоби моделювання, які поширюються під ліцензією GNU GPL.

Більшість з них, такі як D-ITG, Friendly Pinger, Netperf, RUDE, MGEN, Iperf та UDP Packet Generator відносяться до програм, що оцінюють комп'ютерні мережі.

Вони мають низьку функціональність та не підтримують глобальні мережі.

Таблиця 1.1

Програмне забезпечення для моделювання комп'ютерних мереж

Компанія та продукт	Вартість (долларів)	Тип мережі	Необхідні ресурси	Примітки
American HYTech, Prophesy	1495	ЛС	8Мб ОП, 6 Мбдиск, DOS, Windows, OS/2	Оцінювання продуктивності при роботі з текстовими і графічними даними по окремих
CACI Product, COMNET III	34500-39500	ЛС, ГС	32 МбОП, 100 Мбдиск, Windows, Windows NT, OS/2, Unix	Моделює мережі X.25, ATM, зв'язки LAN-WAN, SNA, OSPF, RIP. Доступ CSMA / CD і токенний маршрутизаторів 3COM, Cisco, Wellfleet, ...
Make System, NetMaker XA	6995- 14995	ЛС, ГС	128 МбОП, 2000 Мбдиск, AIX, Sun OS, Sun Solaris	Перевірка даних про топологію мережі; імпорт інформації про трафік, що отримується в реальному часі
NetMagic System, StressMagic	2995	ЛС	2 МбОП, 8 Мбдиск, Windows	Підтримка стандартних тестів виміру продуктивності; імітація пікового навантаження на файл-сервер
Network Analysis Center, MIND	9400-70000	ГС	8 МбОП, 65 Мбдиск, DOS, Windows	Засіб проєктування, оптимізації мережі, містить дані про вартість типових конфігурацій з можливістю точного оцінювання продуктивності
Network Design and Analysis Group, AutoNet/ Designer	25000	ГС	8 МбОП, 40 Мбдиск, Windows, OS/2	Визначення оптимального розташування концентратора в ГС, можливість оцінки економії коштів за рахунок зниження тарифної плати, зміни постачальника послуг і оновлення обладнання; порівняння варіантів зв'язку через найближчу і оптимальну точку доступу, а також через міст і місцеву телефонну мережу
Network Design and Analysis Group,	30000	ГС	8 МбОП, 40 Мбдиск, Windows, OS/2	Моделювання смуги пропускання і оптимізація витрат на організацію ГС шляхом імітації пошкоджених

Компанія та продукт	Вартість (долларів)	Тип мережі	Необхідні ресурси	Примітки
AutoNet/ MeshNET				ліній, підтримка тарифної сітки компаній AT & T, Sprint, WiTel, Bell
Network Design and Analysis Group, AutoNet/ Performance-1	4000	ГС	8 МбОП, 1 Мбдиск, Windows, OS/2	Моделювання продуктивності ієрархічних мереж шляхом аналізу чутливості до тривалості затримки, часу відповіді, а також вузьких місць в структурі мережі
Network Design and Analysis Group, AutoNet/ Performance-3	6000	ГС	8 МбОП, 3 Мбдиск, Windows, OS/2	Моделювання продуктивності багатопрокольних об'єднань локальних і глобальних мереж; оцінювання затримок в чергах, прогнозування часу відповіді, а також вузьких місць в структурі мережі; облік реальних даних про трафік, що надходять від мережевих аналізаторів
System& Networks, BONES	20000-40000	ЛС, ГС	32 МбОП, 80 Мбдиск, Sun OS, Sun Solaris, HP-UX	Аналіз впливу додатків клієнт-сервер і нових технологій на роботу мережі
MIL3, Opnet	16000-40000		16 МбОП, 100 Мбдиск, DEC AXP, Sun OS, Sun Solaris, HP-UX	Має бібліотеку різних мережевих пристроїв, підтримує анімацію, генерує карту мережі, моделює смугу пропускання.
Friendly Pinger	Вільне програмне забезпечення	ЛС	16 МбОП, 20 Мбдиск, Windows 98/ME/2000/XP/Vista/7	Безкоштовний додаток для адміністрування, моніторингу та інвентаризації комп'ютерних мереж.
GNS3	Вільне програмне забезпечення	ЛС, ГС	Windows XP/Vista/7, процесор з частотою 1.5 ГГц, 1 Гбайт ОП	Програмний симулятор маршрутизаторів Cisco, працює на більшості Linux-систем, Mac OS X і Windows, при цьому дозволяє емулювати апаратну частину маршрутизаторів, безпосередньо завантажуючи і взаємодіючи з реальними образами Cisco IOS
NS3	Вільне програмне забезпечення	ЛС, ГС	Windows XP/Vista/7	Потужний засіб імітаційного моделювання телекомунікаційних мереж, що дозволяє моделювати мережі будь-якої топології і складності
Netcracker	—	ЛС, ГС	128 МбОП, 160 Мбдиск, 2 MB відео пам'яті (рекомендується 4 MB або більше)	Система являє собою CASE-засоби автоматизованого проєктування, моделювання і аналізу комп'ютерних мереж. Дозволяє провести експерименти, результати яких можуть бути використані для

Компанія та продукт	Вартість (долларів)	Тип мережі	Необхідні ресурси	Примітки
				обґрунтування вибору типу мережі, серед передачі, мережевих компонент обладнання та програмно-математичного забезпечення.
D-ITG	Вільне програмне забезпечення	ЛС	8 МбОП, 3 Мбдиск, Windows, Debian	Засіб генерування IPv4 та IPv6 трафіка на мережному, транспортному та прикладному рівнях
RUDE	Вільне програмне забезпечення	ЛС	4 МбОП, 2 Мбдиск, Windows, Linux	Невелика і гнучка програма, яка генерує трафік у мережі
MGEN	Вільне програмне забезпечення	ЛС	4 МбОП, 2 Мбдиск, Windows, Linux	Набір інструментів для генерації трафіку в реальному часі
Iperf	Вільне програмне забезпечення	ЛС	6 МбОП, 4 Мбдиск, Windows, Linux	Консольна клієнт-серверна програма - генератор TCP і UDP трафіку для тестування пропускної здатності мережі
UDP Packet Generator	Вільне програмне забезпечення	ЛС	4 МбОП, 2 Мбдиск, Windows, Linux	Пакет інструментів, який дозволяє створювати і відправляти один або декілька UDP пакетів
Netperf	Вільне програмне забезпечення	ЛС	3 МбОП, 2 Мбдиск, Windows	Безкоштовна утиліта, яка була розроблена компанією Hewlett-Packard. Вона дозволяє отримати миттєву інформацію про пропускну здатність мережі.

Так, Friendly Pinger дозволяє адмініструвати, проводити моніторинг та інвентаризацію комп'ютерної мережі. За допомогою Netperf можна отримати миттєву інформацію про пропускну здатність мережі. D-ITG дозволяє генерувати IPv4 та IPv6 трафік на мережному, транспортному та прикладному рівнях. Програми RUDE та MGEN являють собою набір інструментів для генерації трафіку в реальному часі. Iperf – консольна клієнт-серверна програма-генератор TCP і UDP трафіку для тестування пропускної здатності мережі. UDP Packet Generator дозволяє створювати і відправляти один або декілька UDP пакетів та аналізувати трафік.

Але серед вільного програмного забезпечення є такі засоби моделювання комп'ютерних мереж, які за функціональністю не поступаються своїм високо-

функціональним комерційним аналогам, таким як COMNET III, Opnet та ін. Це GNS3 та NS3.

GNS3 – програмний симулятор маршрутизаторів Cisco, працює на більшості Linux-систем, Mac OS X і Windows, при цьому дозволяє емулювати операційні системи реального мережевого устаткування, безпосередньо завантажуючи і взаємодіючи з реальними образами Cisco IOS, підтримує локальні та глобальні мережі.

NS3 – потужний засіб імітаційного моделювання телекомунікаційних мереж, що дозволяє моделювати мережі будь-якої топології і складності.

Він є дуже гнучким і в той же час потужним засобом моделювання за рахунок використання C++ в якості вбудованої мови опису моделей. Так само, крім C++, може використовуватися Python. Обидві мови в симуляторі рівноправні і застосовуються для опису моделей телекомунікаційних систем.

Завдяки дуже великому і гнучкому API, а також повноті документації програмних інтерфейсів, розробник моделі практично нічим не обмежується. Йому надається можливість побудови як власних моделей будь-якої складності, так і, завдяки використовуваній ліцензії GNU GPLv2, зміни і доповнення вже існуючих моделей, що входять в комплект програмного забезпечення.

В NS3 розроблені моделі бездротових типів мереж, що дозволяють проводити моделювання навіть з рухомими об'єктами в тривимірному просторі. Розроблено моделі для побудови дротових топологій різної складності, а також змішаних. Присутня реалізація різних типів Mesh-мереж на основі стека протоколів 802.11s. Розроблено Framework під назвою FlowMonitor, що надає гнучкі методи збору показань з активних мережних пристроїв і каналів зв'язку, що моделюються.

Висновки до розділу

Як видно з наведеного аналізу, основною тенденцією розвитку засобів моделювання комп'ютерних мереж є застосування модульного принципу. Тобто при моделюванні мережі здійснюється емуляція операційних систем реального

мережевого устаткування. Це дозволяє створити таку модель, яка є максимально адекватною мережі, що моделюється, та отримати найбільш достовірні результати.

РОЗДІЛ 2. РОЗРОБКА ПРОГРАМНИХ РЕАЛІЗАЦІЙ АЛГОРИТМІВ

2.1. Технологія Gigabit Ethernet

2.1.1. Загальні положення

Основні особливості сучасного обладнання Gigabit Ethernet:

- підтримка оптичних інтерфейсів із великою дальністю дії;
- повна інтеграція та сумісність із наявними мережами Ethernet;
- масштабованість, що забезпечує плавний розвиток мережі та можливість створення магістральних з'єднань (транків);
- висока швидкість передачі даних, включаючи підтримку стандарту 10 Gigabit Ethernet;
- підвищена надійність мережі завдяки резервним і транковим з'єднанням;
- простота і ефективність у передачі Ethernet-трафіку, без зайвих ускладнень;
- підтримка сучасних технологій забезпечення якості обслуговування (QoS), таких як DiffServ і MPLS.

У березні 1996 року комітет IEEE 802.3 затвердив проєкт стандартизації Gigabit Ethernet 802.3z. У травні того ж року 11 компаній (серед яких 3Com Corp., Bay Networks Inc., Cisco Systems Inc., Compaq Computer Corp., Granite Systems Inc., Intel Corporation, LSI Logic, Packet Engines Inc., Sun Microsystems Computer Company, UB Networks і VLSI Technology) створили Gigabit Ethernet Alliance.

Цей альянс, об'єднуючи провідних виробників мережевого обладнання, ставив за мету розробку єдиного стандарту і випуск взаємосумісних продуктів Gigabit Ethernet. Основні завдання альянсу включали:

- підтримку розвитку технологій Ethernet і Fast Ethernet для задоволення потреб у вищій швидкості передачі даних;
- розробку технічних пропозицій для включення до стандарту;
- створення процедур і методів тестування продуктів від різних виробників для забезпечення сумісності.

На початок 1998 року до складу Gigabit Ethernet Alliance входило понад 100 компаній. Альянс забезпечував зворотний зв'язок між технічним комітетом IEEE 802.3 і виробниками мережевого обладнання, що сприяло підвищенню ефективності роботи комітету та прискоренню затвердження специфікацій стандартів Gigabit Ethernet IEEE 802.3z і IEEE 802.3ab. Найбільші труднощі виникли на фізичному рівні, зокрема через адаптацію багатомодового волокна і кручених пар.

29 червня 1998 року, з піврічною затримкою від початково запланованого терміну, викликану доопрацюванням стандарту щодо використання багатомодового волокна (проблема, відома як аномалія DMD), було прийнято стандарт IEEE 802.3z. Цей стандарт, затверджений на основі п'ятого драфту (802.3z/D5), регламентує використання одномодового та багатомодового волокна, а також крученої пари UTP категорії 5 на короткі відстані (до 25 м).

Для стандартизації передачі Gigabit Ethernet по неекранованій крученій парі на відстані до 100 м виникла необхідність розробки спеціального завадостійкого коду. Для цього було створено окремий підкомітет P802.3ab. 28 червня 1999 року був одноголосно затверджений відповідний стандарт (шостий драфт 802.3ab/D6).

2.1.2. Архітектура стандарту Gigabit Ethernet

На рисунку 2.1 представлено структуру рівнів Gigabit Ethernet. Подібно до стандарту Fast Ethernet, у Gigabit Ethernet відсутня універсальна схема кодування сигналу, яка б однаково ефективно працювала для всіх фізичних інтерфейсів. Для стандартів 1000Base-LX/SX/CX застосовується кодування 8B/10B, тоді як для стандарту 1000Base-T використовується спеціальний розширений лінійний код TX/T2.

Кодування здійснюється на підрівні кодування PCS, який знаходиться нижче за середонезалежний інтерфейс GMII.

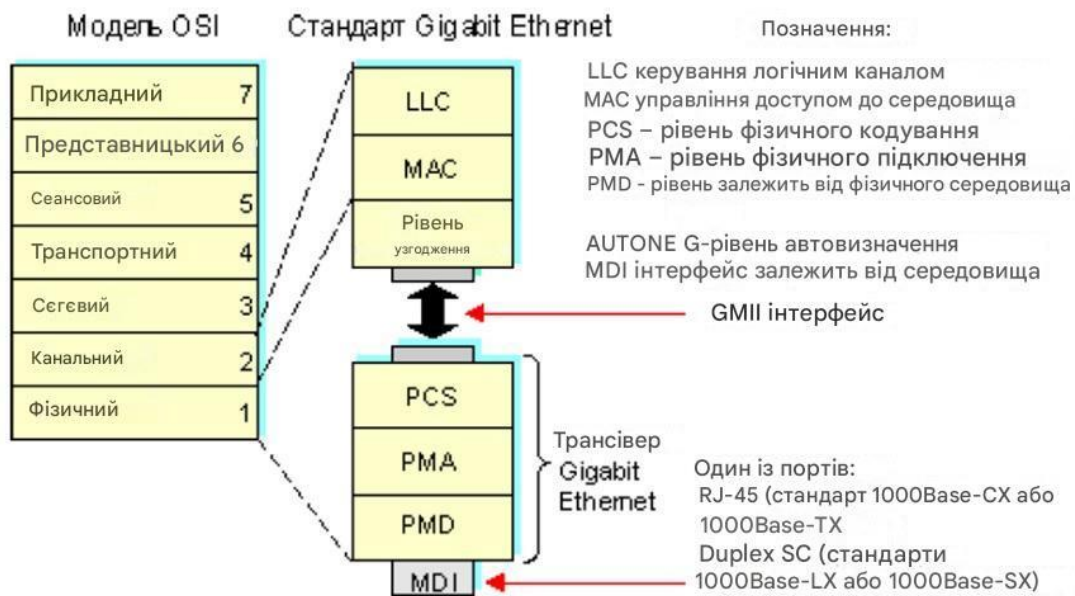


Рис. 2.1. Структура рівнів стандарту Gigabit Ethernet, GII інтерфейс і трансіввер Gigabit Ethernet

Інтерфейс GMI (Gigabit Media Independent Interface) забезпечує зв'язок між рівнем MAC і фізичним рівнем мережі. Він є розширенням інтерфейсу MII та підтримує швидкості передачі даних 10, 100 і 1000 Мбіт/с. GMI має окремі 8-бітні приймач і передавач, а також підтримує як напівдуплексний, так і повнодуплексний режими роботи.

Інтерфейс включає один сигнал синхронізації (clock signal) та два сигнали стану лінії: перший сигнал вказує наявність несучої хвилі (активний стан ON), а другий сигнал підтверджує відсутність колізій (активний стан ON). Крім цього, GMI забезпечує передачу кількох додаткових сигнальних каналів і живлення.

Трансівверний модуль, який відповідає за фізичний рівень та реалізує один із фізичних середозалежних інтерфейсів, може підключатися до комутатора Gigabit Ethernet через GMI інтерфейс.

Підрівень фізичного кодування PCS. У разі використання інтерфейсів 1000Base-X підрівень PCS застосовує блочне надлишкове кодування 8B10B, запозичене зі стандарту ANSI X3T11 Fibre Channel. Це схоже на стандарт FDDI, але використовує більш складну кодову таблицю, де кожен 8 біт вхідних даних перетворюються на 10 бітні символи (code groups), які передаються на

віддалений вузол. Крім того, у вихідному потоці присутні спеціальні контрольні 10-бітні символи, наприклад, для розширення носія, що дозволяє доповнити кадр Gigabit Ethernet до мінімального розміру 512 байт. Для інтерфейсу 1000Base-T підрівень PCS застосовує спеціальне завадостійке кодування для забезпечення передачі по кручений парі UTP Cat.5 на відстань до 100 метрів, використовуючи лінійний код TX/T2, розроблений компанією Level One Communications.

Цей підрівень також генерує два сигнали стану лінії: сигнал наявності несучої та сигнал відсутності колізій.

Підрівні PMA та PMD. Фізичний рівень Gigabit Ethernet включає кілька типів інтерфейсів, таких як традиційна крута пара категорії 5, багатомодові та одномодові волокна. Підрівень PMA перетворює паралельний потік символів від PCS на послідовний потік і здійснює зворотне перетворення (розпаралелювання) для послідовного потоку від PMD. Підрівень PMD визначає оптичні та електричні характеристики фізичних сигналів для різних середовищ. Всього існують 4 різні типи фізичних інтерфейсів, що описані у специфікаціях стандарту 802.3z (1000Base-X) і 802.3ab (1000Base-T) (рис.2.2).

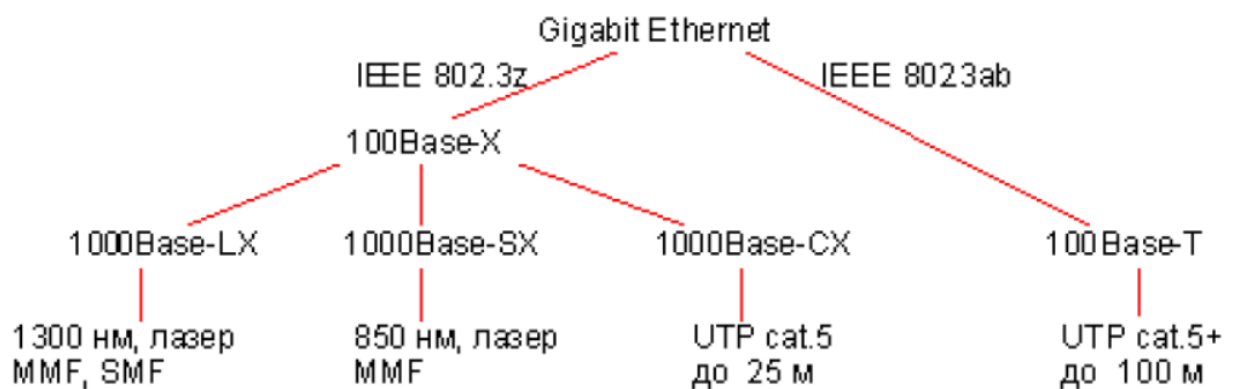


Рис. 2.2. Фізичні інтерфейси стандарту Gigabit Ethernet

2.1.3. Інтерфейс 1000Base -X

Інтерфейс 1000Base-X базується на стандарті фізичного рівня Fibre Channel, який є технологією для з'єднання робочих станцій, суперкомп'ютерів, пристроїв зберігання даних і периферійних вузлів. Fibre Channel має чотири рівні архітектури, з яких два нижні рівні — FC-0 (інтерфейси і середовище) та FC-1

(кодування/декодування) — були перенесені в стандарт Gigabit Ethernet. Оскільки Fibre Channel є вже затвердженою технологією, це значно пришвидшило розробку стандарту Gigabit Ethernet.

Блоковий код 8B/10B схожий на код 4B/5B, який застосовувався в стандарті FDDI. Однак код 4B/5B був відхилений у Fibre Channel, оскільки він не забезпечував необхідного балансу по постійному струму. Відсутність такого балансу могла призвести до перегріву лазерних діодів у передавачах, оскільки передавач міг передавати більше бітів "1" (випромінювання активно) ніж "0" (випромінювання відсутнє), що в свою чергу могло викликати додаткові помилки при високих швидкостях передачі.

Інтерфейс 1000Base-X включає три фізичних інтерфейси, основні характеристики яких наведені нижче [3]:

- Інтерфейс 1000Base-SX використовує лазери з довжиною хвилі випромінювання в діапазоні від 770 до 860 нм, потужність випромінювання передавача від -10 до 0 дБм, з відношенням ON/OFF (сигнал/відсутність сигналу) не гірше за 9 дБ. Чутливість приймача складає -17 дБм, а його насичення - 0 дБм.
- Інтерфейс 1000Base-LX використовує лазери з довжиною хвилі випромінювання в діапазоні від 1270 до 1355 нм, потужність випромінювання передавача варіюється від -13,5 до -3 дБм, з відношенням ON/OFF (сигнал/відсутність сигналу) не гірше за 9 дБ. Чутливість приймача складає -19 дБм, а насичення -3 дБм.
- Інтерфейс 1000Base-CX використовує екрановану кручена пару (STP "twinaх") для коротких відстаней.

При кодуванні 8B/10B бітова швидкість в оптичній лінії становить 1250 біт/с, що означає, що смуга пропускання ділянки кабелю на допустимій довжині повинна бути не менше 625 МГц. Оскільки Gigabit Ethernet працює на високих швидкостях, важливо ретельно підходити до проектування довгих сегментів мережі. Перевага віддається одномодовому волокну, оскільки його

характеристики можуть бути значно кращими. Наприклад, компанія NBase випускає комутатори з портами Gigabit Ethernet, які забезпечують передачу на відстань до 40 км по одномодовому волокну без ретрансляції, використовуючи узкоспектральні DFB лазери, що працюють на довжині хвилі 1550 нм.

