

ВІДЗИВ НАУКОВОГО КЕРІВНИКА НА ВИПУСКНУ КВАЛІФІКАЦІЙНУ РОБОТУ

Студента (ки) **Кривохижі В.О.**

(прізвище та ініціали)

представлену на здобуття освітнього ступеня «бакалавр»
зі спеціальності 123 «Комп'ютерна інженерія»

1. Робота виконана на тему Симулятор комп'ютерних мереж та основних мережевих протоколів

2. Актуальність. Актуальність обраної теми полягає в тому, що симулятори мереж широко використовуються при вирішенні дуже різноманітних задач – як навчальних при дослідженні роботи протоколів, так і робочих при проектуванні справжньої мережі..

(вказати, які питання теоретичного і практичного характеру визначають актуальність роботи)

3. Основний зміст роботи викладено на 48 сторінках тексту, містить 25 ілюстрацій. Список використаних літературних джерел налічує 9 найменувань.

4. Зміст розділів роботи (вказати за кожним розділом, які питання досліджені та з якою метою)

Розділ 1. Аналіз напрямків використання інформаційних технологій для симуляції роботи мережі. У даному розділі здійснено постановку задачі, проведено аналіз аналогів програмного продукту, обґрунтовано вибір інструментальних засобів, описано властивості комп'ютерних мереж, пристроїв на протоколів.

Розділ 2. Проектування системи симулятора комп'ютерних мереж. У розділі визначено варіанти використання та структури системи, здійснено проектування та реалізацію алгоритмів роботи систем, міждопоміжкової комунікації.

Розділ 3. Інтерфейс та порядок роботи з симулятором комп'ютерних мереж. Описано порядок встановлення та налаштування системи, описано структуру інтерфейсу та порядок роботи з системою, досліджено роботу системи.

5. Висновки обґрунтовані, повністю відповідають поставленим завданням, мають теоретичне та практичне значення. Практичне значення отриманих результатів полягає у розробці програмного додатку для проектування та симуляції комп'ютерних мереж та протоколів.

(обґрунтованість, теоретичне і практичне значення)

6. Список використаних літературних джерел. Джерела опрацьовані у повному обсязі, відповідають змісту роботи.

(обсяг і повнота опрацювання за змістом роботи)

7. Додатки _____

(перелік додатків та повнота їх опрацювання за змістом роботи)

8. Таблиці та ілюстрації Ілюстрації носять інформаційний характер, є доповненням до змісту роботи.

(призначення таблиць та їх використання для підтвердження досліджень в роботі)

9. Якість оформлення роботи на достатньому рівні

10. Ставлення студента (ки) до виконання випускної кваліфікаційної роботи відповідально

11. Висновок

В цілому при написанні випускної кваліфікаційної роботи студент (ка) **Кривохижа В.О.**

(прізвище та ініціали)

вміння /невміння/ підібрати необхідну літературу, інформаційну базу, проаналізувати її, зробити відповідні висновки і оформити свої дослідження у випускній кваліфікаційній роботі.

Випускна кваліфікаційна робота студента (ки)

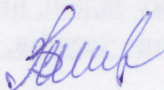
Кривохижі В.О.

(прізвище та ініціали)

написана на достатньо /недостатньо/ високому рівні та може /не може/ бути допущена до захисту в Екзаменаційній комісії.

Науковий керівник д. п.н., доцент, професор кафедри інженерії програмного забезпечення
(вчений ступінь, звання, посада)

(підпис)



Вакалюк Т.А.

(прізвище та ініціали)

«19» 06 2020 р.

РЕЦЕНЗІЯ
на кваліфікаційну роботу
студента 4 курсу групи КІ-2 спеціальності 123 «Комп'ютерна інженерія»
факультету інформаційно-комп'ютерних технологій
Державного університету «Житомирська політехніка»
Кривохижі В'ячеслава Олександровича
на тему: „Симулятор комп'ютерних мереж та основних мережевих
протоколів”

Кваліфікаційну роботу студента Кривохижі В. О. “Симулятор комп'ютерних мереж та основних мережевих протоколів” присвячено актуальній тематиці – розроблення програмного забезпечення для проектування та симуляції комп'ютерних мереж та протоколів.

У роботі описано проектування програмного комплексу системи симуляції комп'ютерних мереж, його інтерфейс та порядок роботи з ним.

До складу пояснювальної записки входить вступ, три розділи основного тексту, висновки, список використаної літератури та додатки.

У першому розділі пояснювальної записки описано основну функціональність програмного комплексу, визначено основні етапи реалізації поставленого завдання, проаналізовано аналогічні програмні засоби (Cisco Packet Tracer, GNS3), описано обрані для реалізації мову програмування (C#) та технології (WPF, WCF), розглянуто основні властивості комп'ютерних мереж та мережевих пристроїв, відтворюваних у пропонованому програмному засобі.