2.1.4. Інтерфейс 1000Base-T

1000Base-T — це стандарт інтерфейсу Gigabit Ethernet для передачі даних по неекранованій кручений парі категорії 5 і вище на відстані до 100 метрів. Для передачі використовуються всі чотири пари мідного кабелю, при цьому швидкість передачі на кожній парі складає 250 Мбіт/с. Стандарт передбачає забезпечення дуплексної передачі, тобто дані будуть передаватися одночасно в обох напрямках по кожній парі — подвійний дуплекс (dual duplex). Реалізація дуплексної передачі зі швидкістю 1 Гбіт/с по кручений парі UTP Cat.5 виявилася значно складнішою, ніж у стандарті 100Base-TX [5]. Вплив перехідних завад від трьох сусідніх пар на кожну з чотирьох пар кабелю вимагає розробки спеціальної перешкодостійкої схемотехніки передачі та інтелектуального модуля для розпізнавання і відновлення сигналу на прийомі. На початковому етапі для стандарту 1000Base-T розглядалися кілька методів кодування, зокрема: 5-рівнева імпульсно-амплітудна кодування PAM-5, квадратурна амплітудна модуляція QAM-25 та інші. Зрештою, стандартом було затверджено використання PAM-5 [5].

Схема чотирирівневого кодування PAM-4 обробляє вхідні біти парами. Існує всього чотири можливі комбінації бітів — 00, 01, 10, 11. Для кожної пари біт передавач може встановлювати відповідний рівень напруги переданого сигналу, що дозволяє знизити частоту модуляції чотирирівневого сигналу вдвічі, з 250 МГц до 125 МГц (див. рис. 2.3), а також зменшити частоту випромінювання. П'ятий рівень був доданий для створення надмірності коду, що дозволяє здійснювати корекцію помилок на прийомі [6]. Це дає додаткові 6 дБ у співвідношенні сигнал/шум.

Технічні характеристики стандартів 1000Base-X

Стандарт	Тип кабелю	Смуга пропускання (не гірше), МГц*Км	Макс. відстань, м *
1000BASE-LX (лазерний діод 1300 нм)	Одномодове волокно (9 мкм)	-	5000 **
	Багатомодове волокно (50 мкм)	500	550
	Багатомодове волокно (62,5 мкм)	320	400
	Багатомодове волокно (50 мкм)	400	500
	Багатомодове волокно (62,5 мкм)	200	275
	Багатомодове волокно (62,5 мкм)	160	220
1000BASE-SX (лазерний діод 850 нм)	Екранована кручена пара STP (150 OM)	-	25

* стандарти 1000BASE-SX і 1000BASE-LX припускають наявність дуплексного режиму
 ** Обладнання деяких виробників може забезпечувати більшу відстань, оптичні сегменти без проміжних ретрансляторів/підсилювачів можуть досягати 100 км.

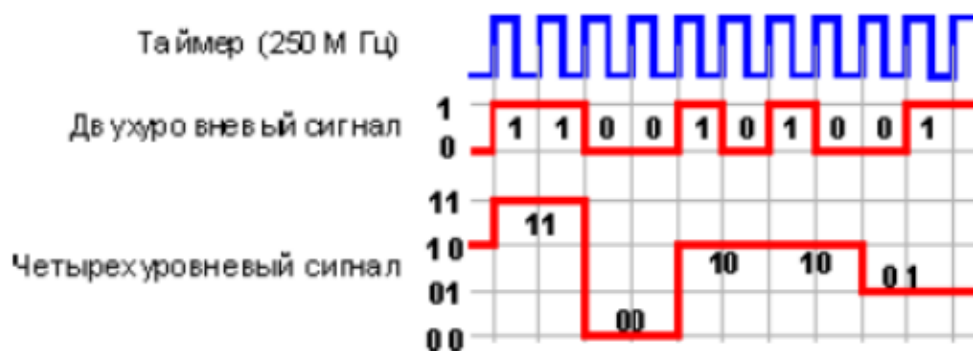


Рис. 2.3 - Схема 4-х рівневого кодування PAM-4

2.1.5. Рівень MAC

Рівень MAC стандарту Gigabit Ethernet використовує той самий протокол передачі CSMA/CD, що і попередні версії Ethernet та Fast Ethernet. Основні обмеження на максимальну довжину сегмента (або колізійного домену) визначаються саме цим протоколом.

У стандарті Ethernet IEEE 802.3 встановлено мінімальний розмір кадру 64 байти. Це значення мінімального розміру кадру визначає максимальну допустиму відстань між станціями (діаметр колізійного домену). Час передачі такого кадру — час каналу — складає 512 байт або 51,2 мкс. Максимальна довжина мережі Ethernet обмежена умовою вирішення колізій, що означає, що час, за який сигнал проходить до віддаленого вузла та повертається назад (RDT), не має перевищувати 512 байт (без урахування преамбули).

При переході від Ethernet до Fast Ethernet швидкість передачі збільшується, що призводить до скорочення часу трансляції кадру довжиною 64 байти — він становить 512 байт або 5,12 мкс (в Fast Ethernet 1 байт = 0,01 мкс). Для того, щоб можна було виявити всі колізії до завершення передачі кадру, необхідно виконати одну з наступних умов [4]:

1. Зберегти попередню максимальну довжину сегмента, але збільшити час каналу (а отже, і мінімальну довжину кадру);
2. Зберегти час каналу (залишивши колишній розмір кадру), але зменшити максимальну довжину сегмента.

У Fast Ethernet залишили той самий мінімальний розмір кадру, що й у Ethernet, що забезпечило сумісність, але призвело до значного зменшення діаметра колізійного домену. При переході до Fast Ethernet був обраний другий варіант, і діаметр сегмента скоротили.

У випадку з Gigabit Ethernet це виявилось неприйнятним, оскільки в такому разі стандарт, який успадкував елементи Fast Ethernet, такі як мінімальний розмір кадру, CSMA/CD і час виявлення колізій, міг би працювати лише в колізійних доменах діаметром не більше 20 метрів. Тому було запропоновано збільшити час на передачу мінімального кадру.

Для забезпечення сумісності з попередніми версіями Ethernet мінімальний розмір кадру залишили таким самим — 64 байти. До кадру додано додаткове поле *carrier extension* (розширення носія), яке доповнює кадр до 512 байт, але це поле не додається, якщо розмір кадру перевищує 512 байт. Таким чином,

мінімальний розмір кадру склав 512 байт, час на виявлення колізій збільшився, і діаметр сегмента зріс до 200 метрів (для стандарту 1000Base-T) [4].

Символи в полі *carrier extension* не містять корисної інформації, і контрольна сума для них не обчислюється. При прийомі кадру це поле відкидається на рівні MAC, тому вище розташовані рівні продовжують працювати з мінімальними кадрами довжиною 64 байти.

Хоча розширення носія допомогло зберегти сумісність з попередніми стандартами, воно призвело до неефективного використання смуги пропускання. У випадку коротких кадрів втрати можуть досягати 448 байт (512-64) на кадр.

Для вирішення цієї проблеми стандарт 1000BASE-T був оновлений, і введено поняття *Packet Bursting* (пакетна перевантаженість), що дозволяє ефективніше використовувати поле розширення. Це працює так: якщо в адаптера або комутатора є кілька малих кадрів для передачі, перший з них відправляється стандартним способом, з додаванням поля розширення до 512 байт. Наступні кадри передаються без поля розширення, з мінімальним інтервалом 96 біт між ними. Цей міжкадровий інтервал заповнюється символами розширення носія. Такий процес триває до того часу, поки сумарний розмір переданих кадрів не досягне межі 1518 байт.

Отже, середовище не залишається в "тиші" під час передачі малих кадрів, і колізія може виникнути лише на початковому етапі, під час передачі першого коректного малого кадру з полем розширення носія (розміром 512 байт) [4].

Цей механізм значно підвищує ефективність мережі, особливо за високих навантажень, зменшуючи ймовірність виникнення колізій. Проте існують деякі недоліки. Хоча розширення носія допомогло зберегти сумісність з попередніми стандартами, воно призвело до неефективного використання смуги пропускання. Втрати можуть досягати 448 байт (512-64) на кадр у випадку коротких кадрів. Тому стандарт 1000BASE-T був оновлений шляхом введення концепції *Packet Bursting* (пакетної перевантаженості). Цей підхід дозволяє значно ефективніше використовувати поле розширення. Процес виглядає наступним чином: якщо

адаптер або комутатор має кілька невеликих кадрів для передачі, перший з них відправляється стандартним способом, з додаванням поля розширення до 512 байт. Наступні кадри передаються без поля розширення, з мінімальним інтервалом 96 біт між ними. Важливо, що цей міжкадровий інтервал заповнюється символами розширення носія. Така передача триває до тих пір, поки загальний розмір переданих кадрів не досягне межі 1518 байт.

Отже, середовище не залишається в "тиші" протягом всієї передачі малих кадрів, тому колізія може виникнути лише на початковому етапі, при передачі першого правильного малого кадру з полем розширення носія (розміром 512 байт). Цей механізм значно підвищує ефективність мережі, зокрема при великих навантаженнях, завдяки зниженню ймовірності виникнення колізій. Однак цього було недостатньо. Спочатку Gigabit Ethernet підтримував тільки стандартні розміри кадрів Ethernet — від мінімального 64 байти (які доповнюються до 512) до максимального 1518 байт. З цих 18 байтів займає стандартний заголовок, а для даних залишається від 46 до 1500 байт відповідно. Однак навіть розмір пакета в 1500 байт є занадто малим для гігабітної мережі, особливо коли йдеться про сервери, що передають великі обсяги даних.

Поглянемо на це детальніше. Для передачі файлу обсягом 1 гігабайт по незавантаженій мережі Fast Ethernet сервер обробляє 8200 пакетів за секунду і витрачає мінімум 11 секунд на цю операцію. При цьому лише на обробку переривань у комп'ютера з потужністю 200 MIPS йде близько 10% часу, оскільки центральний процесор повинен обробити кожен пакет — перевірити контрольну суму, передати дані на зберігання тощо [6].

У гігабітних мережах ситуація ускладнюється ще більше — навантаження на процесор зростає приблизно в десять разів через скорочення тимчасових інтервалів між кадрами і відповідно запитами на переривання до процесора.

Характеристики передачі для різних швидкостей технології Ethernet

Швидкість	10 Мбіт/сек		100 Мбіт/сек		1000 Мбіт/сек	
Розмір кадру	64 байта	1518 байт	64 байта	1518 байт	64 байта	1518 байт
Кадри/сек	14.8K	812	148K	8,1K	1,48M	81K
Швидкість передачі даних, Мбіт/сек	5,5	9,8	55	98	550	980
Проміжок між кадрами, мкс	67	1200	6,7	120	0,7	12

Навіть за найкращих умов (при використанні кадрів максимального розміру) кадри розміщуються один від одного з часовим інтервалом, що не перевищує 12 мкс. Якщо використовуються кадри меншого розміру, цей інтервал ще зменшується [4]. В гігабітних мережах вузьким місцем, як не дивно, стала обробка кадрів процесором. Тому на початку розвитку Gigabit Ethernet реальні швидкості передачі значно відставали від теоретичних — процесори не встигали обробляти навантаження. Очевидним рішенням цієї проблеми є:

- збільшення тимчасового інтервалу між кадрами;
- перенесення частини навантаження на обробку кадрів із центрального процесора на мережевий адаптер.

На даний момент реалізовані обидва підходи. У 1999 році було запропоновано збільшити розмір пакета, і ці пакети отримали назву гіга-кадрів (Jumbo Frames), розмір яких може коливатися від 1518 до 9018 байт (зараз деякі виробники обладнання підтримують і більші розміри). Використання Jumbo Frames дозволило зменшити навантаження на центральний процесор до шести разів (пропорційно їхньому розміру), що значно підвищило продуктивність. Наприклад, пакет Jumbo Frame розміром 9018 байт містить 9000 байт даних, що відповідає шести стандартним максимальним кадрам Ethernet, з додаванням 18-байтового заголовка. Поліпшення продуктивності не відбувається через зменшення кількості службових заголовків (оскільки трафік від їх передачі

займає лише кілька відсотків загальної пропускну здатності), а завдяки зниженню часу, необхідного для обробки одного великого кадру. Це означає, що час на обробку не змінюється, але замість обробки кількох малих кадрів, кожен з яких потребує N тактів процесора та одного переривання, обробляється лише один великий кадр.

2.1.6. Специфікації фізичного середовища стандарту 802.3z

Стандарт 802.3z визначає наступні типи фізичних середовищ для передачі даних:

- одномодовий волоконно-оптичний кабель;
- багатомодовий волоконно-оптичний кабель 62,5/125;
- багатомодовий волоконно-оптичний кабель 50/125;
- подвійний коаксіальний кабель з хвильовим опором 75 Ом.

Для передачі по традиційному багатомодовому волоконно-оптичному кабелю стандарт передбачає використання випромінювачів, які працюють на двох довжинах хвиль: 1300 і 850 нм. Використання світлодіодів з довжиною хвилі 850 нм обумовлено їх низькою вартістю порівняно з діодами, що працюють на хвилі 1300 нм. Однак варто зазначити, що максимальна довжина кабелю скорочується, оскільки загасання багатомодового волокна на хвилі 850 нм більше ніж у два рази, порівняно з хвилею 1300 нм.

Стандарт 802.3z визначає специфікації для багатомодового оптоволокна 1000Base-SX і 1000Base-LX [7]. У разі 1000Base-SX використовується довжина хвилі 850 нм (S позначає коротку хвилю), а для 1000Base-LX — 1300 нм (L позначає довгу хвилю).

Для 1000Base-SX максимальна довжина сегмента кабелю 62,5/125 становить 220 м, а для кабелю 50/125 — 500 м.

Для 1000Base-LX джерелом випромінювання завжди є напівпровідниковий лазер з довжиною хвилі 1300 нм. Основна область застосування цього стандарту — одномодове оптоволокно, де максимальна довжина кабелю може досягати 5000 м.

Специфікація 1000Base-LX також може працювати на багатомодовому кабелі, але в такому випадку максимальна відстань обмежена 550 м. Це зумовлено особливостями поширення когерентного світла в широкому каналі багатомодового кабелю. Для підключення лазерного трансівера до багатомодового кабелю необхідно використовувати спеціальний адаптер.

Як середовище передачі даних застосовується високоякісний твінаксіальний кабель (Twinaх) з хвильовим опором 150 Ом (2x75 Ом). Дані передаються одночасно через пару провідників, кожен з яких має екрануючу оплітку. Це забезпечує напівдуплексний режим передачі. Для повнодуплексної передачі потрібні додаткові дві пари коаксіальних провідників. Максимальна довжина сегмента твінаксіального кабелю обмежена 25 метрами, що робить це рішення підходящим для пристроїв, розташованих в межах однієї кімнати.

2.1.7. 10 Gigabit Ethernet

Новий стандарт 10-Гігабітного Ethernet охоплює сім різних стандартів фізичного середовища для LAN, MAN і WAN. Цей стандарт наразі описується через виправлення IEEE 802.3а та має бути включений до наступної версії стандарту IEEE 802.3 [7].

- **10GBASE-CX4** - Технологія 10-Гігабітного Ethernet для коротких відстаней (до 15 метрів), що використовує мідний кабель CX4 і конектори InfiniBand.
- **10GBASE-SR** - Технологія 10-Гігабітного Ethernet для коротких відстаней (до 26 або 82 метрів, в залежності від типу кабелю), що використовує багатомодове оптоволокно. Ця технологія також підтримує відстані до 300 метрів за допомогою нового багатомодового оптоволокна з характеристиками 2000 МГц/км.
- **10GBASE-LX4** - Використовує ущільнення по довжині хвилі для підтримки відстаней від 240 до 300 метрів по багатомодовому оптоволокну. Також підтримує відстані до 10 кілометрів при використанні одномодового оптоволокна.

- **10GBASE-LR і 10GBASE-ER** - ці стандарти підтримують передачу на відстані до 10 і 40 кілометрів відповідно.
- **10GBASE-SW, 10GBASE-LW та 10GBASE-EW** - ці стандарти мають фізичний інтерфейс, сумісний за швидкістю та форматом даних з інтерфейсом OC-192 / STM-64 SONET/SDH. Вони аналогічні стандартам 10GBASE-SR, 10GBASE-LR і 10GBASE-ER, оскільки використовують ті ж типи кабелів і мають подібні відстані передачі.
- **10GBASE-T, IEEE 802.3an-2006** - прийнятий у червні 2006 року після чотирьох років розробки. Цей стандарт використовує екрановану виту пару і підтримує передачу на відстані до 100 метрів.

2.1.8. Сімейство 40GBASE-R

Технології 40 і 100 Gigabit Ethernet на сьогоднішній день є найшвидшими у сфері комп'ютерних мереж. Останні версії специфікацій цих технологій були затверджені до 2012 року та стали частиною стандарту IEEE 802.3-2012.

- **40 Gigabit Ethernet** - загальний термін, що позначає специфікації Ethernet для передачі даних на швидкості до 40 Гбіт/с.
- **100 Gigabit Ethernet** - загальний термін, що позначає специфікації Ethernet для передачі даних на швидкості до 100 Гбіт/с.

У технологіях 40 і 100 Gigabit Ethernet збереглися попередні формати кадру, а також мінімальний і максимальний розміри кадру. Як і в технології 10GE, ці технології на рівні MAC підтримують лише повнодуплексний режим роботи. Максимальна довжина сегмента становить 40 000 м при використанні одномодового волоконно-оптичного кабелю.

Одним із основних застосувань технології 40 Гбіт/с є організація ядра високошвидкісних мереж для центрів обробки даних, де необхідна велика пропускна здатність, а також створення магістральних каналів. Технологія 100 Гбіт/с може бути використана в якості ядра для мереж операторів зв'язку або мереж Metro Ethernet.

Сімейство 40GBASE-R включає п'ять специфікацій:

1. **40GBASE-KR4:** Призначений для об'єднувачих плат (Backplane) модульних комутаторів і маршрутизаторів. Використовується чотиріпототокова передача з кодуванням 64B/66B. Швидкість кожного потоку — 10,3125 Гбод. Максимальна довжина сегмента по мідному кабелю — 1 м. Підтримується автоузгодження.
2. **40GBASE-CR4:** Використовує твінаксіальний кабель з максимальною довжиною сегмента 7 м. Передача здійснюється чотирма потоками з кодуванням 64B/66B. Швидкість кожного потоку — 10,3125 Гбод. Підтримується автоузгодження.
3. **40GBASE-SR4:** Використовує чотири волокна багатомодового волоконно-оптичного кабелю типу OM3 або OM4 з діаметром 50/125 мкм. Довжина хвилі — 850 нм. Здійснюється чотиріпототокова передача з кодуванням 64B/66B. Швидкість кожного потоку — 10,3125 Гбод. Максимальна довжина сегмента: до 100 м з кабелем OM3 і до 150 м з кабелем OM4.
4. **40GBASE-FR:** Використовує одномодовий волоконно-оптичний кабель для передачі на довжині хвилі 1550 нм, прийом можливий на хвилях 1310 нм або 1550 нм. Здійснюється однопотокова передача з кодуванням 64B/66B і швидкістю 41,25 Гбод. Максимальна довжина сегмента — 2 000 м.
5. **40GBASE-LR4:** Використовує одномодовий волоконно-оптичний кабель. Передача здійснюється чотирьохпототоковою системою з кодуванням 64B/66B. Швидкість кожного потоку — 10,3125 Гбод. Для передачі та прийому використовуються чотири довжини хвиль: 1271 нм, 1291 нм, 1311 нм та 1331 нм. Потоки комбінуються мультиплексором WDM на передавальній стороні та демультиплексуються на приймальній. Максимальна довжина сегмента — 10 000 м.

2.1.9. 100-Gigabit Ethernet (100-GE)

Інтенсивний розвиток інформаційних технологій та зростаючі вимоги до пропускної здатності мереж стимулювали дослідження в галузі високошвидкісного доступу до даних. Одним з найважливіших етапів у цьому напрямку стало прийняття рішення про розробку стандарту 100 Gigabit Ethernet (100GbE).

Ідея створення стандарту 100GbE виникла в результаті роботи дослідницької групи Higher Speed Study Group (HSSG) під егідою IEEE 802.3. У квітні 2007 року на зустрічі комітету в Оттаві було прийнято рішення про розробку технічних специфікацій для нових оптичних та мідних каналів передачі даних зі швидкістю 100 Гбіт/с. Для реалізації цього завдання була створена робоча група 802.3ba.

Потреба в таких високих швидкостях передачі даних обумовлена кількома факторами:

- Зростання обсягів даних. Постійне збільшення обсягів переданих даних, особливо мультимедійного контенту (відео, аудіо), вимагає відповідного збільшення пропускної здатності мереж.
- Розвиток хмарних сервісів. Популярність хмарних технологій та віртуальних серверів призвела до зростання обсягу трафіку між дата-центрами.
- Вимоги до якості сервісу. Сучасні користувачі очікують високої якості сервісу, що передбачає мінімальну затримку передачі даних і високу пропускну здатність.

При розробці стандарту 100GbE були враховані такі фактори:

- новий стандарт повинен забезпечувати зворотну сумісність з існуючими мережевими інфраструктурами.
- вартість реалізації технології 100GbE повинна бути економічно обґрунтованою.

- стандарт повинен базуватися на сучасних технологіях і відповідати вимогам до надійності та масштабованості.

Сімейство стандартів 100GBASE-R, розроблене для забезпечення передачі даних зі швидкістю 100 Гбіт/с, включає в себе кілька специфікацій, кожна з яких орієнтована на певні умови застосування та типи мережевих середовищ.

- 100GBASE-CR10: цей стандарт передбачає використання мідного твіаксіального кабелю для передачі даних на відстань до 7 метрів. Для досягнення необхідної пропускної здатності використовується десятипотокова передача з кодуванням 64B/66B та сигнальною швидкістю кожного потоку 10,3125 Гбод. Стандарт підтримує функцію автоузгодження.
- 100GBASE-SR10: даний стандарт використовує багатомодовий оптичний кабель 50/125 мкм (OM3 або OM4) з довжиною хвилі 850 нм. Загальна кількість волокон в кабелі становить десять. За принципом передачі даних він аналогічний 100GBASE-CR10, але забезпечує більшу дальність передачі – до 100 метрів для кабелю OM3 і до 150 метрів для кабелю OM4.
- 100GBASE-LR4 та 100GBASE-ER4: ці стандарти призначені для використання в мережах з великою протяжністю і використовують одномодовий оптичний кабель. Для досягнення необхідної пропускної здатності застосовується чотирьохпотокова передача з кодуванням 64B/66B та сигнальною швидкістю кожного потоку 25,7812 Гбод. Для передачі потоків використовується технологія мультиплексування з поділенням за довжиною хвилі (WDM). Стандарт 100GBASE-LR4 забезпечує дальність передачі до 10 000 метрів, а 100GBASE-ER4 – до 40 000 метрів.