У другому розділі пояснювальної записки виконано проектування програмного комплексу системи симуляції комп'ютерних мереж засобами мови UML, описано процес комунікації між додатками на базі WCF.

У третьому розділі пояснювальної записки розглянуто порядок встановлення та налаштування програмного комплексу системи симуляції комп'ютерних мереж, вказано мінімальні системні вимоги, продемонстровано роботу системи на прикладі (створення пристроїв, з'єднання пристроїв в мережу, налаштування пристроїв, пересилання даних між пристроями, підключення одного додатку до іншого).

Зміст роботи відповідає поставленому завданню. У цілому кваліфікаційна робота виконана на належному рівні. Тему роботи розкрито досить повно. Студентом продемонстровано ґрунтовні теоретичні знання та практичні фахові навички. Пояснювальну записку сформовано загалом логічно, науково й технічно грамотно. Функціонал програмного комплексу відповідає заявленому. Позитивний бік роботи полягає у тому, що одержані результати може бути застосовано для проектування мереж та відстежування результату їх роботи. Водночас, робота має окремі недоліки, серед яких: недостатньо чітке виокремлення основних переваг розробленого програмного комплексу, деякі відхилення від вимог щодо оформлення пояснювальної записки (зокрема підзаголовки та нумерація рисунків), мовні та стилістичні неточності. Вказані недоліки у цілому не знижують якості роботи.

У цілому кваліфікаційна робота студента Кривохижі В. О. „Симулятор комп'ютерних мереж та основних мережевих протоколів” відповідає вимогам рівня вищої освіти “бакалавр” спеціальності 123 “Комп'ютерна інженерія” і рекомендується до захисту в ЕК.

Рецензент,
доцент кафедри
КІ та КБ,
к.пед.н.
19 червня 2020 р.



О. С. Головня

Міністерство освіти і науки України
Державний університет «Житомирська політехніка»
Факультет інформаційно-комп'ютерних технологій
Кафедра комп'ютерної інженерії та кібербезпеки

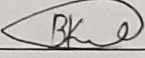
ПОЯСНЮВАЛЬНА ЗАПИСКА

до випускної кваліфікаційної роботи бакалавра

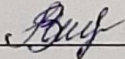
на тему:

на тему: «Проект програми-симулятора комп'ютерних мереж та основних
мережевих протоколів»

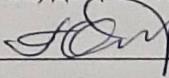
Виконав студент 4-го курсу, групи КІ-2
спеціальності 123 «Комп'ютерна інженерія»

 Кривохижа В.О.

Керівник, професор кафедри ІПЗ, д.п.н., доц.

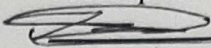
 Вакалюк Т.А.

Рецензент, доцент кафедри КІ та КБ, канд. пед. наук.

 Головня О.С.

ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА КІБЕРБЕЗПЕКИ
СПЕЦІАЛЬНІСТЬ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

ЗАТВЕРДЖУЮ
Завідувач кафедри
комп'ютерної інженерії
та кібербезпеки

 Єфіменко А. А.
«06» квітня 2020 р.

ЗАВДАННЯ
на випускню кваліфікаційну роботу

Студента **Кривохижі В'ячеслава Олександровича**

Тема роботи: **Проект програми-симулятора комп'ютерних мереж та основних мережевих протоколів.**

затверджена Наказом по університету від «06» квітня 2020 р. № 98с

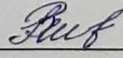
Термін здачі студентом закінченої роботи – **1 червня 2020 р.**

Вихідні дані для роботи:

1. Технічне завдання
2. Аналіз аналогічних розробок
3. Алгоритми роботи програми
4. Опис роботи програми

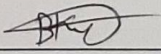
Консультанти з випускної кваліфікаційної роботи із зазначенням розділів, що їх стосуються

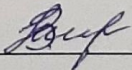
Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
1,2,3	Проф. кафедри ІПЗ, д.п.н. Вакалюк Т. А.	06.04.2020 р.	06.04.2020 р.
2	Проф. кафедри ІПЗ, д.п.н. Вакалюк Т. А.	27.04.2020 р.	27.04.2020 р.
3	Проф. кафедри ІПЗ, д.п.н. Вакалюк Т. А.	18.05.2020 р.	18.05.2020 р.

Керівник  Вакалюк Т.А.

Календарний план

№ з/п	Назва етапів випускної кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1	Постановка задачі	1 квітня 2020 – 3 квітня 2020	Виконано
2	Пошук, огляд та аналіз аналогічних розробок	4 квітня 2020 – 6 квітня 2020	Виконано
3	Формулювання технічного завдання	7 квітня 2020 – 8 квітня 2020	Виконано
4	Опрацювання літературних джерел	9 квітня 2020 – 26 квітня 2020	Виконано
5	Проектування структури	27 квітня 2020 – 3 травня 2020	Виконано
6	Написання програмного коду	4 травня 2020 – 27 травня 2020	Виконано
7	Тестування	28 травня 2020 – 31 травня 2020	Виконано
8	Написання пояснювальної записки	1 червня 2020 – 18 червня 2020	Виконано
9	Захист	21 червня 2020	Виконано

Студент  Кривохижа В.О.

Керівник  Вакалюк Т.А.

РЕФЕРАТ

Випускна робота бакалавра складається з програмного комплексу системи симуляції комп'ютерних мереж та пояснювальної записки. Пояснювальна записка до випускної роботи містить 48 сторінок, 25 ілюстрацій та 9 літературних джерел.

Метою випускної роботи є розробка програмного додатку для проектування та симуляції комп'ютерних мереж та протоколів.

В роботі визначені головні завдання для розробки системи, проаналізовано методи вирішення поставленої задачі та проаналізовано аналоги. Обґрунтовано вибір технологій для розробки програмного додатку. Наведено загальну структуру програмного коду, описані основні алгоритми та внутрішні процеси. Описано методологію користування додатком та сценарії роботи програмного комплексу.

КЛЮЧОВІ СЛОВА: КОМП'ЮТЕРНА МЕРЕЖА, ПРОГРАМНИЙ ДОДАТОК, МЕРЕЖЕВИЙ ПРИСТРІЙ, ПРОТОКОЛ, КОРИСТУВАЦЬКИЙ ІНТЕРФЕЙС.

Державний університет «Житомирська
політехніка». 20.123.14.000 - ПЗ

Зм.	Арк.	№ докум.	Підпис	Дата			
Розроб.		Кривохижа В. О.		19.06.20	Проект програмно-симулятора комп'ютерних мереж та основних мережевих протоколів Пояснювальна записка		
Керівник		Вакалюк Т. А.		19.06.20			
Н. контроль		Ефіменко А. А.		19.06.20			
Рецензент		Головня О. С.		19.06.20			
Зав. каф.		Сфіменко А. А.		19.06.20			
					Літ.	Арк.	Аркушів
						4	86
					ФІКТ, гр. КІ-2		

ABSTRACT

Bachelor's graduate work consists of a software package of a computer network simulation system and documentation. Documentation for bachelor's graduate work contains 48 pages, 25 illustrations and 9 references.

The goal of the graduate work is to develop a software application for the design and simulation of computer networks and protocols.

The main tasks for system development are defined in the work, the methods of solving the set task are analyzed and analogues are analyzed. The choice of technologies for software application development is substantiated. The general structure of the program code is given, the basic algorithms and internal processes are described. The methodology of using the application and the scenarios of the software package are described.

KEY WORDS: COMPUTER NETWORK, SOFTWARE APPLICATION, NETWORK DEVICE, PROTOCOL, USER INTERFACE.

ЗМІСТ

РЕФЕРАТ.....	4
ABSTRACT	5
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СИМУЛЯЦІЇ РОБОТИ МЕРЕЖІ	10
1.1. Постановка задачі	10
1.2. Аналіз аналогів програмного продукту.....	12
1.3. Обґрунтування вибору інструментальних засобів	16
1.4. Опис властивостей комп'ютерних мереж, пристроїв на протоколів.....	18
Висновки до розділу 1.....	23
РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ СИМУЛЯТОРА КОМП'ЮТЕРНИХ МЕРЕЖ.....	25
2.1. Визначення варіантів використання та структури системи.....	25
2.2. Проектування та реалізація алгоритмів роботи системи	33
2.3. Проектування та реалізація міжодаткової комунікації	36
Висновки до розділу 2.....	37
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИМУЛЯТОРОМ КОМП'ЮТЕРНИХ МЕРЕЖ.....	38
3.1. Порядок встановлення та налаштування системи.	38
3.2. Структура інтерфейсу та порядок роботи з системою.....	39
3.3. Дослідження роботи системи.....	43
Висновки до розділу 3.....	45
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	47
ДОДАТКИ.....	48