Стандарти сімейства 100GBASE-R знаходять широке застосування в сучасних мережах, де необхідна висока пропускна здатність для передачі великих обсягів даних. Зокрема, вони використовуються в таких областях, як:

Центри обробки даних – для створення високошвидкісних мереж між серверами та системами зберігання даних.

Телекомунікаційні мережі – для забезпечення передачі великих обсягів відеоконтенту, таких як потокове відео та відеоконференції.

Корпоративні мережі – для підключення до мережі високопродуктивних серверів та систем зберігання даних.

Поява та широке розповсюдження стандартів Gigabit Ethernet (1 Гбіт/с) наприкінці 20 століття значно вплинули на розвиток комп'ютерних мереж. Стандарт IEEE 802.3z, затверджений у 1998 році, визначив основні характеристики та вимоги до мережевого обладнання, що підтримує цю швидкість передачі даних. Спочатку Gigabit Ethernet використовував виключно оптичні кабелі, що забезпечувало високу пропускну здатність та мінімальні перешкоди.

Проте, з метою розширення сфери застосування та зниження вартості побудови мереж, було розроблено доповнення до стандарту 802.3z – стандарт 802.3ab. Це дозволило використовувати неекрановану кручену пару категорії 5 (UTP5e) як середовище передачі для Gigabit Ethernet. Таке рішення зробило технологію більш доступною для широкого кола користувачів та стимулювало її активне впровадження в корпоративних та домашніх мережах.

Gigabit Ethernet став фундаментом для подальшого розвитку технологій передачі даних. Його надійність, масштабованість та здатність задовольнити вимоги різноманітних застосувань сприяли появі нових стандартів, таких як 10 Gigabit Ethernet та 100 Gigabit Ethernet. Ці стандарти забезпечують ще більшу пропускну здатність та дозволяють ефективно передавати великі обсяги даних, що є критично важливим для сучасних інформаційних систем.

Стандарт Gigabit Ethernet (1 Гбіт/с), затверджений як IEEE 802.3z, ознаменував значний прорив у розвитку локальних мереж. Однією з ключових переваг цього стандарту стала його здатність забезпечити високу пропускну здатність при збереженні сумісності з попередніми версіями Ethernet. Завдяки

збереженню існуючої кабельної інфраструктури та формату кадрів, міграція на Gigabit Ethernet стала відносно простою та економічно вигідною.

Теоретична пропускна здатність Gigabit Ethernet у 1000 Мбіт/с (приблизно 120 Мбайт/с) робить його вкрай конкурентоспроможним порівняно з іншими технологіями передачі даних того часу. Варто зазначити, що така швидкість була співмірною з пропускною здатністю шин PCI, що широко використовувалися в комп'ютерах. Це дозволило створити гігабітні мережеві адаптери, які ефективно працювали з різними типами систем.

Незважаючи на значне збільшення швидкості, Gigabit Ethernet зберіг основні принципи та технології, успадковані від попередніх версій Ethernet, такі як метод доступу до середовища CSMA/CD та можливість роботи в режимі повного дуплексу. Така консервативність у підході дозволила забезпечити плавний перехід на новий стандарт та мінімізувати ризики, пов'язані з впровадженням нових технологій.

Порівняно з альтернативними технологіями, такими як ATM та Fibre Channel, Gigabit Ethernet мав ряд суттєвих переваг: збереження існуючої інфраструктури та форматів кадрів дозволило легко інтегрувати Gigabit Ethernet в існуючі мережі; Gigabit Ethernet легко масштабується, що дозволяє будувати мережі різного розміру та складності; відносно невисока вартість обладнання та простота впровадження зробили Gigabit Ethernet доступним для широкого кола користувачів.

Завдяки своїм перевагам Gigabit Ethernet швидко завоював популярність і став домінуючою технологією в області локальних мереж. Сьогодні Gigabit Ethernet широко використовується в корпоративних мережах, мережах малого та середнього бізнесу, а також в домашніх мережах.

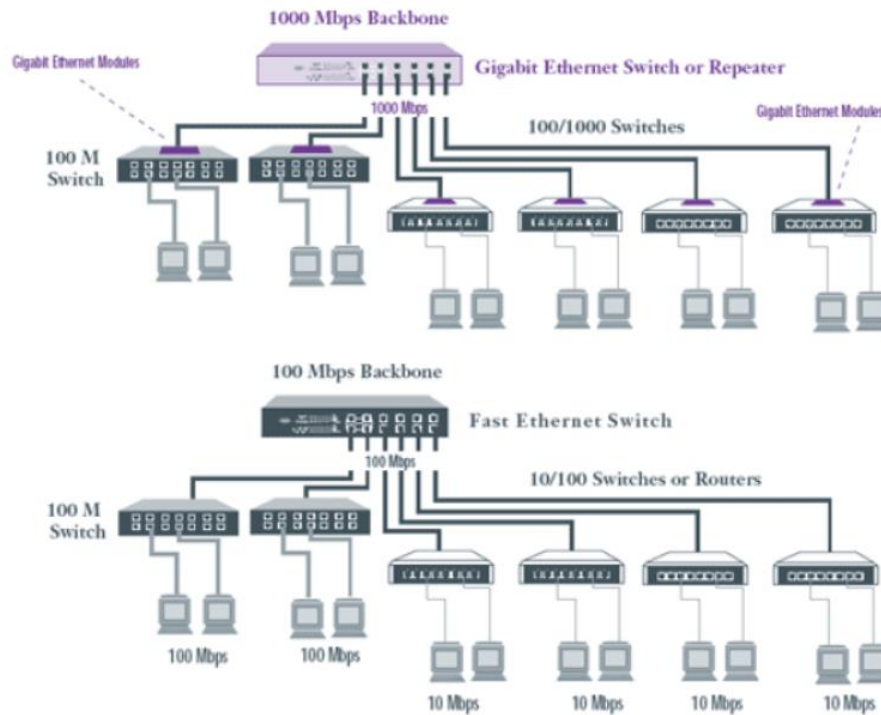


Рис. 2.4. Структура побудови мережі Ethernet із плавним переходом на більше високі швидкості передачі даних

2.1.10. Використання технології Ethernet для побудови мультисервісних мереж

Традиційно технологія Ethernet розглядалася як ефективний засіб для передачі даних в локальних обчислювальних мережах. Її відносна простота, висока пропускна здатність та доступність робили Ethernet популярним вибором для побудови корпоративних та домашніх мереж. Однак, з розвитком інформаційних технологій та зростанням вимог до мережевих сервісів, сфера застосування Ethernet значно розширилася.

Збільшення швидкості передачі даних від 10 Мбіт/с до 10 Гбіт/с при збереженні відносно невисокої вартості обладнання зробило Ethernet привабливим не тільки для локальних мереж, але й для більш масштабних інфраструктур. Оператори зв'язку та провайдери послуг Інтернету почали проявляти все більший інтерес до цієї технології, бачачи в ній потенціал для побудови високопродуктивних мереж.

Однак, з появою концепції мультисервісних мереж, заснованих на протоколі IP, вимоги до мережевих технологій значно зросли. Інтеграція даних, телефонії та відеоконференцій висунула на перший план такі характеристики, як надійність, якість обслуговування (QoS) та здатність до передачі різноманітних типів трафіку. Традиційні мережі Ethernet, орієнтовані на передачу даних, не завжди могли забезпечити необхідний рівень якості для мультимедійних додатків.

Для вирішення цієї проблеми було розроблено ряд нових технологій та протоколів, які дозволили адаптувати Ethernet для передачі інтегрованого трафіку. Серед них можна виділити:

Quality of Service (QoS). Механізми QoS дозволяють забезпечити пріоритетну обробку різних типів трафіку, що дозволяє гарантувати якість передачі голосових та відеоданих.

Virtual LANs (VLAN). Технологія VLAN дозволяє логічно розділити мережу на кілька віртуальних підмереж, що підвищує безпеку та спрощує управління мережею.

Multiprotocol Label Switching (MPLS). Протокол MPLS забезпечує швидку комутацію пакетів на основі ярликів, що дозволяє оптимізувати маршрутизацію трафіку і забезпечити високу якість обслуговування.

Стандарт IEEE 802.1Q/p став значним кроком у розвитку технології Ethernet, запровадивши концепцію віртуальних локальних мереж (VLAN) та механізм пріоритезації трафіку. Завдяки VLAN, стало можливим логічно розділити фізичну мережу на кілька віртуальних, що дозволило підвищити безпеку та ефективність використання мережевих ресурсів. Механізм пріоритезації трафіку, визначений стандартом 802.1p, дозволив забезпечити різний рівень якості обслуговування для різних типів даних, таких як голос, відео та дані.

Однак, незважаючи на значні досягнення, технологія VLAN мала свої обмеження. Вона була орієнтована на вирішення задач сегментації локальних

мереж та не забезпечувала достатньої гнучкості для побудови масштабних мереж операторів зв'язку. Для передачі мультимедійного трафіку, такого як голос та відео, вимагалися більш складні механізми управління трафіком, які б дозволяли гарантувати якість обслуговування від кінця до кінця.

Концепція якості обслуговування (QoS) є фундаментальною для сучасних мережевих технологій, особливо в контексті зростаючих вимог до передачі мультимедійних даних. QoS визначає рівень гарантій, які мережа надає для доставки даних, забезпечуючи передбачувану якість сервісу для різних типів додатків.

Конкретні параметри QoS, що характеризують якість передачі даних, залежать від вимог конкретного застосування. Наприклад, для голосової передачі критичними є затримка та її варіація, оскільки навіть незначні затримки можуть призводити до спотворення мови і зниження якості зв'язку. Для передачі відеоданих важливими є як затримка, так і втрати пакетів, оскільки вони впливають на плавність відтворення і якість зображення.

Зазвичай, параметри QoS поділяють на три основні групи:

- Параметри пропускної здатності. Характеризують швидкість передачі даних і включають такі показники, як мінімальна, середня та максимальна швидкість. Ці параметри визначають, з якою швидкістю дані можуть бути передані через мережу.
- Параметри затримки. Визначають час, необхідний для доставки пакета від джерела до призначення. До важливих параметрів належать середня та максимальна затримка, а також варіація затримки (джиттер), яка характеризує нестабільність затримок.
- Параметри надійності. Характеризують ймовірність успішної доставки пакетів без помилок. До основних параметрів належать рівень втрат пакетів та рівень спотворень.

Вимірювання параметрів якості обслуговування (QoS) проводиться на певних часових інтервалах. Чим коротший цей інтервал, тим точніше можна

оцінити динамічні зміни в мережі та забезпечити більш адаптивне управління трафіком. Однак, зменшення інтервалу вимірювання призводить до зростання вимог до мережевої інфраструктури. Для забезпечення гарантій QoS на всьому шляху проходження пакетів необхідно, щоб всі елементи мережі – від комутаторів до маршрутизаторів – підтримували функції QoS і могли координувати свої дії. Наявність хоча б одного пристрою, який не підтримує QoS, може суттєво вплинути на загальну якість обслуговування.

Одним із прикладів обмежень, пов'язаних із забезпеченням QoS, є використання концентраторів (hub) в мережах Ethernet. Концентратори працюють на фізичному рівні і не підтримують такі технології, як VLAN. Це означає, що якщо в мережі присутні концентратори, то неможливо забезпечити ізоляцію трафіку різних VLAN і гарантувати пріоритетну обробку пакетів.

Між постачальником мережевих послуг та його клієнтом зазвичай укладається угода про рівень обслуговування (Service Level Agreement, SLA). SLA визначає взаємні зобов'язання сторін, включаючи:

Вартість послуг. Ціна, яку клієнт платить за мережеві послуги, залежить від обраного рівня QoS.

Параметри QoS. У SLA конкретизуються кількісні характеристики якості обслуговування, такі як максимальна затримка передачі даних, варіація затримки (джиттер), пропускна здатність, час відновлення мережі після збоїв тощо. Ці параметри визначають, які гарантії якості надає провайдер клієнту.

Методи вимірювання. У SLA вказуються методи, за допомогою яких будуть проводитися вимірювання параметрів QoS. Це дозволяє об'єктивно оцінювати відповідність наданих послуг обумовленим умовам.

Штрафні санкції. У випадку порушення провайдером умов SLA, передбачаються штрафні санкції, які компенсують клієнту збитки.

Додаткові умови. SLA може містити інші положення, які стосуються надання послуг, такі як порядок вирішення спорів, умови розірвання угоди тощо.

Для реалізації різних рівнів QoS в IP-мережах використовуються спеціальні механізми управління трафіком. Серед найбільш поширених можна виділити:

RSVP (Resource Reservation Protocol). Цей протокол дозволяє резервувати ресурси мережі для індивідуальних потоків даних, забезпечуючи гарантовану якість обслуговування. RSVP використовується для реалізації QoS на рівні окремих додатків.

DiffServ (Differentiated Services). Цей механізм дозволяє класифікувати трафік на кілька класів з різними рівнями пріоритету. Маршрутизатори, що підтримують DiffServ, обробляють пакети різних класів по-різному, надаючи пріоритет більш важливим потокам.

Технологія DiffServ в мережах Ethernet

Технологія DiffServ (Differentiated Services) надає механізм для забезпечення різного рівня якості обслуговування (QoS) різним типам трафіку в мережі. Основна ідея DiffServ полягає в поділі всього мережевого трафіку на кілька класів, кожному з яких відповідає певний рівень QoS. Цей підхід дозволяє пріоритезувати важливий трафік, наприклад, голосовий або відео, забезпечуючи його більш швидку доставку та меншу втрату пакетів.

Реалізація DiffServ передбачає створення так званих доменів DiffServ - логічних областей мережі, в межах яких діють однакові правила класифікації та обробки трафіку. На кордонах домену відбувається кондиціонування трафіку, тобто його класифікація за певними критеріями, такими як IP-адреса джерела та призначення, номер порту, тип протоколу тощо. В результаті класифікації кожному пакету присвоюється спеціальне значення DSCP (DiffServ Code Point), яке вказує на його клас і, відповідно, на рівень QoS, який повинен бути забезпечений для цього пакета. Цю функцію виконують пристрої, які називаються портами доступу в домен.

Після того, як трафік класифікується на кордонах домену DiffServ і пакетам присвоюються відповідні значення DSCP, подальша обробка пакетів

здійснюється на основі саме цього поля. Проміжні вузли мережі, такі як маршрутизатори, аналізують значення DSCP в заголовку кожного IP-пакета і, виходячи з цієї інформації, приймають рішення про подальшу маршрутизацію та пріоритет обслуговування.

Процес обробки пакетів всередині домену DiffServ є відносно простим і ефективним. Маршрутизатор, отримавши пакет, аналізує лише 6-бітове поле DSCP і направляє його в відповідну чергу. Цей процес відбувається зі швидкістю, близькою до швидкості комутації, що забезпечує низьку затримку обробки пакетів. Після того, як пакет потрапляє в чергу, для його подальшої передачі використовується алгоритм справедливого обслуговування (weighted fair queuing), який забезпечує справедливий розподіл пропускної здатності між різними чергами (рис. 2.5).

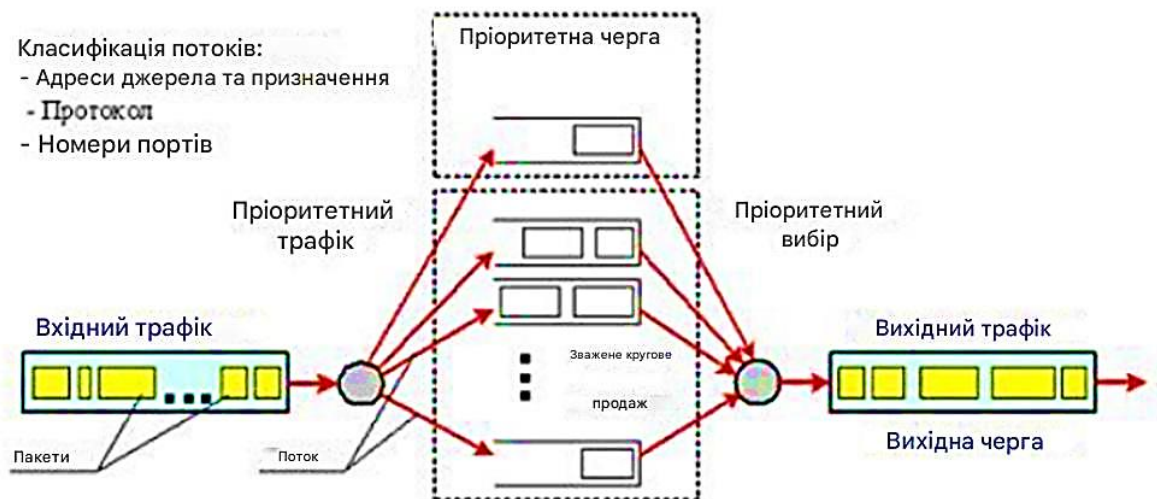


Рис. 2.5. Механізм зваженого справедливого обслуговування

Алгоритми обробки трафіку з використанням технології DiffServ можуть значно відрізнятися в залежності від конкретної моделі обладнання різних виробників. Розглянемо два приклади: маршрутизатори Cisco та комутатори OptiSwitch.

Компанія Cisco Systems у своїх маршрутизаторах використовує два молодших біти з поля IP Precedence для класифікації трафіку. За замовчуванням, цим бітам відповідають чотири класи трафіку з наступним розподілом пропускної здатності: клас 0 – 10%, класи 1, 2 і 3 – відповідно 20%, 30% і 40%.

Пакеты, які не відносяться до жодного з цих класів, автоматично потрапляють в клас 0 і отримують 1% від загальної пропускної здатності. Така гнучка система дозволяє адаптувати розподіл пропускної здатності під конкретні вимоги мережі.

Компанія Optical Access пропонує більш простий, але ефективний підхід до реалізації DiffServ у своїх комутаторах. Адміністратор може вибрати один з чотирьох режимів роботи черг:

- Зважене кругове обслуговування (WRR): кожна черга отримує певну вагу, яка визначає її частку в загальній пропускній здатності. Пакети обслуговуються по черзі, причому кількість пакетів, оброблених з кожної черги, пропорційна її вазі.
- Змішане обслуговування 1/3: черги поділяються на дві групи: з високим і низьким пріоритетом. Трафік з високим пріоритетом обслуговується в три рази частіше, ніж трафік з низьким пріоритетом.
- Змішане обслуговування 2/2: черги також поділяються на дві групи, але трафік з обох груп обслуговується з однаковою частотою.
- Обслуговування з прямим пріоритетом (SP): пакети з більш високим пріоритетом завжди обслуговуються першими, незалежно від стану інших черг.

Іншим важливим аспектом забезпечення QoS в технології DiffServ є формування трафіку. Алгоритм "маркерного відра" (token bucket) дозволяє згладжувати піки навантаження і запобігати перевантаженню мережі.

Суть алгоритму полягає в наступному:

- 1) Існує віртуальне відро, в яке періодично додаються маркери.
- 2) Кожен пакет, який потрібно передати, повинен "витратити" один маркер.
- 3) Якщо в відрі немає вільних маркерів, пакет відкидається або затримується.

Цей механізм дозволяє обмежити середню швидкість передачі даних і гарантувати, що піки навантаження не перевищать заданого значення (рис. 2.6).

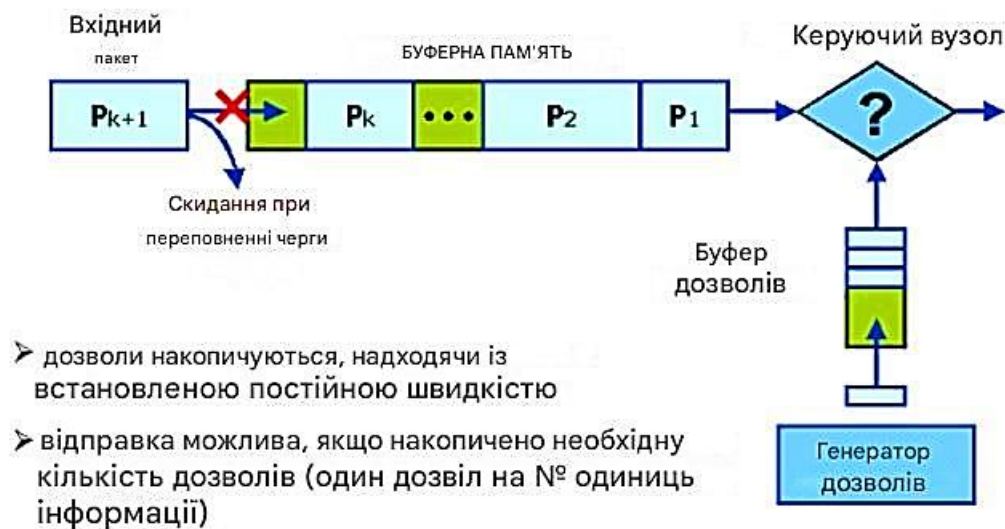


Рис. 2.6. Алгоритм "token bucket"

Алгоритм "token bucket" передбачає використання віртуального буфера (відра), в якому накопичуються маркери – дозволи на передачу даних. Швидкість накопичення маркерів визначає максимальну середню швидкість передачі. Для передачі пакета необхідно витратити кількість маркерів, що відповідає його розміру. При відсутності достатньої кількості маркерів пакет відкидається або затримується. Цей механізм дозволяє ефективно обмежити пікові значення швидкості передачі даних і забезпечити стабільну пропускну здатність мережі.

2.2. Огляд та обґрунтування вибору середовища розробки

Qt Creator - не так давно розроблене кросплатформене інтегроване середовище, призначене для розробки Qt-додатків на таких мовах програмування, як C, C++ і QML. В нього включений зручний відладчик, а також різні візуальні засоби розробки інтерфейсу, що використовують QtWidgets і QML. Творці даного програмного рішення ставили собі цілі створити засіб, що значно спрощує розробку додатків за допомогою фреймворка Qt для різних платформ.

Qt Creator це повністю інтегроване середовище розробки (IDE), яке надає вам інструменти проєктування і розробки складних додатків для безлічі настільних і мобільних платформ.



Проекти

Одним з найголовніших досягнень Qt Creator є те, що він дозволяє команді розробників працювати над проєктом на різних платформах з використанням загальних інструментів для розробки і налагодження.

Але навіщо вам потрібні проєкти? Щоб бути в змозі збирати і запускати додатки, Qt Creator потребує тієї ж інформації, яка буде потрібна компілятору. Ця інформація вказана в налаштуваннях збірки і запуску проєкту.

Створення проєкту дозволить вам:

- Групувати файли разом
- Додати власні кроки збірки
- Включити форми і файли ресурсів
- Вказувати настройки для додатків, що запускаються

Ви можете або створити проєкт з нуля, або імпортувати існуючий проєкт. Qt Creator генерує всі необхідні файли в залежності від типу створюваного проєкту. Наприклад, якщо ви оберете створення додатка з графічним інтерфейсом користувача (GUI), Qt Creator створить порожній .файл, який ви можете змінити в інтегрованому Qt Designer.

Qt Creator інтегрований з кросплатформеними системами автоматизації збирання: qmake і CMake. Також ви можете імпортувати існуючі проекти, які не використовують qmake або CMake, і вказати Qt Creator просто проігнорувати вашу систему збирання.

Редактори

Qt Creator поставляється з редактором коду і Qt Designer для проєктування і складання графічних інтерфейсів користувача (GUI) з віджетів Qt.

Редактор коду

Так як він є IDE, Qt Creator відрізняється від текстового редактора тим, що знає, як збирати і запускати додатки. Він розуміє мови C ++ і QML як код, а не як простий текст. Це дозволяє йому:

- Дати вам можливість писати добре форматований код
- Угадувати що ви хочете написати і доповнювати код
- Відображати повідомлення про помилки і попередження
- Дати вам можливість переміщатися між класами, функціями і символами
- Надавати вам контекстно-залежну довідку по класах, функціям і символам
- Осмислено перейменовувати символи так, що інші символи з таким же ім'ям, але які належать іншим областям дії, не будуть перейменовані
- Показувати вам місце в коді де функція була описана або викликана

Дизайнер інтерфейсу

Ви можете використовувати Qt Designer щоб мати у своєму розпорядженні і налаштовувати ваші віджети або діалоги і тестувати їх використовуючи різні стилі та роздільну здатність екрану. Створені за допомогою Qt Designer віджети і форми легко інтегруються в програмний код з використанням механізму сигналів і слотів Qt, які дозволять вам легко визначити поведінку графічних елементів. Всі властивості, встановлені в Qt Designer, можуть бути динамічно змінені в коді. Більш того, такі особливості як просування віджетів і власні модулі дозволять вам використовувати власні віджети з Qt Designer.

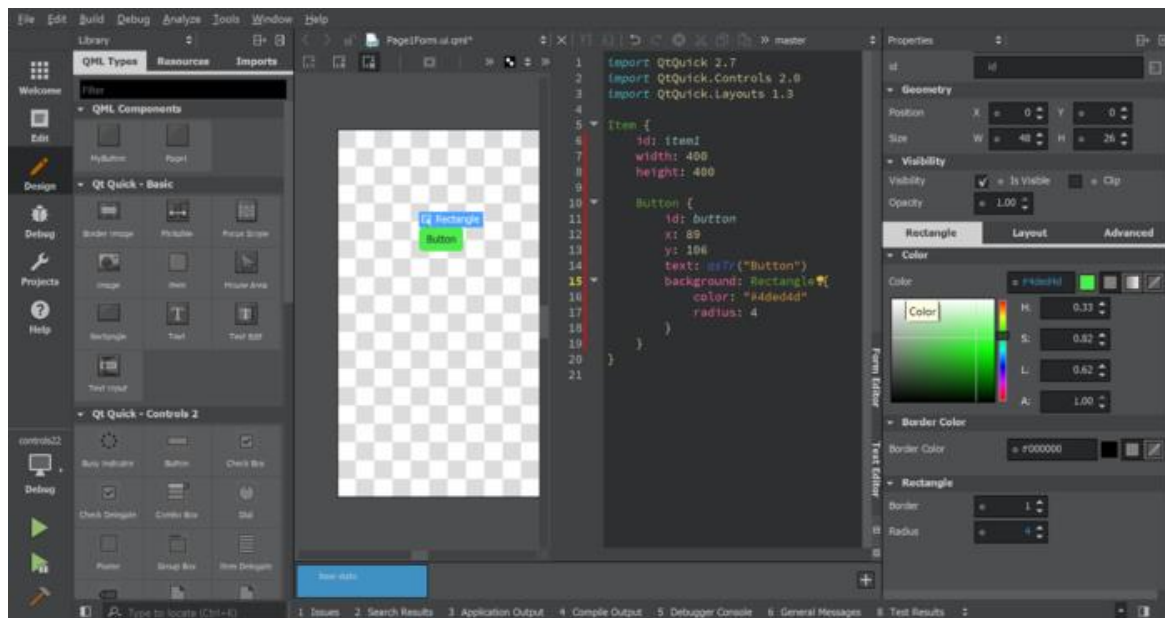


Рис. 2.7. Інтерфейс середовища розробки

Мови

Ви можете використовувати редактор для написання коду на Qt C++ або на мові декларативного програмування QML.

Ви можете використовувати QML для створення дуже гнучкого інтерфейсу користувача з великого набору елементів QML. QML допомагає розробникам і дизайнерам працювати разом над створенням гнучких призначених для користувача інтерфейсів, які поширяться на портативних пристроях, таких як мобільні телефони, медіаплеєри, неттопи і нетбуки.

QML це розширення JavaScript, яке надає механізм декларативною збірки дерева об'єктів з елементів QML. QML покращує інтеграцію між JavaScript і існуючою системою Qt, заснованої на QObject, додає підтримку автоматичного зв'язування властивостей і забезпечує мережеву прозорість на рівні мови.

Цілі

Qt Creator надає підтримку для збірки і запуску додатків на Qt для настільних комп'ютерів (Windows, Linux і Mac OS) і мобільних пристроїв (Symbian, Maemo і MeeGo). Налаштування збірки дозволять вам швидко перемикаватися між цілями збірки.

Коли ви збираєте додаток для мобільного пристрою, підключеного до вашого комп'ютера, Qt Creator генерує пакет установки, встановлює його на пристрої і запускає його.

Ви можете відправити пакет установки в Ovi Store. Пакети для пристроїв Symbian повинні бути підписані.

Інструменти

Qt Creator інтегрований з набором корисних інструментів, такі як системи управління версіями і емулятор Qt.

Системи управління версіями

Рекомендованим способом для розробки проєкту є використання системи управління версіями. Для доступу до ваших сховищ Qt Creator використовує консольні клієнти систем управління версіями. Підтримуються наступні системи управління версіями:

- Git
- Subversion
- Perforce
- CVS
- Mercurial

Доступні вам функції в Qt Creator залежать від системи управління версіями. Базові функції доступні для всіх підтримуваних систем. Вони включають порівняння файлів з останньою версією, що знаходиться в сховищі, і відображення різниці, перегляд історії версій і подробиць змін, анотацію файлів і прийняття та скасування змін.

Емулятор Qt

Для перевірки додатків на Qt призначених для мобільних пристроїв в схожому оточенні, ви можете використовувати емулятор Qt. Ви можете змінити інформацію пристрою про його налаштуваннях і оточенні.

Емулятор Qt встановлюється як частина Nokia Qt SDK. Після установки ви можете вибрати його в якості мети збірки в Qt Creator.

Відладчики

Qt Creator може допомогти вам з налагодженням ваших додатків. Він надає інтерфейси до GNU Symbolic Debugger (gdb) і Microsoft Console Debugger (CDB) для налагодження звичайних додатків на C ++ і внутрішні відладчики для JavaScript. Це включає можливість підключити мобільні пристрої до свого комп'ютера і налагоджувати запущені на них програми.

Qt Creator відображає сиру інформацію, надану відладчиками, явним і лаконічним чином з метою спростити процес налагодження наскільки можливо без обмеження можливостей відладчиків.

2.3. Елементи програми

Мережевий концентратор або Хаб (жарг. від англ. hub — центр діяльності) — мережевий пристрій, для об'єднання декількох пристроїв Ethernet в загальний сегмент мережі. Пристрої підключаються за допомогою виті пари, коаксіального кабелю або оптоволокна [5].

Маршрутизатор або роутер — мережевий пристрій, на підставі інформації про топологію мережі і певних правил, що приймає рішення про пересилку пакетів мережевого рівня (рівень 3 моделі OSI) між різними сегментами мережі.

Мережевий пристрій (англ. gateway) або програмний засіб для сполучення різнорідних мереж (локальною і глобальною). Мережевий пристрій, який передає протоколи одного типу фізичного середовища в протоколи іншого фізичного середовища (мережі). Наприклад, при з'єднанні комп'ютера з Інтернетом ви використовуєте шлюз.

Мережа функціонує таким чином, що пакет, надісланий з якогось вузлу на інший проходить між іншими вузлами (такими як hub, router, тощо) та досягає кінцевої цілі.

ПК матиме змогу генерувати та відсилати пакет даних на інший ПК. ПК може мати декілька мережевих карт. Також ПК може бути маршрутизатором, шлюзом або мостом.

Концентратор (hub) працює на фізичному рівні мережевої моделі OSI, повторює сигнал, що приходить на один порт, на всі активні порти. У разі надходження сигналу на два і більш за порт одночасно виникає колізія, і кадри даних які передані втрачаються. Таким чином, всі підключені до концентратора пристрої знаходяться в одному домені колізій. Концентратори завжди працюють в режимі напівдуплекса, всі підключені пристрої Ethernet розділяють між собою смугу доступу, що надається.

Багато моделей концентраторів мають простий захист від зайвої кількості колізій, що виникають унаслідок одного з підключених пристроїв. В цьому випадку вони можуть ізолювати порт від загального середовища передачі. З цієї причини, мережеві сегменти, засновані на витій парі набагато стабільніше в роботі сегментів на коаксіальному кабелі, оскільки в першому випадку кожен пристрій може бути ізольоване концентратором від загального середовища, а в другому випадку декілька пристроїв підключаються за допомогою одного сегменту кабелю, і, у разі великої кількості колізій, концентратор може ізолювати лише весь сегмент.

Останнім часом концентратори використовуються достатньо рідко, замість них набули поширення комутатори — пристрої, що працюють на канальному рівні моделі OSI і мережі, що підвищують продуктивність, шляхом логічного виділення кожного підключеного пристрою в окремий сегмент, домен колізії.

Зазвичай маршрутизатор використовує адреса одержувача, вказана в пакетах даних, і визначає по таблиці маршрутизації шлях, по якому слід передати дані. Якщо в таблиці маршрутизації для адреси немає описаного маршруту, пакет відкидається.

Існують і інші способи визначення маршруту пересилки пакетів, коли, наприклад, використовується адреса відправника, використовувані протоколи верхніх рівнів і інша інформація, що міститься в заголовках пакетів мережевого рівня. Нерідко маршрутизатори можуть здійснювати трансляцію адрес відправника і одержувача, фільтрацію транзитного потоку даних на основі

певних правил з метою обмеження доступу, шифрування/дешифровка передаваних даних і так далі.

2.4. Модулі програми

Модуль у програмуванні являє собою функціонально закінчений фрагмент програми, оформлений у вигляді окремого файлу з початковим кодом або поійменований безперервної його частини (наприклад, Active Oberon), призначений для використання в інших програмах. Модулі дозволяють розбивати складні завдання на більш дрібні, відповідно до принципу модульності. Зазвичай проєктуються таким чином, щоб надавати програмістам зручну для багаторазового використання функціональність (інтерфейс) у вигляді набору функцій, класів, констант. Модулі можуть об'єднуватися в пакети і, далі, в бібліотеки [12].

- `connectionform.cpp` – форма, яка організує побудову зв'язку між елементами. Завдяки їй можливо задати який блок з яким треба зв'язати.
- `deviceoptform.cpp` – форма для редагування опцій пристрою, тобто там можливо вибирати генерацію пакетів, та до якого пристрою вони будуть наслані.
- `deviceoptionsdialog.cpp` – також є формою для реагування опцій маршрутизатора.
- `ipassignmentdialog.cpp` – діалог встановлення IP адрес.
- `ipmanager.cpp` – модуль, який управляє підмережами та IP адресами, завдяки йому, можливо спочатку розподілити мережу на підмережі, а потім роздати IP адреси.
- `main.cpp` – точка входу в програму.
- `mainwindow.cpp` – форма головного вікна програми
- `routertableform.cpp` – форма таблиці маршрутизації
- `wire.cpp` - об'єкт який реалізує зв'язок між усіма пристроями

- diagramitem.cpp – модуль програми, в якому реалізовані усі пристрої, та генерацію пакетів.

Для запобігання колізій на пристрої були «встановлені» буфери на 10 пакетів.

Блок схема алгоритму роботи РС зображена на рисунку 2.8.

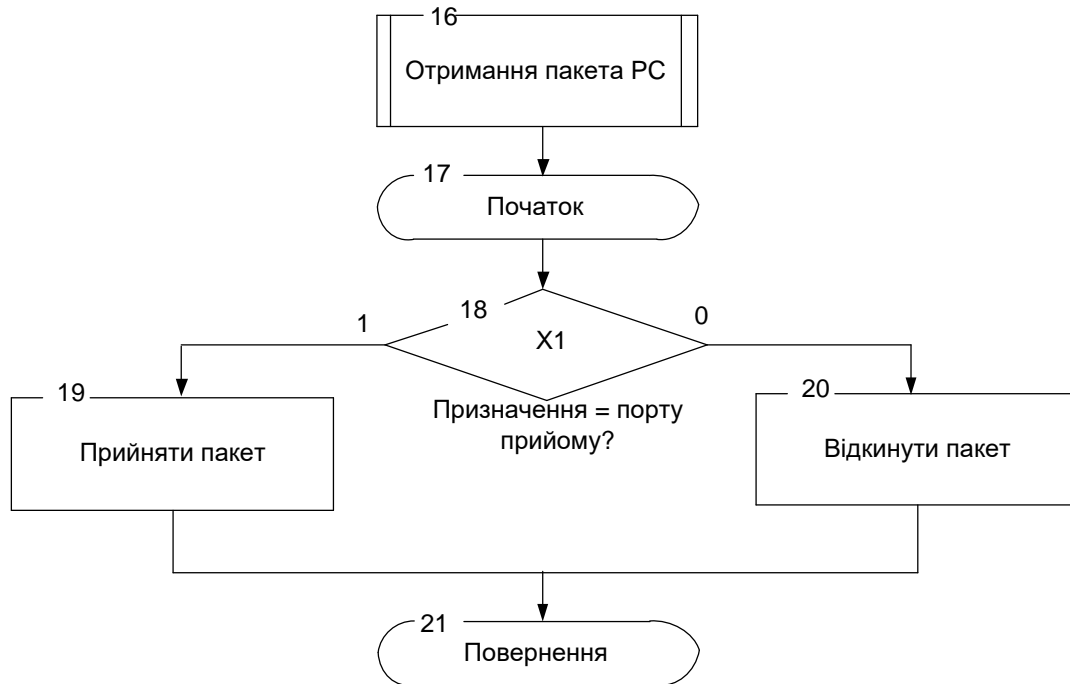


Рис. 2.8. Алгоритм роботи РС

Алгоритм роботи РС на мові с++ приведено у додатку А у лістингу файлу diagramitem.cpp.

На рисунку 2.9 наведена схема роботи роутера

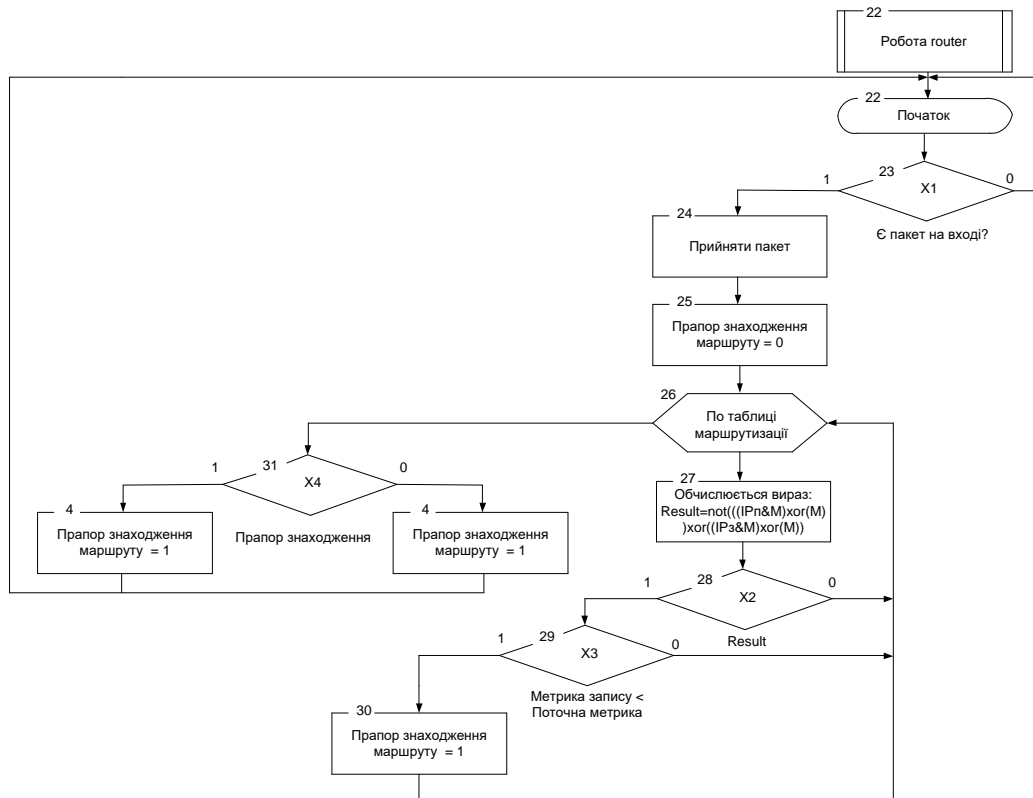


Рис. 2.9. Схема роботи роутера

Алгоритм роботи роутера на мові C++ наведено в додатку А у лістингу файла diagramitem.cpp.

```

Алгоритм генерації пакетів на мові C++:
Packet DiagramItem::generatePacket(int port)
{
    Packet p;
    p.destIP=0;
    p.length=0;
    p.sourcePort=0;
    p.sourceIP=0;
    p.HubPort=0;
    quint64 random=qrand();
    double percent=(100*random)/RAND_MAX;
    int base=0;
    for (int i=0;i<this->genOptions.count();i++)
    {
        if (genOptions.at(i).port==port)
        {
            base+=this->genOptions.at(i).percent;
            if (percent<base)
            {
                p.destIP=genOptions.at(i).destIP;
            }
        }
    }
}

```

```

p.sourcePort=genOptions.at(i).port;
p.sourceIP=this->getIP(genOptions.at(i).port);
quint64 dif=genOptions.at(i).maxlen-genOptions.at(i).minlen;
if (dif==0)
    p.length=genOptions.at(i).maxlen;
else
{
    random=qrand();
    double x=(dif*random)/RAND_MAX;
    p.length=genOptions.at(i).minlen+(quint32)x;
}
}
}
}

return p;
}

```

Алгоритм роботи хаба наведено на рисунку 2.10.

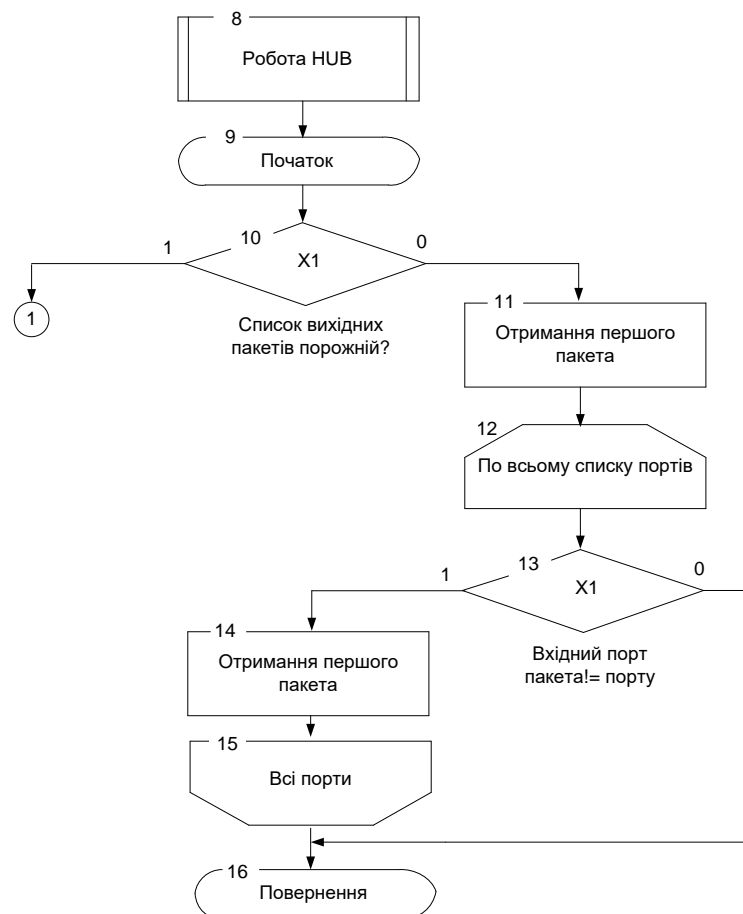


Рис. 2.10. Алгоритм роботи хаба

Алгоритм роботи хаба на мові C++:

```
void DiagramItem::ProcessHub()
{
    if (this->incommingBufferHub.isEmpty() == true)
        return;
    // if (hubreceived > 0)
    // {
    //     hubreceived--;
    //     if (hubreceived == 0)
    //         return;
    // }
    Packet p = this->incommingBufferHub.takeFirst();

    QMapIterator<int, Wire*> i(this->wires);
    while (i.hasNext()) {
        i.next();
        if (i.key() != (int)p.HubPort)
        {
            Wire *w = i.value();
            w->SendPacket(p, this);
            prevActivePort |= 1;
        }
    }
}
```

Блок схеми усіх алгоритмів знаходиться у додатку В.

Також у програмі є можливість відстежити колізію, алгоритм «піймання» колізій знаходиться у додатку А лістингу файла wire.cpp у частині коду яка відповідає за провід.

Висновки до розділу

В даному розділі розглянуто технологію **Gigabit Ethernet**. Розроблені наступні алгоритми роботи системи: роботи ПК, роботи роутера, роботи хаба та відстеження колізій. Також виконано тестування всієї системи.

РОЗДІЛ 3. РОЗРОБКА ТА АНАЛІЗ СИСТЕМИ МОДЕЛЮВАННЯ ЛОКАЛЬНОЇ МЕРЕЖІ

3.1. Розробка системи моделювання

Даний розділ дипломної роботи присвячений розробці графічної оболонки програми (GUI). Побудова графічної частини було реалізовано за допомогою редактора форм середовища Qt create.

Головне вікно програми складається з кількох частин:

- робоча панель;
- поле вибору елементів;
- робоче поле;
- консоль.

Робоча панель складається зі сполучної лінії, блоку управління моделюванням, кнопки встановлення міток (A) і встановлення IP адреси (IP). Зображена на рисунку 3.1

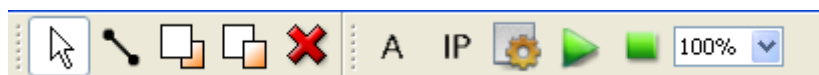


Рис. 3.1. Робоча панель

Поле вибору елементів складається з кнопок, за допомогою яких можливо будувати будь-яку мережу. Поле елементів зображене на рисунку

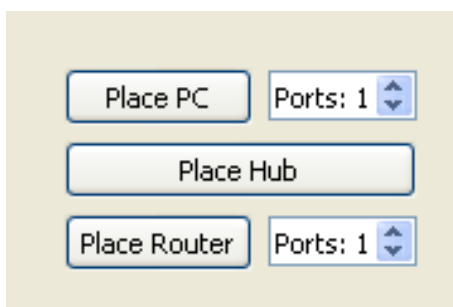


Рис. 3.2. Поле вибору елементів

Програмний код зображення поля:

```
void MainWindow::buttonGroupClicked(QAbstractButton *button)
{
    QList<QAbstractButton *> buttons = this->ui->buttonGroup->buttons();
    foreach (QAbstractButton *btn, buttons) {
        if (button!=btn)
            btn-> setChecked(false);}
}
```

```

if (button==this->ui->PCButton)
{
    this->scene->setItemType(DiagramItem::PC);
}
if (button==this->ui->HubButton)
{
    this->scene->setItemType(DiagramItem::Hub);
}
if (button==this->ui->RouterButton)
{
    this->scene->setItemType(DiagramItem::Router);
}
this->scene->setMode(DiagramScene::InsertItem);
this->ui->actionPointer->setChecked(false);
}

```

Загальний вид головного вікна зображене на рисунку 3.3

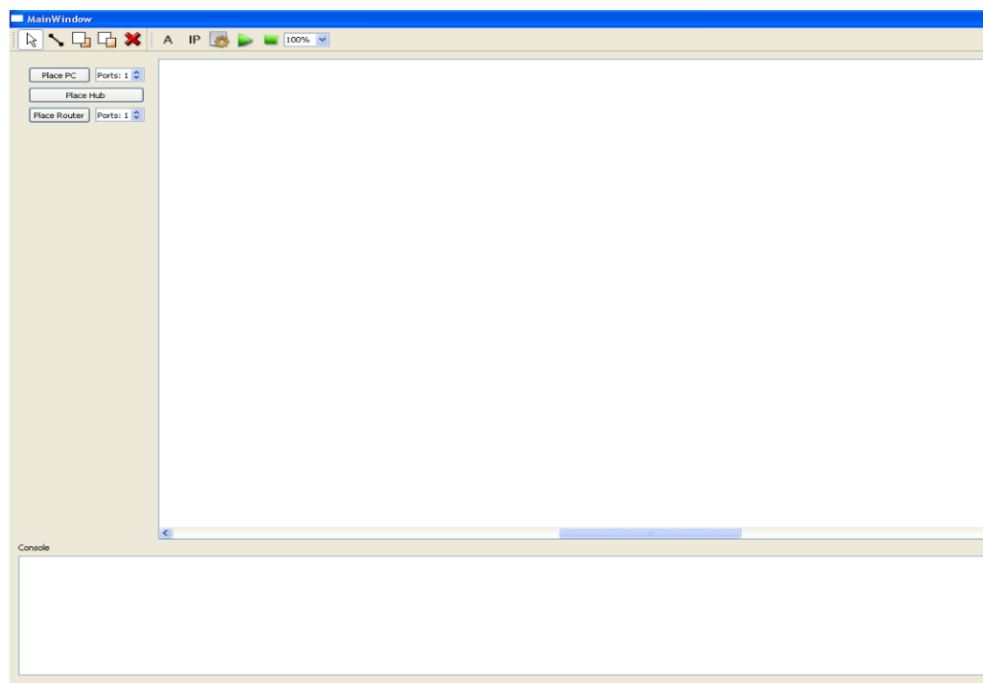


Рис. 3.3. Загальний вид головного вікна

Вікно для редагування опцій елемента має багато деталей тому реалізація цієї форми займає майже весь модуль `deviceoptform.cpp`, який буду далі описано у додатках. Детальний вид вікна редагування зображено на рисунку 3.4. В ньому можливо вибирати пункт призначення пакету, кількість згенерованих пакетів у процентному відношенні до загальної кількості, також можливо встановити мінімальний та максимальний розмір пакету.

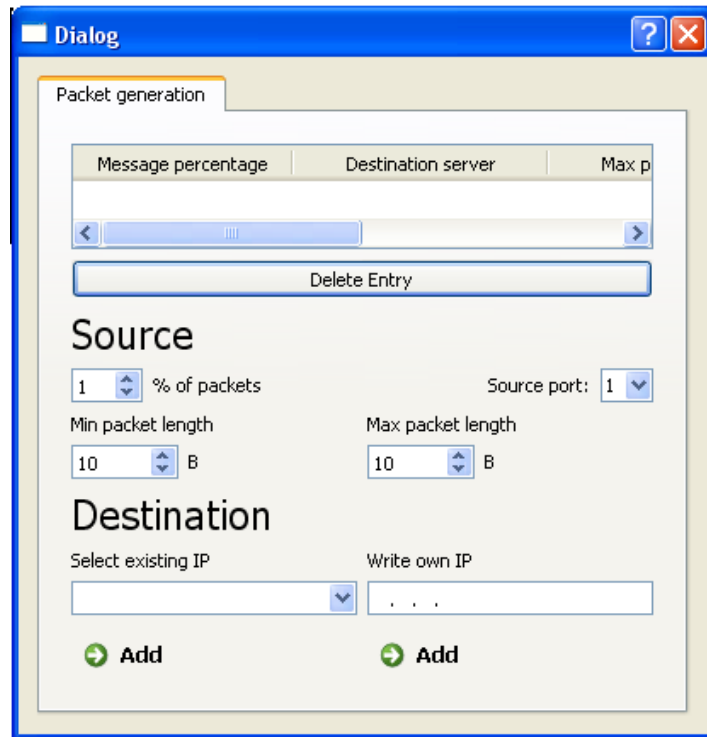


Рис. 3.4. Вікно для редагування опцій елементів

У роутера є окреме поле для редагування таблиці маршрутизацій, за це відповідає модуль `routertableform.cpp`. На рисунку 3.5 зображено таблицю маршрутизації.



Рис. 3.5. Таблиця маршрутизації роутера

Для з'єднання елементів потрібно натиснути на кнопку wire на робочій панелі при цьому з'явиться вікно, яке зображене на рисунку 3.6

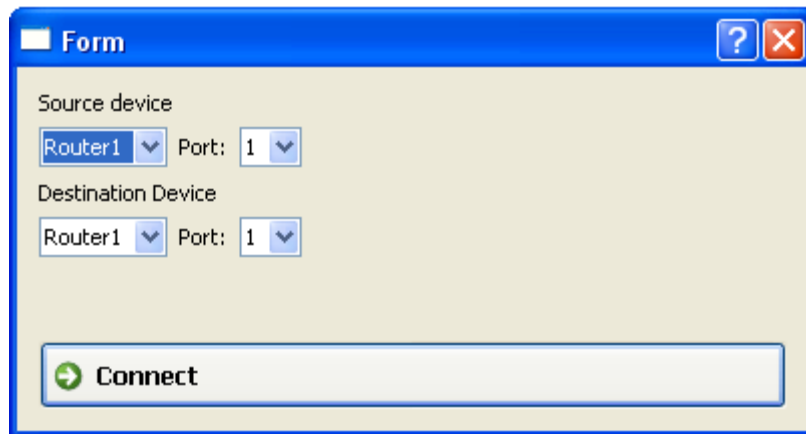


Рис. 3.6. Вікно зв'язку

В ньому можна, які порти яких елементів треба з'єднати. Опис реалізації цієї форми у модулі connectionform.cpp.

Форма призначення IP адрес зображено на рисунку 3.7



Рис. 3.7. Форма призначення IP адрес

У списку обирається потрібна підмережа, та встановлюється IP адрес, та маска для підмережі. Програмний код описаний у модулі ipmanager.cpp

Також у програмі можливо подивитись статистику, яка зображує, скільки пакетів було згенеровано, скільки пойнято, скільки пройдено скрізь роутер та хаб. На рисунку 3.8 зображене вікно статистики.

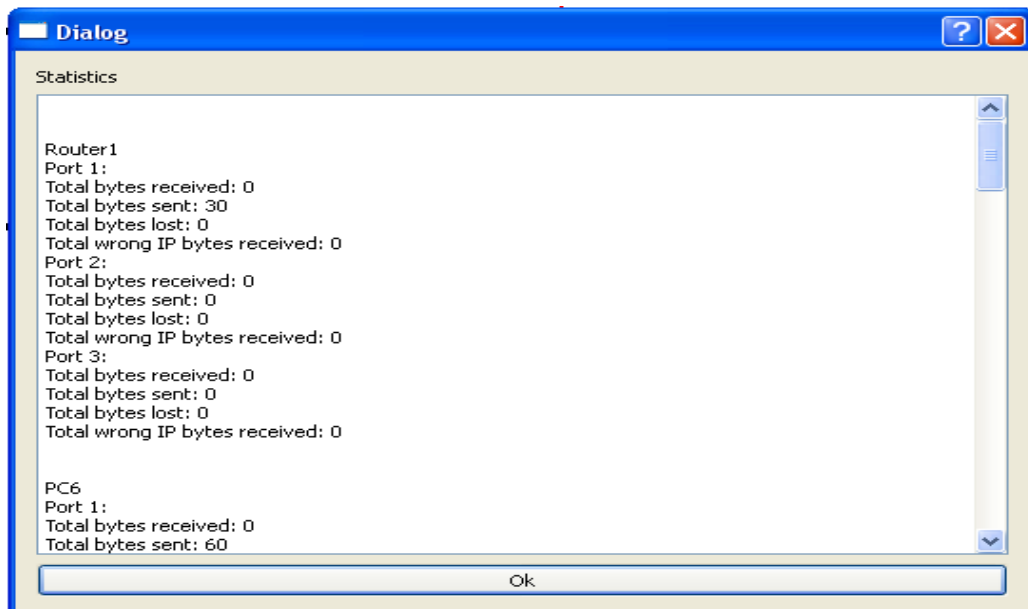


Рис. 3.8. Вікно статистики

3.2. Аналіз роботи системи моделювання

В попередніх пунктах була розглянута розробка програми. У цьому розділі наочно продемонстровано роботу програми на основі побудови мережі. Топологія обраної мережі зображена на рисунку 3.9.

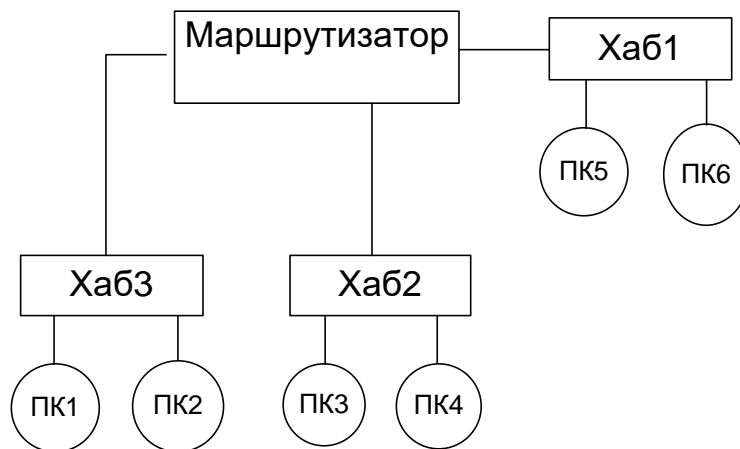


Рис. 3.9. Топологія мережі

Використовуючи програму потрібно побудувати таку ж мережу. Збудована мережа зображена на рисунку 3.10. Як бачимо з рисунку підмережі встановлені автоматично.

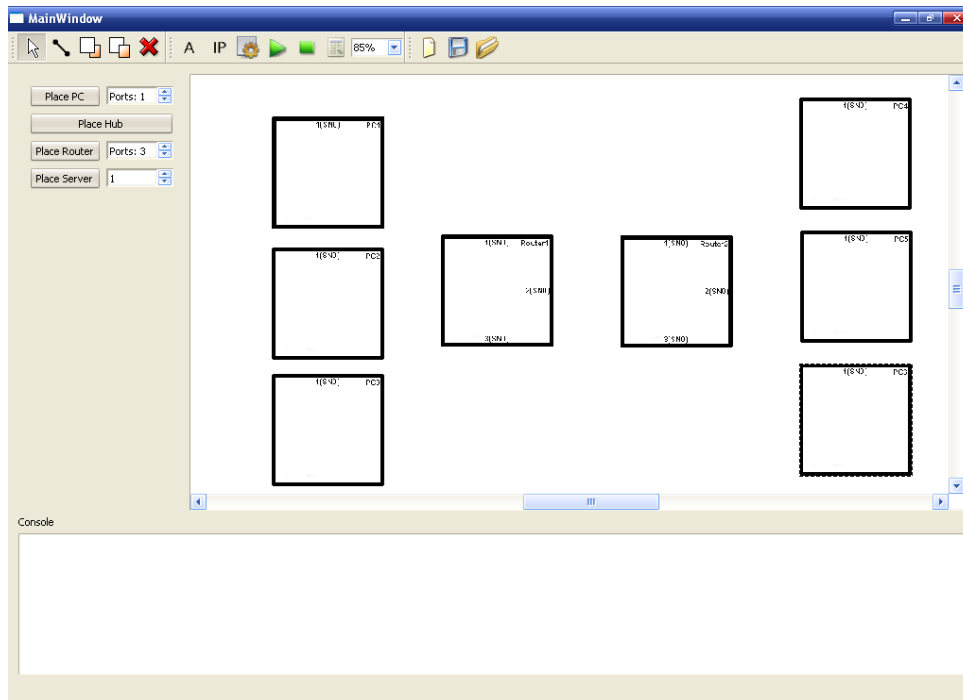


Рис. 3.10. Побудована мережа

Наступним кроком буде встановлення IP адрес, для цього потрібно встановити необхідну підмережу, та встановити для неї IP. Процес встановлення показано на рисунку 3.11. Встановимо значення IP адресу для першої підмережі 204.33.22.2 для другої 210.44.33.1, та для третьої 187.22.11.1. та маски 1, 2, 3 відповідно. Приклад встановлення IP адреси зображено на рисунку 3.11.

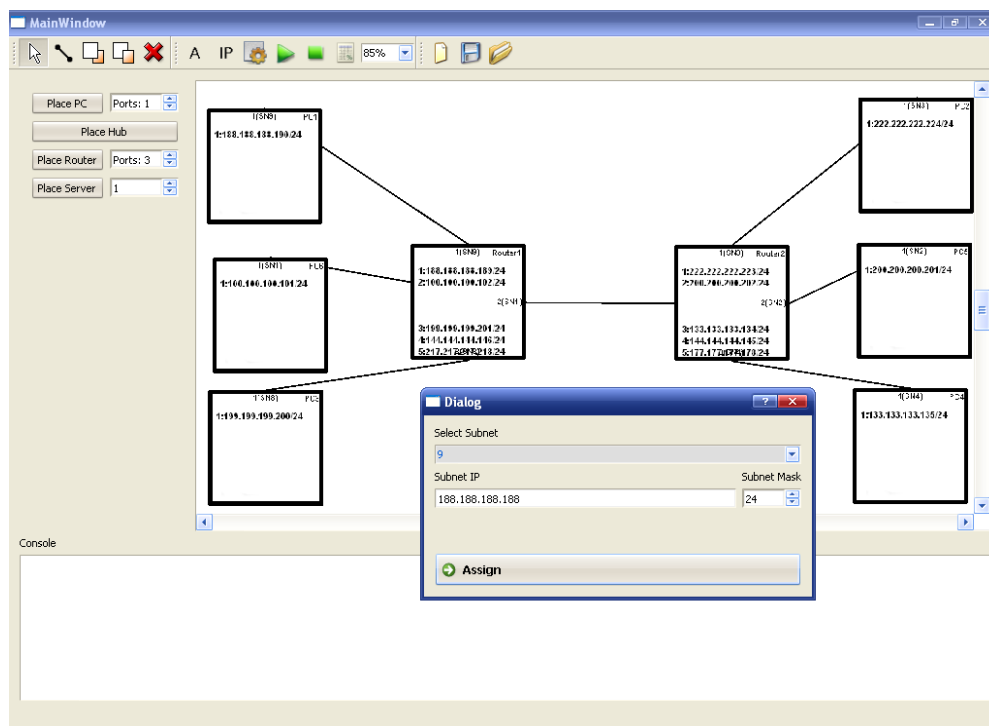


Рис. 3.11. Встановлення IP адреси

Треба заповнити таблицю маршрутизатора. Заповнена таблиця зображена на рисунку 3.12.

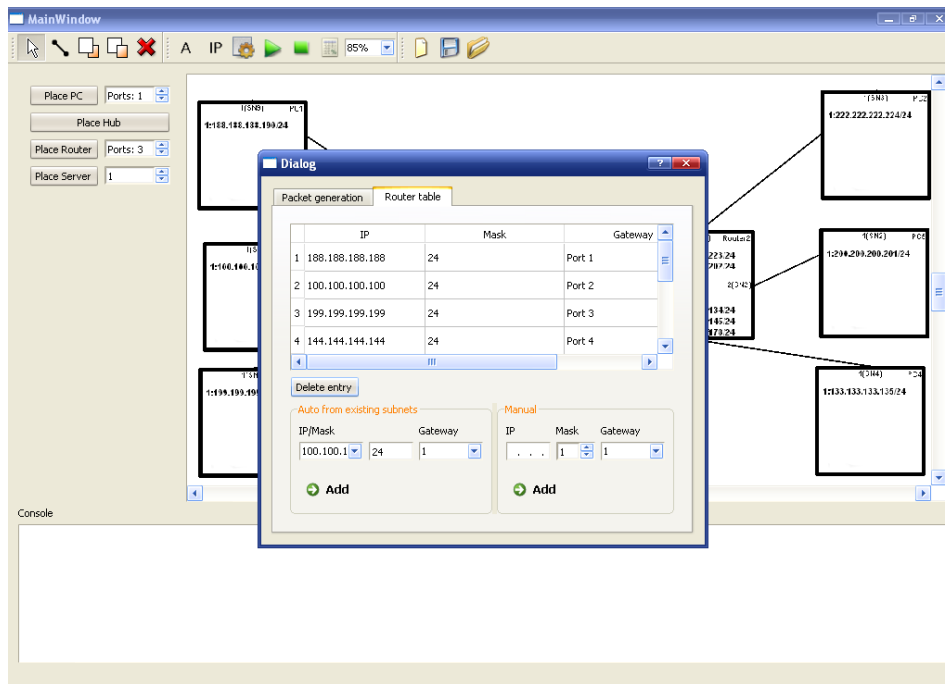


Рис. 3.12. Таблиця маршрутизації

Для перевірки відправимо пакет з PC5 до PC4, для цього треба згенерувати пакет, вказати максимальну та мінімальну довжину пакета. На рисунку 3.13 зображено форма для генерації пакета. Для перевірки кількість пакетів дорівнює 100%.

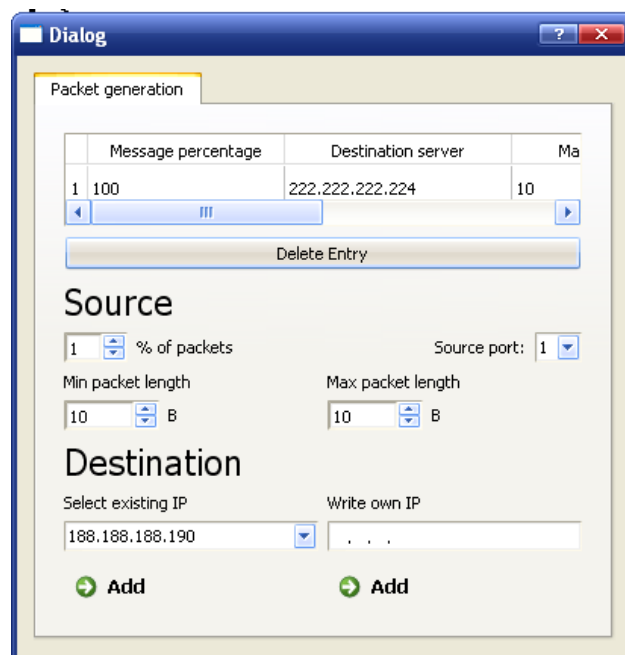


Рис. 3.13 – Генерація пакета

Рисунки 3.14 та 3.15 зображують початок та кінець моделювання. В консолі описується весь процес проходження пакету.

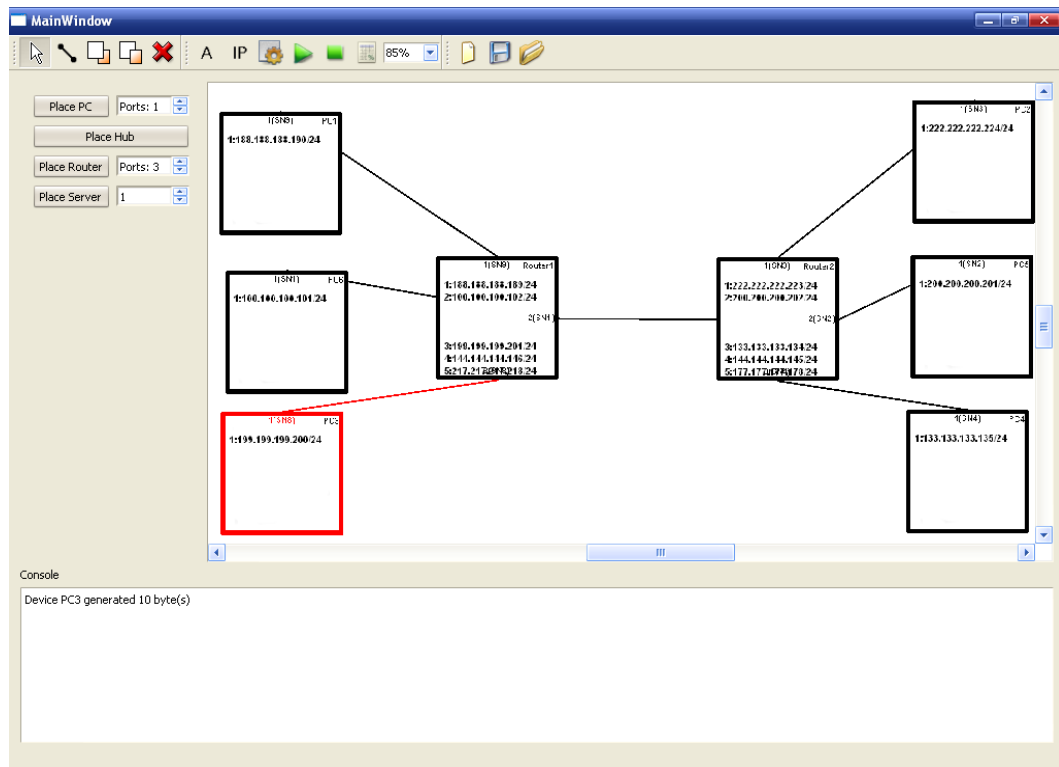


Рис. 3.14 – Початок моделювання

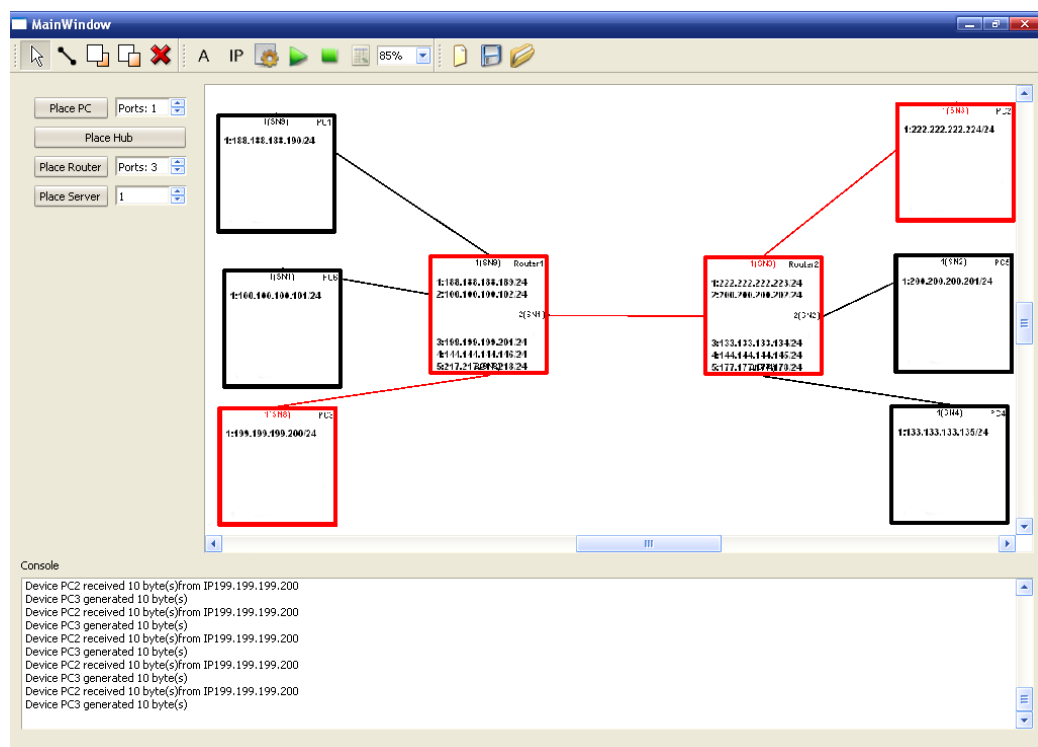


Рис. 3.15 – Кінець моделювання

Висновки до розділу

Даний розділ роботи присвячений розробці графічної оболонки програми (GUI). Побудова графічної частини було реалізовано за допомогою редактора форм середовища Qt create.

У цьому розділі покроково змодельована мережа, за заданою топологією, та проаналізована робота кожного вузла, та всієї системи в цілому.

Щодо розробленого програмного комплексу, то можна відзначити, що проєкт є досить недорогим та простим у експлуатації, але має сучасні багатофункціональні можливості.

ВИСНОВКИ

У магістерській роботі було проведено аналіз декількох вже існуючих і реально функціонуючих систем моделювання комп'ютерних мереж та було проаналізовано фізичний, математичний методи, і метод експертних оцінок. За підсумками аналізу було виділено метод імітаційного моделювання. Було з'ясовано, що на основі даного методу розроблені такі програмні засоби: CPSIM, NetDA / 2, NPAT, SES / Workbench, WinMIND, COMNET III. Була виділена і рекомендована до використання система COMNETIII на підприємствах будь-якого типу. а також була розроблена нова платформа для систем моделювання з використанням математичного методу, яка є кросплатформною. Розроблена програма дозволяє проєктувати топологічні моделі мереж і імітувати передачу пакетів по мережі з розділенням навантаження на заданих ділянках мережі. Ключовою особливістю розробленої програми є її простота та невисока ціна.

У разі подальшої роботи над програмою можливо наступне:

- створити базу даних мережевих пристроїв різних виробників: робочих станцій, серверів, середовищ передачі, мережевих адаптерів, повторювачів, мостів, комутаторів, маршрутизаторів, що використовуються в мережах типу Ethernet, з можливістю поповнення та редагування.
- прогнозування затримки між кінцевими і проміжними вузлами мережі, пропускні спроможності каналів, коефіцієнти використання сегментів, буферів і процесорів.
- підраховування піків і спадів трафіку як функцію часу, а не як усереднені значення.
- визначення джерела затримок і вузьких місць мережі.

В першому розділі були докладно розглянуті наступні системи моделювання: BONEs, Netmaker, Optimal Perfomance, Prophecy, CANE, COMNET.

В результаті проведення аналізу цих систем моделювання зроблено висновок, що даний напрям розвинутий добре, але існуючі системи мають декілька недоліків, головний з яких – це вартість. Враховуючи це, було спробовано реалізувати нову систему, яка б була більш доступна.

У другому та третьому розділі були розроблені наступні алгоритми роботи системи: роботи ПК, роботи роутера, роботи хаба та відстеження колізій. Також виконано тестування всієї системи. У цьому розділі покроково змодельована мережа, за заданою топологією, та проаналізована робота кожного вузла, та всієї системи в цілому.

Щодо розробленого програмного комплексу, то можна відзначити, що проєкт є досить недорогим та простим у експлуатації, але має сучасні багатофункціональні можливості.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. *Ethernet* [сайт вільної енциклопедії] – вхід вільний
<http://ru.wikipedia.org/wiki/Ethernet>
2. QT assistant [руководство по библиотеке QT] – вхід вільний
<http://doc.trolltech.com/3.3/assistant.html>
3. Вишне夫斯基 В.М. Теоретические основы проектирования компьютерных сетей. М.: Техносфера, 2003.- 512 с.
4. Ги К. Введение в локальные вычислительные сети. М., "Радио и связь" 1986.
5. Жасмин Бланшет, Марк Саммерфилд Qt 4: программирование GUI на C++. КУДИЦ-Пресс 641с.
6. Земсков Ю.В. Программирование на C++ с использованием библиотеки Qt. 400с.
7. Кельтон В. Имитационное моделирование. Классика CS. 3-е изд. / В. Кельтон, А. Лоу. – СПб.: Питер; К.: Издательская группа BHV, 2004. – 847 с.
8. Компьютерные сети. Учебный курс, 2-е изд. (+CD-ROM). - MicrosoftPress, Русская редакция, 1998.
9. Коробов М.Я. Фінансово-економічний аналіз діяльності підприємств: М.Я. Коробков Навч.посіб. – К.: Т-во “Знання”, КОО, 2000. – 378 с.
- 10.Кренг Х. Персональные компьютеры в сетях TCP/IP. ;/Х.Кренг перев. с англ. - BHV-Киев, 1997.
- 11.Кроссплатформенное программирование с использованием Qt [портал магістрів ДоННТУ] вхід вільний - <http://masters.donntu.edu.ua>
- 12.Липаев В.В Управление разработкой программных средств. Методы, стандарты, технология. – М.: Финансы и статистика, 1993. – 400 с.
13. Маклаков С.В. ВРwin и ERwin. CASE-средства разработки информационных систем / С.В. Маклаков. – М.: ДИАЛОГ-МИФИ, 1999. – 304 с.

- 14.Макс Шлее QT профессиональное программирование на C++. 2005г 544с.
- 15.Марк А. Высокопроизводительные сети. Энциклопедия пользователя.
А.Марк, Спортак и др.; перев. с англ. - Киев, ДиаСофт, 1998.
16. Олифер В.Г. Компьютерные сети. Принципы, технологии, протоколы. 4-е изд. / В.Г. Олифер, Н.А. Олифер. – СПб.: Питер, 2012. – 943 с.
17. Омаров О.М. Имитационная модель для проектирования и анализа инфокоммуникационных сетей / О.М. Омаров // Труды Всероссийской научно-методической конференции «Телематика-2004». – СПб.: СПбГУ ИТМО, 2004. –С. 41-42.
- 18.Организация локальных сетей на базе персональных компьютеров.
"И.В.К.- СОФТ", М., 1991.
- 19.Паронжанов С. Объектно-ориентированные средства анализа, проектирования и реинжиниринга информационных систем. – М.: Учебные материалы конференции «Индустрия программирования 96». 1996 г. с.117-123.
20. Пономаренко Л.А. Инструментальные средства проектирования, имитационного моделирования и анализа компьютерных сетей / Л.А. Пономаренко, В.И. Щелкунов, А.Я. Складов. – К.: Наукова думка, 2002. – 508 с.
21. Пылькин А.Н. Моделирование и синтез оптималь- ной структуры сети Ethernet / А.Н. Пылькин, А.В. Благодаров, Д.М. Скуднєв. – М.: Горячая Линия-Телеком, 2011. – 112 с.
- 22.Раицкий К.А. Экономика предприятия: Учебник для вузов. – М.:К.А.Рвицкий Информационно-внедренческий центр “Маркетинг”, 1999. – 693 с.
- 23.Шафрин Ю., «Основы компьютерной технологии». М., АБФ, 1997

ДОДАТОК А

Текст програми

Лістинг 1. Файл connectionform.cpp

```
#include "connectionform.h"
#include "ui_connectionform.h"
#include "diagramitem.h"
#include "diagramscene.h"
#include "mainwindow.h"
#include "wire.h"
#include <QGraphicsItem>
#include <QList>
#include <QMessageBox>

connectionForm::connectionForm(MainWindow *w,DiagramScene *s,QWidget *parent) :
    QDialog(parent),
    ui(new Ui::connectionForm)
{
    ui->setupUi(this);
    this->scene=s;
    this->window=w;
    this->FillForm();
}

void connectionForm::FillForm()
{
    this->ui->comboD->clear();
    this->ui->comboDP->clear();
    this->ui->comboS->clear();
    this->ui->comboSP->clear();

    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;
    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if (ditem)
        {
            if (ditem->diagramItemType()==DiagramItem::Hub)
            {
                this->ui->comboD->addItem(ditem->getItemStr());
                this->ui->comboS->addItem(ditem->getItemStr());
            }
            else
            {
                QList<int> ports=ditem->hasFreePorts();
                if (ports.isEmpty()==false)
                {
                    this->ui->comboD->addItem(ditem->getItemStr());
                    this->ui->comboS->addItem(ditem->getItemStr());
                }
            }
        }
    }
}
```

```

connectionForm::~~connectionForm()
{
    delete ui;
}
void connectionForm::changeEvent(QEvent *e)
{
    QWidget::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}
void connectionForm::FillPortCombo(QComboBox *box,QString item)
{
    box->clear();

    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;
    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if ((ditem)&&(ditem->getItemStr()==item))
        {
            if (ditem->diagramItemType()==DiagramItem::Hub)
                break;
            QList<int> ports=ditem->hasFreePorts();
            foreach (int tmpport,ports)
            {
                QString str;
                box->addItem(str.setNum(tmpport));
            }
            break;
        }
    }
}

void connectionForm::on_comboD_currentIndexChanged(QString str)
{
    this->FillPortCombo(this->ui->comboDP,str);
}

void connectionForm::on_comboS_currentIndexChanged(QString str)
{
    this->FillPortCombo(this->ui->comboSP,str);
}

void connectionForm::on_Connect_clicked()
{
    QString strD=this->ui->comboD->currentText();
    QString strS=this->ui->comboS->currentText();
    QString strDP=this->ui->comboDP->currentText();
    QString strSP=this->ui->comboSP->currentText();
    if ((strD.isEmpty())||(strS.isEmpty()))
    {

```

```

    QMessageBox::critical(0, tr("Error"),tr("Select device"));
    return;
}
if (strD==strS)
{
    QMessageBox::critical(0, tr("Error"),tr("Cannot connect same devices"));
    return;
}
QList<QGraphicsItem *> items=this->scene->items();
QGraphicsItem * tmp;
DiagramItem* ditem,*itemS=0,*itemD=0;
int portS=0,portD=0;
foreach (tmp,items)
{
    ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
    if ((ditem)&&(ditem->getItemStr()==strD))
        itemD=ditem;
    if ((ditem)&&(ditem->getItemStr()==strS))
        itemS=ditem;
}
if ((itemD==0)|| (itemS==0))
{
    QMessageBox::critical(0, tr("Error"),tr("Oooops. This shouldn't happen!"));
    return;
}
if ((itemD->diagramItemType()!=DiagramItem::Hub)&&(strDP.isEmpty()))
{
    QMessageBox::critical(0, tr("Error"),tr("Oooops. This shouldn't happen!"));
    return;
}
else
    portD=strDP.toInt();
if ((itemS->diagramItemType()!=DiagramItem::Hub)&&(strSP.isEmpty()))
{
    QMessageBox::critical(0, tr("Error"),tr("Oooops. This shouldn't happen!"));
    return;
}
else
    portS=strSP.toInt();
// if ((portS==0)|| (portD==0))
// {
//     QMessageBox::critical(0, tr("Error"),tr("Oooops. This shouldn't happen!"));
//     return;
// }
if ((itemD->diagramItemType()==DiagramItem::Hub)&&(itemS->diagramItemType()==DiagramItem::Hub))
{
    if (itemD->isDeviceConnected(itemS)==true)
    {
        QMessageBox::critical(0, tr("Error"),tr("Hubs are already connected"));
        return;
    }
}
Wire *w=new Wire(scene,itemS,itemD);
w->setZValue(qMin(itemS->zValue(),itemD->zValue())-0.1);
scene->addItem(w);
itemD->addWire(w,portD);
itemS->addWire(w,portS);
w->updatePosition();
QObject::connect(w,SIGNAL(logMessage(QString)),this->window,SLOT(appendLog(QString)));
this->FillForm();}

```

Лістинг 2. Файл deviceoptform.cpp

```
#include <QtGui>
#include "deviceoptform.h"
#include "ui_deviceoptform.h"
#include "diagramitem.h"
#include "diagramscene.h"
#include "general.h"
#include "ipmanager.h"

deviceOptForm::deviceOptForm(DiagramScene *s, DiagramItem *d, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::deviceOptForm)
{
    ui->setupUi(this);
    this->scene=s;
    this->ditem=d;
    this->fillData();
    if ((this->ditem)&&(this->ditem->diagramItemType()!=DiagramItem::Hub))
    {
        IPManager m(this->scene);

        QList<quint32> ips=m.GetAllIPs();
        for (int i=0;i<ips.count();i++)
        {
            this->ui->comboBoxA->addItem(IPManager::parseIP(ips.at(i)));
        }
        int portCnt=ditem->getPortCounter();
        for (int i=1;i<=portCnt;i++)
        {
            QString str;
            str=str.setNum(i);
            this->ui->comboBoxSP->addItem(str);
        }
    }
}

deviceOptForm::~deviceOptForm()
{
    delete ui;
}

void deviceOptForm::changeEvent(QEvent *e)
{
    QWidget::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void deviceOptForm::fillData()
{
    this->ui->tableWidget->clearContents();
    QStringList labels;
    labels.clear();
    labels << tr("Message percentage") << tr("Destination server")<<tr("Max packet length")<<tr("Min packet
length")<<tr("Port")<<tr("ID");
    this->ui->tableWidget->setColumnCount(labels.size());
}
```

```

this->ui->tableWidget->setHorizontalHeaderLabels(labels);
this->ui->tableWidget->setColumnWidth(0,130);
this->ui->tableWidget->setColumnWidth(1,150);
this->ui->tableWidget->setColumnWidth(2,150);
this->ui->tableWidget->setColumnWidth(3,150);
this->ui->tableWidget->setColumnWidth(4,50);
this->ui->tableWidget->setColumnWidth(deviceOptForm::ColForID,50);
if ((this->ditem)&&(this->ditem->diagramItemType()!=DiagramItem::Hub))
{
    QList <genOption> lst=ditem->getGenOptionsList();
    this->ui->tableWidget->setRowCount(lst.count());
    genOption opt;
    int counter=0;
    foreach (opt,lst)
    {
        QString str;
        str.setNum(opt.percent);
        QTableWidgetItem *newItem = new QTableWidgetItem(str);
        this->ui->tableWidget->setItem(counter, 0, newItem);

        str=IPManager::parseIP(opt.destIP);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 1, newItem);

        str=str.setNum(opt.maxlen);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 2, newItem);

        str=str.setNum(opt.minlen);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 3, newItem);
        str=str.setNum(opt.port);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 4, newItem);
        str=str.setNum(opt.ID);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, deviceOptForm::ColForID, newItem);
        counter++;
    }
}
}
void deviceOptForm::on_commandLinkButton_clicked()
{
    QString destipStr=this->ui->comboBoxA->currentText();
    this->AddEntry(destipStr);
}
void deviceOptForm::on_commandLinkButton_2_clicked()
{
    QString destipStr=this->ui->lineEdit->text();
    this->AddEntry(destipStr);
}
void deviceOptForm::AddEntry(QString ipStr)
{
    quint32 ip=IPManager::parseIP(ipStr);
    if (ip==0)
    {
        QMessageBox::critical(0, tr("Error"),tr("Wrong IP"));
        return;
    }
    QString str=this->ui->comboBoxSP->currentText();
    quint8 port=str.toInt();

```

```

if (port==0)
{
    QMessageBox::critical(0, tr("Error"),tr("Wrong source port"));
    return;
}
quint32 maxL=this->ui->spinBoxMaxL->value();
quint32 minL=this->ui->spinBoxMinL->value();
if (maxL<minL)
{
    QMessageBox::critical(0, tr("Error"),tr("Max packet length cannot be lower than min packet length"));
    return;
}
quint8 percent=this->ui->spinBoxP->value();

QList <genOption> optLst=ditem->getGenOptionsList();
int totalP=percent;
for (int i=0;i<optLst.count();i++)
{
    if (optLst.at(i).port==port)
        totalP+=optLst.at(i).percent;
}
if (totalP>100)
{
    QMessageBox::critical(0, tr("Error"),tr("Percentage is too big"));
    return;
}
quint32 ID;
if (optLst.isEmpty())
{
    ID=0;
}
else
{
    QList<int> intList;
    for (int i=0;i<optLst.count();i++)
    {
        intList.append(optLst.at(i).ID);
    }
    qSort(intList.begin(), intList.end());
    int prefID=0;
    int num;
    foreach(num,intList)
    {
        if (prefID!=num)
        {
            ID=prefID;
            break;
        }
        prefID++;
    }
    ID=prefID;
}
genOption option;
option.destIP=ip;
option.maxlen=maxL;
option.minlen=minL;
option.percent=percent;
option.port=port;
option.ID=ID;
this->ditem->addGenOption(option);
this->fillData();

```

```

}
void deviceOptForm::on_pushButton_clicked()
{
    int selectedRow=this->ui->tableWidget->currentRow();
    if (selectedRow==-1)
    {
        QMessageBox::critical(0, tr("Error"),tr("No row selected."));
        return;
    }
    QTableWidgetItem * itm = this->ui->tableWidget->item(selectedRow,deviceOptForm::ColForID);
    QString str=itm->text();
    quint32 ID=str.toInt();
    QList <genOption> optLst=ditem->getGenOptionsList();
    for (int i=0;i<optLst.count();i++)
    {

        if (optLst.at(i).ID==ID)
        {
            this->ditem->removeGenOption(ID);
            break;
        }
    }
    this->fillData();
}

```

Лістинг 3. Файл diagramitem.cpp

```
#include <QtGui>
#include "mainwindow.h"
#include "diagramitem.h"
#include "wire.h"
#include "diagramscene.h"
#include "ipmanager.h"

DiagramItem::DiagramItem(DiagramItemType itemType,MainWindow *w,DiagramScene *sc,QMenu
*contextMenu,QGraphicsItem *parent, int itemNum,quint8 portCnt)
: QGraphicsPixmapItem(parent)
{

    myContextMenu = contextMenu;
    myDiagramItemType=itemType;
    this->scene=sc;
    prevActivePort=0;

    for (int i=0;i<4;i++)
    {
        totalReceived[0]=0;
        totalSent[0]=0;
        totalLost[0]=0;
        totalReceivedWrong[i]=0;
    }
    //hubreceived=0;

    // QPainterPath path;
    // myPolygon << QPointF(-70, -70) << QPointF(70, -70)
    // << QPointF(70, 70) << QPointF(-70, 70)
    // << QPointF(-70, -70);
    setFlag(QGraphicsItem::ItemIsMovable, true);
    setFlag(QGraphicsItem::ItemIsSelectable, true);

    for (int i=0;i<4;i++)
        subnets[i]=0;

    if (itemNum==-1)
    {
        if (scene!=0)
        {
            QList<QGraphicsItem *> items= scene->items();
            QList<quint16> itemNums;
            QGraphicsItem * item;
            DiagramItem* ditem;
            foreach(item,items)
            {
                ditem=qgraphicsitem_cast<DiagramItem *>(item);
                if (ditem==0)continue;
                if (ditem!=this)
                    if (ditem->diagramItemType()==this->diagramItemType())
                        itemNums.append(ditem->getItemNumber());
            }
            if (itemNums.isEmpty())
                this->itemNumber=1;
            else
            {
                qSort(itemNums.begin(), itemNums.end());
                bool numIsSet=false;
                quint16 prefferedNum=1;
            }
        }
    }
}
```



```

        quint16 num=0;
        foreach (num,itemNums)
        {
            if (prefferedNum<num)
            {
                this->itemNumber=prefferedNum;
                numIsSet=true;
            }
            else
                prefferedNum=++num;
        }
        if (numIsSet==false)
            this->itemNumber=num;
    }
    else
        this->itemNumber=1;
}
else
    this->itemNumber=itemNum;

switch (itemType)
{
case PC:
    this->itemTypeString="PC";
    break;
case Router:
    this->itemTypeString="Router";
    break;
default:
    this->itemTypeString="Hub";
    break;
}

if (this->myDiagramItemType==DiagramItem::Hub)
    portCounter=0;
else
    portCounter=portCnt;
QString str;
this->itemTypeString=itemTypeString+str.setNum(this->itemNumber);

QPixmap* pixmap=this->buildPixmap(false,0);
setPixmap(*pixmap);
delete pixmap;

QObject::connect(this,SIGNAL(logMessage(QString)),
    w,SLOT(appendLog(QString)));
}

DiagramItem::~DiagramItem()
{
    disconnect(this, 0, 0, 0);
    this->removeWires();
    //this->scene->removeItem(this);
}

QPixmap * DiagramItem::buildPixmap(bool active,quint8 activePort)
{

```

```

QPixmap* pixmap;

if (active==true)
    pixmap=new QPixmap("./images/deviceActive.png");
else
    pixmap=new QPixmap("./images/devicePassive.png");

QPainter painter(pixmap);
painter.setPen(Qt::black);
painter.setFont(QFont("Arial", 7, QFont::Bold));
QRectF rec(0,5,pixmap->width()-5,pixmap->height());
QString str=itemTypeString;
painter.drawText(rec,str,QTextOption(Qt::AlignRight));

if (this->myDiagramItemType!=DiagramItem::Hub)
{
    QString st;
    if (this->portCounter>=1)
    {
        if (activePort&0x1)
            painter.setPen(Qt::red);
        QRectF rec(0,5,pixmap->width(),10);
        QString str="1";
        str=str+"(SN"+st.setNum(this->subnets[0])+")";
        painter.drawText(rec,str,QTextOption(Qt::AlignCenter));
        painter.setPen(Qt::black);
    }
    if (this->portCounter>=2)
    {
        if (activePort&0x2)
            painter.setPen(Qt::red);
        QRectF rec(0,0,pixmap->width()-5,pixmap->height());
        QString str="2";
        str=str+"(SN"+st.setNum(this->subnets[1])+")";
        painter.drawText(rec,str,QTextOption(Qt::AlignRight|Qt::AlignVCenter));
        painter.setPen(Qt::black);
    }
    if (this->portCounter&0x4)
    {
        if (activePort==3)
            painter.setPen(Qt::red);
        QRectF rec(0,pixmap->height()-15,pixmap->width(),10);
        QString str="3";
        str=str+"(SN"+st.setNum(this->subnets[2])+")";
        painter.drawText(rec,str,QTextOption(Qt::AlignCenter));
        painter.setPen(Qt::black);
    }
    if (this->portCounter&0x8)
    {
        if (activePort==4)
            painter.setPen(Qt::red);
        QRectF rec(5,0,pixmap->width(),pixmap->height());
        QString str="4";
        str=str+"(SN"+st.setNum(this->subnets[3])+")";
        painter.drawText(rec,str,QTextOption(Qt::AlignLeft|Qt::AlignVCenter));
        painter.setPen(Qt::black);
    }
}
}

QMap<int,IP_struct>::const_iterator i;

```

```

QString buf;
str="";
for (int j=1;j<=this->portCounter;j++)
{
    i=this->ip.find(j);
    if (i!=ip.end())
    {
        buf=buf.setNum(j)+":";
        str=str+buf;
        buf=IPManager::parseIP(i.value().ip);
        str=str+buf+"/";
        buf=buf.setNum(i.value().mask);
        str=str+buf;
        str=str+"\n";
        if (j==2)
            str=str+"\n\n";
    }
}

QRectF rec1(5.f,25.f,(float)pixmap->width()-25.f,(float)pixmap->height()-25.f);
painter.setPen(Qt::black);
painter.setFont(QFont("Arial", 8, QFont::Bold));
painter.drawText(rec1,str,QTextOption(Qt::AlignHCenter|Qt::AlignTop));

painter.end();

width=pixmap->width();
height=pixmap->height();
return pixmap;
}

void DiagramItem::removeWires()
{
    QMapIterator<int,Wire*> i(this->wires);
    Wire *w;
    while (i.hasNext()) {
        i.next();
        w=i.value();
        w->removeConnectedItems();
        this->scene->removeItem(w);
        delete w;
    }
}

void DiagramItem::removeWire(Wire *w)
{
    QMapIterator<int,Wire*> i(this->wires);
    while (i.hasNext()) {
        i.next();
        if (i.value()==w)
        {
            this->wires.remove(i.key());
            break;
        }
    }
}

```

```

QPoint DiagramItem::getWirePos(Wire *w)
{
    QPointF p1= this->pos();
    QPoint p((int)p1.x(),(int)p1.y());
    QMapIterator<int,Wire*> i(this->wires);
    if (this->myDiagramItemType==DiagramItem::Hub)
    {
        p.setX(p.x()+70);
        p.setY(p.y()+70);
        return p;
    }
    while (i.hasNext()) {
        i.next();
        if (i.value()==w)
        {
            switch (i.key())
            {
                case 1:p.setX(p.x()+width/2);break;
                case 2:p.setX(p.x()+width);p.setY(p.y()+height/2);break;
                case 3:p.setX(p.x()+width/2);p.setY(p.y()+height);break;
                case 4:p.setY(p.y()+height/2);break;
            }

            break;
        }
    }
    return p;
}

QVariant DiagramItem::itemChange(GraphicsItemChange change,
                                const QVariant &value)
{
    if (change == QGraphicsItem::ItemPositionChange) {
        QMapIterator<int,Wire*> i(this->wires);
        while (i.hasNext()) {
            i.next();
            i.value()->updatePosition();
        }
    }

    return value;
}

QList<int> DiagramItem::hasFreePorts()
{
    QList<int> freePorts;
    if (this->myDiagramItemType!=DiagramItem::Hub)
        for (int i=1;i<=this->portCounter;i++)
        {
            if (this->wires.find(i)==this->wires.end())
                freePorts.append(i);
        }
    return freePorts;
}

IP_descriptor DiagramItem::getFromWire(Wire *w)
{
    IP_descriptor d;
    d.d=this;
    d.port=0;
}

```

```

 QMapIterator<int,Wire*> i(this->wires);
 while (i.hasNext()) {
     i.next();
     if (i.value()==w)
     {
         d.d=this;
         d.port=i.key();
         return d;
     }
 }
 return d;
}

quint8 DiagramItem::getPortCounter()
{
    return portCounter;
}

bool DiagramItem::hasIP(quint8 port)
{
    if (ip.find(port)==ip.end())
        return false;
    else
        return true;
}

IP_descriptor DiagramItem::getConnectedItem(quint8 port)
{
    //Wire* w;
    QMap<int, Wire*>::const_iterator i=this->wires.find(port);
    //w=
    if (i!=wires.end())
    {
        return i.value()->getConnectedItem(this);
    }

    IP_descriptor d;
    d.d=0;
    d.port=0;

    return d;
}

QList<IP_descriptor> DiagramItem::getConnectedDevices_HUB()
{
    QList<IP_descriptor> lst;

    IP_descriptor itm;
    if (this->myDiagramItemType!=DiagramItem::Hub)
        return lst;

    QList<IP_descriptor> lsth;

    QMapIterator<int,Wire*> i(this->wires);
    while (i.hasNext()) {
        i.next();
        itm=i.value()->getConnectedItem(this);
        if (itm.d)
            lsth.append(itm);
    }
}

```

```

    return lsth;

}

int DiagramItem::getSubnet(int port)
{
    if (port>4)return 0;
    return this->subnets[port-1];
}

void DiagramItem::setSubnet(int port,int subnet)
{
    if (port>4)return;
    this->subnets[port-1]=subnet;

    QPixmap* pixmap=this->buildPixmap(false,0);
    setPixmap(*pixmap);
    delete pixmap;
}

void DiagramItem::ClearIP()
{
    this->ip.clear();
    for (int i=0;i<4;i++)
        subnets[i]=0;

    QPixmap* pixmap=this->buildPixmap(false,0);
    setPixmap(*pixmap);
    delete pixmap;
}

void DiagramItem::setIP(IP_struct s,int port)
{
    this->ip.insert(port,s);
    QPixmap* pixmap=this->buildPixmap(false,0);
    setPixmap(*pixmap);
    delete pixmap;
}

quint32 DiagramItem::getIP(int port)
{
    QMap<int, IP_struct>::const_iterator i=this->ip.find(port);
    //w=
    if (i!=ip.end())
        return i.value().ip;
    else
        return 0;
}

void DiagramItem::removeGenOption(quint32 ID)
{
    for (int i=0;i<this->genOptions.count();i++)
    {
        if (genOptions.at(i).ID==ID)
        {
            this->genOptions.removeAt(i);
            break;
        }
    }
}

```

```

}

void DiagramItem::removeRouterEntry(quint32 ID)
{
    for (int i=0;i<this->routerTable.count();i++)
    {
        if (routerTable.at(i).ID==ID)
        {
            this->routerTable.removeAt(i);
            break;
        }
    }
}

bool DiagramItem::isDeviceConnected(DiagramItem *d)
{
    IP_descriptor itm;

    QMapIterator<int,Wire*> i(this->wires);
    while (i.hasNext()) {
        i.next();
        itm=i.value()->getConnectedItem(this);
        if (itm.d==d)
            return true;
    }
    return false;
}

void DiagramItem::BeginWork()
{
    if (this->myDiagramItemType!=DiagramItem::Hub)
    {
        for (int i=1;i<=this->portCounter;i++)
        {
            Packet p=this->generatePacket(i);
            if (p.length!=0)
            {
                if (p.destIP==this->getIP(i))
                    this->incommingBuffer[p.sourcePort-1].append(p);
                else
                    this->outgoingBuffer[p.sourcePort-1].append(p);
                emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" generated %1 byte(s)")
                    .arg(p.length));
            }
        }
    }

    switch (this->myDiagramItemType)
    {
        case DiagramItem::Hub:ProcessHub();break;
        case DiagramItem::Router:ProcessRouter();break;
        case DiagramItem::PC:ProcessPC();break;
    }
}

void DiagramItem::addWire (Wire *w,int port)
{

```

```

int p=port;
if (this->diagramItemType()==DiagramItem::Hub)
{
    p=this->wires.count()+1;
    this->portCounter=p;
}
this->wires.insertMulti(p,w);
}

```

```

void DiagramItem::ProcessPC()
{
    for (int i=1;i<=this->portCounter;i++)
    {
        if (this->incommingBuffer[i-1].isEmpty()==false)
        {
            Packet p=this->incommingBuffer[i-1].takeFirst();
            if (p.destIP==this->getIP(i))
            {
                emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" received %1 byte(s)")
                    .arg(p.length) + QObject::tr("from IP") + IPManager::parseIP(p.sourceIP));
                this->totalReceived[i-1] += p.length;
            }
            else
            {
                emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" received %1 byte(s)")
                    .arg(p.length) + QObject::tr(" from IP ") + IPManager::parseIP(p.sourceIP) + QObject::tr(" which were
sent to the wrong IP - ") + IPManager::parseIP(p.destIP));

                this->totalReceivedWrong[i-1] += p.length;

                switch (i)
                {
                    {
                        case 1:prevActivePort&=~0x1;break;
                        case 2:prevActivePort&=~0x2;break;
                        case 3:prevActivePort&=~0x4; break;
                        case 4:prevActivePort&=~0x8; break;
                    }
                    //this->prevActivePort=
                }
            }
        }

        if (this->outgoingBuffer[i-1].isEmpty()==false)
        {
            Packet p=this->outgoingBuffer[i-1].takeFirst();
            if (p.destIP==this->getIP(i-1))
            {
                emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" received %1 byte(s)")
                    .arg(p.length) + QObject::tr("from IP") + IPManager::parseIP(p.sourceIP));
                this->totalReceived[i-1] += p.length;
                continue;
            }
            Wire *w=this->getWire(i);
            if (w!=0)
            {
                w->SendPacket(p,this);
                this->totalSent[i-1] += p.length;
                switch (i)
                {
                    {
                        case 1:prevActivePort|=0x1;break;

```



```

        case 2:prevActivePort|=0x2;break;
        case 3:prevActivePort|=0x4; break;
        case 4:prevActivePort|=0x8; break;
    }
}
}
}

void DiagramItem::ProcessRouter()
{
    for (int i=1;i<=this->portCounter;i++)
    {
        if (this->incommingBuffer[i-1].isEmpty()==false)
        {
            Packet p=this->incommingBuffer[i-1].takeFirst();
            if (p.destIP==this->getIP(i))
            {
                emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr("received %1 byte(s)")
                    .arg(p.length) + QObject::tr("from IP") + IPManager::parseIP(p.sourceIP));
                this->totalReceived[i-1] += p.length;
            }
            else
            {
                bool found=false;
                for (int j=0;j<this->routerTable.count();j++)
                {
                    quint32 mask=IPManager::parseMask(routerTable.at(j).mask);
                    quint32 rip=routerTable.at(j).ip&mask;
                    quint32 dip=p.destIP&mask;
                    if (rip==dip)
                    {
                        //this->AddOutgoingPacket(p,routerTable.at(j).port);
                        this->AddOutgoingPacketRouter(p,routerTable.at(j).port);
                        found=true;
                        break;
                    }
                }
                if (!found)
                {
                    emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" received %1 byte(s)")
                        .arg(p.length) + QObject::tr("from IP") + IPManager::parseIP(p.sourceIP) + QObject::tr("which
were sent to the wrong IP - ") + IPManager::parseIP(p.destIP));

                    this->totalReceivedWrong[i-1] += p.length;
                }
            }
        }
    }

    if (this->outgoingBuffer[i-1].isEmpty()==false)
    {
        Packet p=this->outgoingBuffer[i-1].takeFirst();
        Wire *w=this->getWire(i);
        if (p.destIP==this->getIP(i-1))
        {
            emit logMessage(QObject::tr("Device ") + this->itemTypeString + QObject::tr(" received %1 byte(s)")
                .arg(p.length) + QObject::tr("from IP") + IPManager::parseIP(p.sourceIP));
            this->totalReceived[i-1] += p.length;
        }
    }
}

```

```

        continue;
    }
    if (w!=0)
    {
        w->SendPacket(p,this);
        this->totalSent[i-1]+=p.length;
        switch (i)
        {
            case 1:prevActivePort|=0x1;break;
            case 2:prevActivePort|=0x2;break;
            case 3:prevActivePort|=0x4; break;
            case 4:prevActivePort|=0x8; break;
        }
    }
}
}
}
}

```

```

void DiagramItem::AddOutgoingPacket(Packet p,int port)
{
    if (this->outgoingBuffer[port-1].length()>=DiagramItem::MAX_BUFFER_LENGTH)
    {
        emit logMessage(QObject::tr("Device's' ") + this->itemTypeString + QObject::tr("buffer exceeded. It has lost 1 packet
to ")
            + IPManager::parseIP(p.destIP) + QObject::tr("from ") + IPManager::parseIP(p.sourceIP));
        this->totalLost[port-1] += p.length;
    }
    else
    {
        if (this->myDiagramItemType==DiagramItem::Hub)
            p.HubPort=port;
        this->outgoingBuffer[port-1].append(p);
    }
}

```

```

void DiagramItem::AddOutgoingPacketRouter(Packet p,int port)
{
    if (this->myDiagramItemType==DiagramItem::Router)
    {
        if (this->outgoingBuffer[port-1].length()>=DiagramItem::MAX_BUFFER_LENGTH)
        {
            emit logMessage(QObject::tr("Device's' ") + this->itemTypeString + QObject::tr("buffer exceeded. It has lost 1
packet to ")
                + IPManager::parseIP(p.destIP) + QObject::tr("from ") + IPManager::parseIP(p.sourceIP));
            this->totalLost[port-1] += p.length;
        }
        else
        {
            this->outgoingBufferRouter[port-1].append(p);
        }
    }
}

```

```

Packet DiagramItem::generatePacket(int port)
{
    Packet p;
    p.destIP=0;
    p.length=0;
    p.sourcePort=0;
    p.sourceIP=0;
    p.HubPort=0;
}

```

```

quint64 random=qrand();
double percent=(100*random)/RAND_MAX;

int base=0;
for (int i=0;i<this->genOptions.count();i++)
{
    if (genOptions.at(i).port==port)
    {
        base+=this->genOptions.at(i).percent;
        if (percent<base)
        {
            p.destIP=genOptions.at(i).destIP;
            p.sourcePort=genOptions.at(i).port;
            p.sourceIP=this->getIP(genOptions.at(i).port);

            quint64 dif=genOptions.at(i).maxlen-genOptions.at(i).minlen;
            if (dif==0)
                p.length=genOptions.at(i).maxlen;
            else
            {
                random=qrand();
                double x=(dif*random)/RAND_MAX;
                p.length=genOptions.at(i).minlen+(quint32)x;
            }
        }
    }
}

return p;
}

```

```

Wire *DiagramItem::getWire(int port)
{
    QMap<int,Wire*>::const_iterator i=this->wires.find(port);
    if (i==this->wires.end())
        return 0;
    else
        return i.value();
}

```

```

void DiagramItem::Preprocess()
{
    if (this->myDiagramItemType==DiagramItem::Hub)
    {
        this->incommingBufferHub.append(this->incommingBufferHubP);
        this->incommingBufferHubP.clear();
    }

    if (this->myDiagramItemType==DiagramItem::Router)
    {
        for (int i=1;i<=this->portCounter;i++)
        {
            this->outgoingBuffer[i-1].append(this->outgoingBufferRouter[i-1]);
            this->outgoingBufferRouter[i-1].clear();
        }
    }
}

```

```

void DiagramItem::ProcessHub()
{
    if (this->incommingBufferHub.isEmpty()==true)
        return;

    // if (hubreceived>0)
    // {
    //     hubreceived--;
    //     if (hubreceived==0)
    //         return;
    // }

    Packet p=this->incommingBufferHub.takeFirst();

    QMapIterator<int,Wire*> i(this->wires);
    while (i.hasNext()) {
        i.next();
        if (i.key()!=(int)p.HubPort)
        {
            Wire *w=i.value();
            w->SendPacket(p,this);
            this->totalSent[0]+=p.length;
            prevActivePort|=1;
        }
    }
}

void DiagramItem::AddIncPack(Wire *w,Packet p)
{
    IP_descriptor d=getFromWire(w);
    if (this->myDiagramItemType==DiagramItem::Hub)
    {
        p.HubPort=d.port;
        //this->incommingBufferHub.append(p);
        if (this->incommingBufferHub.length()>=DiagramItem::MAX_BUFFER_LENGTH)
        {
            emit logMessage(QObject::tr("Device's' ") + this->itemTypeString + QObject::tr("buffer exceeded. It has lost 1
packet to ")
                +IPManager::parseIP(p.destIP)+QObject::tr("from ") +IPManager::parseIP(p.sourceIP));
            this->totalLost[0]+=p.length;
            return;
        }
        this->incommingBufferHub.append(p);
        this->totalReceived[0]+=p.length;
        prevActivePort|=1;
        //hubreceived++;
        return;
    }
    //this->AddOutgoingPacket(p,d.port);

    if (this->incommingBuffer[d.port-1].length()>=DiagramItem::MAX_BUFFER_LENGTH)
    {
        emit logMessage(QObject::tr("Device's' ") + this->itemTypeString + QObject::tr("buffer exceeded. It has lost 1 packet
to ")
            +IPManager::parseIP(p.destIP)+QObject::tr("from ") +IPManager::parseIP(p.sourceIP));
        this->totalLost[d.port-1]+=p.length;
        return;
    }
}

```

```

}

this->incommingBuffer[d.port-1].append(p);

switch (d.port)
{
case 1:prevActivePort|=0x1;break;
case 2:prevActivePort|=0x2;break;
case 3:prevActivePort|=0x4; break;
case 4:prevActivePort|=0x8; break;
}

}

void DiagramItem::Update()
{
if (this->myDiagramItemType==DiagramItem::Hub)
{
if (((this->incommingBufferHub.isEmpty()==false)||((this->incommingBufferHubP.isEmpty()==false)))
{
QPixmap* pixmap=this->buildPixmap(true,0);
setPixmap(*pixmap);
delete pixmap;
return;
}

return;
}

quint8 portset=0;
for (int i=1;i<=this->portCounter;i++)
{
if (((this->outgoingBuffer[i-1].isEmpty()==false)||((this->incommingBuffer[i-1].isEmpty()==false)))
{
switch (i)
{
case 1:portset|=0x1;break;
case 2:portset|=0x2;break;
case 3:portset|=0x4; break;
case 4:portset|=0x8; break;
}

//      for (int j=0;j<incommingBuffer2[i-1].count();j++)
//      {
//
//      if (incommingBuffer[i-1].count()>=this->MAX_BUFFER_LENGTH)
//      {
//          Packet p=incommingBuffer2[i-1].takeFirst();
//          emit logMessage(QObject::tr("Device's ") +this->itemTypeString+ QObject::tr("buffer exceeded. It has
lost 1 packet to ")
//              +IPManager::parseIP(p.destIP)+QObject::tr("from ") +IPManager::parseIP(p.sourceIP));
//          }
//      else
//          incommingBuffer[i-1].append(incommingBuffer2[i-1].takeFirst());
//      }

}

}
}

```

```

portset|=prevActivePort;
prevActivePort=0;
if (portset!=0)
{
    QPixmap* pixmap=this->buildPixmap(true,portset);
    setPixmap(*pixmap);
    delete pixmap;
}
else
{
    QPixmap* pixmap=this->buildPixmap(false,portset);
    setPixmap(*pixmap);
    delete pixmap;
}
}

void DiagramItem::Clear()
{
    this->incommingBufferHub.clear();
    this->incommingBufferHubP.clear();
    for (int i=0;i<4;i++)
    {
        this->incommingBuffer[i].clear();
        this->outgoingBuffer[i].clear();
        this->outgoingBufferRouter[i].clear();
    }
    this->prevActivePort=0;

    this->Update();

    for (int i=0;i<4;i++)
    {
        totalReceived[i]=0;
        totalSent[i]=0;
        totalLost[i]=0;
        totalReceivedWrong[i]=0;
    }
}

void DiagramItem::makeActive(Wire* w)
{
    quint8 portset=0;
    if (this->myDiagramItemType==DiagramItem::Hub)
    {
        portset=1;
    }
    IP_descriptor d= getFromWire(w);
    switch (d.port)
    {
        case 1:portset|=0x1;break;
        case 2:portset|=0x2;break;
        case 3:portset|=0x4; break;
        case 4:portset|=0x8; break;
    }

    QPixmap* pixmap=this->buildPixmap(true,portset);
    setPixmap(*pixmap);
    delete pixmap;
}

QString DiagramItem::buildStat()

```

```

{
    QString s="";
    s+="\n"+this->getItemStr();
    if (this->myDiagramItemType==DiagramItem::Hub)
    {
        s=s+QObject::tr("    \nTotal bytes received: %1").arg(this->totalReceived[0]);
        s=s+QObject::tr("    \nTotal bytes sent: %1").arg(this->totalSent[0]);
        s=s+QObject::tr("    \nTotal bytes lost: %1").arg(this->totalLost[0]);
    }
    else
    {
        for (int i=0;i<this->portCounter;i++)
        {
            s=s+QObject::tr("    \nPort %1:").arg(i+1);
            s=s+QObject::tr("        \nTotal bytes received: %1").arg(this->totalReceived[i]);
            s=s+QObject::tr("        \nTotal bytes sent: %1").arg(this->totalSent[i]);
            s=s+QObject::tr("        \nTotal bytes lost: %1").arg(this->totalLost[i]);
            s=s+QObject::tr("        \nTotal wrong IP bytes received: %1").arg(this->totalReceivedWrong[i]);
        }
    }
    return s;
}

```

Лістинг 4. Файл ipmanager.cpp

```
#include "ipmanager.h"
#include "diagramscene.h"
#include "general.h"

IPManager::IPManager(DiagramScene *s)
{
    this->scene=s;
}

void IPManager::AssignSN()
{
    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;
    //clearing all IPs
    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if (ditem)
            ditem->ClearIP();
    }
    scene->ClearSubNets();

    //assigning new IPs
    int subnetCnt=1;
    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if (ditem)
        {
            if (ditem->diagramItemType()!=DiagramItem::Hub)
            {
                quint8 portCounter= ditem->getPortCounter();
                for (int i=1;i<=portCounter;i++)
                {
                    if (ditem->getSubnet(i)==0)
                    {
                        QList <IP_descriptor> items= getSubNet(i,ditem);
                        //IP_descriptor ditem;
                        for (int j = 0; j < items.size(); ++j)
                        {
                            items.at(j).d->setSubnet(items.at(j).port,subnetCnt);
                        }
                        if (items.isEmpty()==false)subnetCnt++;
                    }
                }
            }
        }
    }
}

quint32 IPManager::parseIP(QString s)
{
    quint32 ip=0;
    QString str=s,buf;
    int digit;

    int ind1=0;
    for (int i=3;i>=0;i--)
    {
```



```

        if (i!=0)
            ind1=str.indexOf(".");
        else
            ind1=str.length();

        buf=str.left(ind1);
        digit=buf.toInt();
        if (digit>255) return 0;
        ip=ip|(quint32)digit<<i*8;
        str=str.right(str.length()-ind1);
        if (i!=0)
            str=str.right(str.length()-1);
    }
    return ip;
}

QString IPManager::parseIP(quint32 ip)
{
    QString str,s;
    for (int i=3;i>=0;i--)
    {
        quint8 currDigit=(ip>>8*i) &0xFF;
        s.setNum(currDigit);
        str=str+s;
        if (i!=0)
            str=str+".";
    }
    return str;
}

QList <IP_descriptor> IPManager::getSubNet(int port,DiagramItem *d )
{
    QList <IP_descriptor> lst;
    QList <IP_descriptor> lstz;
    IP_descriptor dsc;
    dsc.d=d;
    dsc.port=port;
    lstz.append(dsc);
    lst.append(dsc);

    IP_descriptor ditem=d->getConnectedItem(port);
    if (ditem.d)
    {
        if (ditem.d->diagramItemType()!=DiagramItem::Hub)
        {
            lstz.append(ditem);
        }
        else
        {
            QList <IP_descriptor> hublst =getSubNetFromHub(ditem,dsc);
            lstz.append(hublst);
        }
    }
    return lstz;
}

QList <IP_descriptor> IPManager::getSubNetFromHub(IP_descriptor hub,IP_descriptor prev)
{
    QList <IP_descriptor> lst;
    if (hub.d->diagramItemType()!=DiagramItem::Hub)
        return lst;

```

```
QList <IP_descriptor> lsth=hub.d->getConnectedDevices_HUB();
```

```
mark:
IP_descriptor ditem;
for (int i = 0; i < lsth.size(); ++i)
{
    ditem=lsth.at(i);
    if ((ditem.d==prev.d)&&(ditem.port==prev.port))
    {
        lsth.removeAt(i);
        goto mark;
    }
    else
    {
        if (ditem.d->diagramItemType()==DiagramItem::Hub)
        {
            if ((ditem.d==hub.d)|| (ditem.d==prev.d))
            {
                lsth.removeAt(i);
                goto mark;
            }
            lsth.append(getSubNetFromHub(ditem,hub));
            lsth.removeAt(i);
            goto mark;
        }
    }
}

return lsth;
}
```

```
bool IPManager::AssignIP(int subnet,quint32 ip,quint8 mask)
```

```
{
    QList<IP_descriptor> lst= getAllSNDev(subnet);

    quint32 bigmask=parseMask(mask);

    quint32 ipstart=(ip&bigmask)+1;

    quint32 ipend=(ip&(~bigmask))-1;

    quint32 counter=ipend-ipstart;

    if (counter<(quint32)lst.count())
        return false;

    for (int i=0;i<lst.count();i++)
    {
        IP_struct s;
        s.ip=ip+i+1;
        s.mask=mask;
        lst.at(i).d->setIP(s,lst.at(i).port);
    }

    SN_descriptor s;
    s.ip=ip;
    s.mask=mask;
    s.subnet=subnet;
    this->scene->setSNDestructor(s);
    return true;
}
```

```

}

quint32 IPManager::parseMask(int mask)
{
    quint32 maskToRet=0;
    for (int i=mask;i>0;i--)
    {
        maskToRet>>=1;
        maskToRet|=0x80000000;
    }
    return maskToRet;
}

QList<IP_descriptor> IPManager::getAllSNDev(int subnet)
{
    QList<IP_descriptor> lst;

    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;

    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if ((ditem)&&(ditem->diagramItemType()!=DiagramItem::Hub))
        {
            int portCnt=ditem->getPortCounter();
            for (int i=1;i<=portCnt;i++)
            {
                if (ditem->getSubnet(i)==subnet)
                {
                    IP_descriptor d;
                    d.d=ditem;
                    d.port=i;
                    lst.append(d);
                }
            }
        }
    }
    return lst;
}

QList<int> IPManager::GetAllSubnets()
{
    QList<int> subnets;
    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;

    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if ((ditem)&&(ditem->diagramItemType()!=DiagramItem::Hub))
        {
            int portCnt=ditem->getPortCounter();
            for (int i=1;i<=portCnt;i++)
            {
                int s=ditem->getSubnet(i);
                if (s==0) continue;
                if (subnets.contains(s)==false)

```

```

        subnets.append(s);
    }
}
return subnets;
}

```

```

QList<quint32> IPManager::GetAllIPs()
{
    QList<quint32> listToReturn;

    QList<QGraphicsItem *> items=this->scene->items();
    QGraphicsItem * tmp;
    DiagramItem* ditem;

    foreach (tmp,items)
    {
        ditem = qgraphicsitem_cast<DiagramItem *>(tmp);
        if ((ditem)&&(ditem->diagramItemType()!=DiagramItem::Hub))
        {
            int portCnt=ditem->getPortCounter();
            for (int i=1;i<=portCnt;i++)
            {
                quint32 ip=ditem->getIP(i);
                if (ip)
                    listToReturn.append(ip);
            }
        }
    }
    return listToReturn;
}

```

Лістинг 5. Файл *Wire.cpp*

```
#include "wire.h"
#include "diagramitem.h"
#include "diagramscene.h"
#include "general.h"
#include <QtGui>

Wire::Wire(DiagramScene *s, DiagramItem *src, DiagramItem *dst)
{
    this->connectedItems[0]=src;
    this->connectedItems[1]=dst;
    setFlag(QGraphicsItem::ItemIsSelectable, true);
    myColor=Qt::black;
    setPen(QPen(myColor, 2, Qt::SolidLine, Qt::RoundCap, Qt::RoundJoin));
    scene=s;
    P0.length=0;
    P1.length=0;
    collision=false;

    totalSent1=0;
    totalSent2=0;
    totalCollisions=0;
}

Wire::~Wire()
{
    this->removeMyself();
    disconnect(this, 0, 0, 0);
}

void Wire::removeMyself()
{
    connectedItems[0]->removeWire(this);
    connectedItems[1]->removeWire(this);
}

void Wire::removeConnectedItems()
{
    this->connectedItems[1]->removeWire(this);
    this->connectedItems[0]->removeWire(this);
}

QRectF Wire::boundingRect() const
{
    qreal extra = (pen().width() + 20) / 2.0;

    return QRectF(line().p1(), QSizeF(line().p2().x() - line().p1().x(),
        line().p2().y() - line().p1().y()))
        .normalized()
        .adjusted(-extra, -extra, extra, extra);
}
//-----
QPainterPath Wire::shape() const
{
    QPainterPath path = QGraphicsLineItem::shape();

    return path;
}
//-----
```

```

void Wire::updatePosition()
{
    QLineF line(mapFromItem(connectedItems[0], 0, 0), mapFromItem(connectedItems[1], 0, 0));
    setLine(line);
}
//-----
void Wire::paint(QPainter *painter, const QStyleOptionGraphicsItem *,
                QWidget *)
{
    if (connectedItems[0]->collidesWithItem(connectedItems[1]))
        return;

    QPen myPen = pen();
    myPen.setColor(myColor);
    painter->setPen(myPen);
    painter->setBrush(myColor);
    QLine l(connectedItems[0]->getWirePos(this), connectedItems[1]->getWirePos(this));
    setLine (l);
    painter->drawLine(line());
    if (isSelected()) {
        painter->setPen(QPen(myColor, 1, Qt::DashLine));
        QLineF myLine = line();
        myLine.translate(0, 4.0);
        painter->drawLine(myLine);
        myLine.translate(0, -8.0);
        painter->drawLine(myLine);
    }
}

IP_descriptor Wire::getConnectedItem(DiagramItem *d)
{
    if (this->connectedItems[0]==d)
    {
        return connectedItems[1]->getFromWire(this);
    }

    if (this->connectedItems[1]==d)
    {
        return connectedItems[0]->getFromWire(this);
    }

    IP_descriptor ds;
    ds.d=0;
    ds.port=0;

    return ds;
}

void Wire::SendPacket(Packet pkt, DiagramItem *d)
{
    if (this->connectedItems[0]==d)
    {
        if (P1.length==0)
            P0=pkt;
        else
        {
            emit logMessage(QObject::tr("Collision between ") + connectedItems[0]->getItemStr() + " and
"+connectedItems[1]->getItemStr());
            //      P0.length=0;
            //      P1.length=0;
            collision=true;

```

```

        this->totalCollisions++;
    }
}

if (this->connectedItems[1]==d)
{
    if (P0.length==0)
        P1=pkt;
    else
    {
        emit logMessage(QObject::tr("Collision between ") + connectedItems[0]->getItemStr() + " and
"+connectedItems[1]->getItemStr());
//        P0.length=0;
//        P1.length=0;
        collision=true;

        this->totalCollisions++;
    }
}
}

```

```

void Wire::BeginWork()
{
    if (P0.length!=0)
    {
        connectedItems[1]->AddIncPack(this,P0);
        this->totalSent1+=P0.length;
        P0.length=0;
    }

    if (P1.length!=0)
    {
        connectedItems[0]->AddIncPack(this,P1);
        this->totalSent2+=P1.length;
        P1.length=0;
    }
}

```

```

void Wire::Update()
{
    if ((P1.length!=0)||(P0.length!=0))
        this->setColor(Qt::red);
    else
        this->setColor(Qt::black);

    if (P1.length!=0)
    {
        connectedItems[1]->makeActive(this);
    }

    if (P0.length!=0)
    {
        connectedItems[0]->makeActive(this);
    }

    if (this->collision==true)
    {
        P0.length=0;
        P1.length=0;
    }
}

```

```

        this->setColor(Qt::green);
    }

    this->collision=false;

}

void Wire::Clear()
{
    this->P0.length=0;
    this->P1.length=0;
    collision=false;
    this->setColor(Qt::black);

    this->totalSent1=0;
    this->totalSent2=0;
    this->totalCollisions=0;

}

QString Wire::buildStat()
{
    QString s="";
    s=QObject::tr("\nWire ") + this->connectedItems[0]->getItemStr() + " - " + this->connectedItems[0]->getItemStr();
    s=s+QObject::tr("    \nTotal collision counter: %1").arg(this->totalCollisions);
    s=s+QObject::tr("    \nTotal sent->: %1").arg(this->totalSent1);
    s=s+QObject::tr("    \nTotal sent<-: %1").arg(this->totalSent2);
    return s;
}

```


Лістинг 6. Файл *Routertableform.cpp*

```
#include <QtGui>
#include "routertableform.h"
#include "ui_routertableform.h"
#include "diagramitem.h"
#include "diagramscene.h"
#include "general.h"
#include "ipmanager.h"

routerTableForm::routerTableForm(DiagramScene *s, DiagramItem *d, QWidget *parent) :
    QWidget(parent),
    ui(new Ui::routerTableForm)
{
    ui->setupUi(this);
    this->scene=s;
    this->ditem=d;

    QList<SN_descriptor> lst=scene->getSubnets();
    for (int i=0;i<lst.count();i++)
    {
        QString str=IPManager::parseIP(lst.at(i).ip);
        this->ui->comboBoxIPA->addItem(str);
    }

    int portCnt=ditem->getPortCounter();
    for (int i=1;i<=portCnt;i++)
    {
        QString str;
        str=str.setNum(i);
        this->ui->comboBoxPA->addItem(str);
        this->ui->comboBoxPM->addItem(str);
    }
}

routerTableForm::~routerTableForm()
{
    delete ui;
}

void routerTableForm::changeEvent(QEvent *e)
{
    QWidget::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}

void routerTableForm::FillData()
{
    this->ui->tableWidget->clearContents();

    QStringList labels;
    labels.clear();
    labels << tr("IP") << tr("Mask")<<tr("Gateway")<<tr("ID");
    this->ui->tableWidget->setColumnCount(labels.size());
}
```

```

this->ui->tableWidget->setHorizontalHeaderLabels(labels);
this->ui->tableWidget->setColumnWidth(0,130);
this->ui->tableWidget->setColumnWidth(1,150);
this->ui->tableWidget->setColumnWidth(2,150);
this->ui->tableWidget->setColumnWidth(routerTableForm::ColForID,50);

if ((this->ditem)&&(this->ditem->diagramItemType()==DiagramItem::Router))
{
    QList <RouterEntry> lst=ditem->getRouterTable();
    this->ui->tableWidget->setRowCount(lst.count());

    RouterEntry opt;
    int counter=0;
    foreach (opt,lst)
    {
        QString str;
        str=IPManager::parseIP(opt.ip);
        QTableWidgetItem *newItem = new QTableWidgetItem(str);
        this->ui->tableWidget->setItem(counter, 0, newItem);

        str.setNum(opt.mask);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 1, newItem);

        str=str.setNum(opt.port);
        str="Port "+str;
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, 2, newItem);

        str=str.setNum(opt.ID);
        newItem = new QTableWidgetItem( str);
        this->ui->tableWidget->setItem(counter, routerTableForm::ColForID, newItem);

        counter++;
    }
}

}

void routerTableForm::on_comboBoxIPA_currentIndexChanged(QString s)
{
    QList<SN_descriptor> lst=scene->getSubnets();
    for (int i=0;i<lst.count();i++)
    {
        QString str=IPManager::parseIP(lst.at(i).ip);

        if (s==str)
            this->ui->lineEditAM->setText(str.setNum(lst.at(i).mask));
    }
}

void routerTableForm::on_commandLinkButton_2_clicked()
{
    //automatic
    quint32 ip;
    QString ipst=this->ui->comboBoxIPA->currentText();
    ip=IPManager::parseIP(ipst);

    if (ip==0)
    {

```

```

    QMessageBox::critical(0, tr("Error"),tr("Select IP"));
    return;
}

QString mst=this->ui->lineEditAM->text();
quint8 mask=(quint8)mst.toInt();

quint8 port;
QString portst=this->ui->comboBoxPA->currentText();
port=(quint8)portst.toInt();

QList<RouterEntry> optLst=this->ditem->getRouterTable();
quint32 ID;
if (optLst.isEmpty())
{
    ID=0;
}
else
{
    QList<int> intList;
    for (int i=0;i<optLst.count();i++)
    {
        intList.append(optLst.at(i).ID);
    }
    qSort(intList.begin(), intList.end());

    int prefID=0;
    int num;
    foreach(num,intList)
    {
        if (prefID!=num)
        {
            ID=prefID;
            break;
        }
        prefID++;
    }
    ID=prefID;
}

if (this->isIPFree(ip,mask)==false)
{
    QMessageBox::critical(0, tr("Error"),tr("Such entry exists (check bitwise and of mask and ip)"));
    return;
}

RouterEntry r;
r.ip=ip;
r.mask=mask;
r.port=port;
r.ID=ID;

this->ditem->addRouterEntry(r);
this->FillData();
}

void routerTableForm::on_commandLinkButton_clicked()
{
    //manual

```

```

quint32 ip;
QString ipst=this->ui->lineEdit->text();
ip=IPManager::parseIP(ipst);

if (ip==0)
{
    QMessageBox::critical(0, tr("Error"),tr("Select IP"));
    return;
}

quint8 mask=(quint8)this->ui->spinBox->value();

quint8 port;
QString portst=this->ui->comboBoxPM->currentText();
port=(quint8)portst.toInt();


QList<RouterEntry> optLst=this->ditem->getRouterTable();
quint32 ID;
if (optLst.isEmpty())
{
    ID=0;
}
else
{
    QList<int> intList;
    for (int i=0;i<optLst.count();i++)
    {
        intList.append(optLst.at(i).ID);
    }
    qSort(intList.begin(), intList.end());

    int prefID=0;
    int num;
    foreach(num,intList)
    {
        if (prefID!=num)
        {
            ID=prefID;
            break;
        }
        prefID++;
    }
    ID=prefID;
}

if (this->isIPFree(ip,mask)==false)
{
    QMessageBox::critical(0, tr("Error"),tr("Such entry exists (check bitwise and of mask and ip)"));
    return;
}
RouterEntry r;
r.ip=ip;
r.mask=mask;
r.port=port;
r.ID=ID;

this->ditem->addRouterEntry(r);
this->FillData();
}

```

```

void routerTableForm::on_pushButton_clicked()
{
    int selectedRow=this->ui->tableWidget->currentRow();
    if (selectedRow==-1)
    {
        QMessageBox::critical(0, tr("Error"),tr("No row selected."));
        return;
    }

    QTableWidgetItem * itm = this->ui->tableWidget->item(selectedRow,routerTableForm::ColForID);
    QString str=itm->text();
    quint32 ID=str.toInt();

    QList<RouterEntry> optLst=this->ditem->getRouterTable();

    for (int i=0;i<optLst.count();i++)
    {
        if (optLst.at(i).ID==ID)
        {
            this->ditem->removeRouterEntry(ID);
            break;
        }
    }

    this->FillData();
}

bool routerTableForm::isIPFree(quint32 ip,quint8 mask)
{
    quint32 maskbig=IPManager::parseMask(mask);
    quint32 ipSN=ip&maskbig;

    QList<RouterEntry> optLst=this->ditem->getRouterTable();
    quint32 ipTmp;
    quint32 maskTmp;
    for (int i=0;i<optLst.count();i++)
    {
        ipTmp=optLst.at(i).ip;
        maskTmp=IPManager::parseMask(optLst.at(i).mask);
        ipTmp&=maskTmp;
        if (ipTmp==ipSN)
            return false;
    }
    return true;
}

```

ДОДАТОК Б

Блок схеми та діаграми застосованих алгоритмів