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

OSI – Open Systems Interconnection
IIS – Internet Information Services
WAS – Windows Activation Service
WCF – Windows Communication Foundation
WPF – Windows Presentation Foundation
XAML – Extensible Application Markup Language
XML – Extensible Markup Language
IDE – IntegratedDevelopment Environment
UI – User Interface
PT – Packet Tracer
UML – Unified Modeling Language
MVVM – Model-View-ViewModel
NIC – Network Interface Controller
FDDI – Fiber Distributed Data Interface
WLAN – Wireless Local Area Network
IP – Internet Protocol
ICMP – Internet Control Message Protocol
ARP – Address Resolution Protocol
TCP – Transmission Control Protocol
UDP – User Datagram Protocol
DNS – Domain Name System
HTTP – HyperText Transfer Protocol
FTP – File Transfer Protocol
SMTP – Simple Mail Transfer Protocol
DHCP – Dynamic Host Configuration Protocol

ВСТУП

Актуальність теми. Ми живемо в столітті інформаційно-комунікаційних систем та технологій, де неможливо уявити жодну організацію без сучасної обчислювальної техніки. Сьогодні на одному підприємстві можуть знаходитись близько ста, а то і більше комп'ютерів. Отже, вся сукупність обладнання, що поєднана лініями зв'язку та обмінюється даними між собою відповідно до певних правил являє собою локальну обчислювальну мережу.

В даний час комп'ютерні мережі характеризуються складністю топології, тому часто постає питання попереднього моделювання комп'ютерних мереж. За допомогою моделювання визначають оптимальну топологію майбутньої мережі, необхідне мережеве обладнання, а також можливість майбутнього розвитку. Крім того, моделювання комп'ютерної мережі дозволяє уникнути витрат, що виникають у результаті реальної побудови мережі в майбутньому.

Також, для правильної взаємодії комп'ютерів, що працюють в мережах різноманітної структури та використовують різне програмне забезпечення, необхідна наявність певних стандартів. Цих стандартів на даний момент існує також досить велика кількість. Стандарти і протоколи строго визначають норми і правила технічної організації комп'ютерних мереж і програм, що реалізують взаємодію по мережі.

Оскільки питома кількість персональних комп'ютерів, об'єднаних в мережі, неухильно зростає, розгляд таких проблем, як мережеві протоколи і стандарти, набувають особливої актуальності. Важливу роль дана проблема має і в аспекті вибору того чи іншого способу побудови комп'ютерної мережі, що відповідає заданому набору вимог.

Для країни, що розвивається, це підкреслює виклик для навчання та для закладів освіти, оскільки попит на IT-фахівців продовжує зростати. Розуміння того, як комп'ютерні мережі проектуються, як вони функціонують і як ними можна керувати є необхідною навичкою, яку IT-спеціаліст повинен придбати.

«Як правило, більшість компаній наймають сертифікованого професіонала для нагляду за командою інженерів, які не сертифіковані та вивчають основні

процедури в рамках навчання на роботі. Хоча практичний досвід, безперечно, цінний у вирішенні питань, структурована та формальна підготовка може принести більше користі для компанії та дозволяє персоналу набути інших важливих та нових навичок" [2].

Актуальність обраної теми полягає в тому, що симулятори мереж широко використовуються при вирішенні дуже різноманітних задач – як навчальних при дослідженні роботи протоколів, так і робочих при проектуванні справжньої мережі.

Метою випускної роботи є розробка програмного додатку для проектування та симуляції комп'ютерних мереж та протоколів.

Встановлена мета обумовлює виконання наступних завдань:

- вивчити основні принципи роботи комп'ютерних мереж;
- визначити алгоритм та архітектуру для реалізації програмного додатку;
- обґрунтувати та провести вибір засобів реалізації програмного додатку;
- реалізувати програмний додаток;
- оптимізувати реалізований програмний додаток завдяки застосуванню адаптивних алгоритмів та експериментів з параметрами;
- провести тестування програмного додатку, аналіз результатів у вигляді остаточних висновків.

Об'єктом дослідження є симулятор комп'ютерних мереж та протоколів.

Предметом дослідження є засоби розробки симулятора комп'ютерних мереж та протоколів.

Розроблена система може бути використана для проектування мереж та відстежування результату їх роботи.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СИМУЛЯЦІЇ РОБОТИ МЕРЕЖІ

1.1. Постановка задачі

Головною метою даної роботи є розробка програмного додатку для проектування та симуляції комп'ютерних мереж та протоколів. Внаслідок чого одним із важливих завдань є ознайомлення з теорією комп'ютерних мереж. Закріплення знань у процесі дослідження тематики комп'ютерних мереж відбувається шляхом реалізації архітектури, алгоритмів, пошуків способів оптимізації, і в результаті - розробка мережевого симулятора, який би дозволив користувачу створювати комп'ютерні мережі та відстежувати процеси, які відбуваються в них.

Симулятор комп'ютерних мереж повинен надавати можливість додавання пристроїв до мережі. Кожен пристрій може мати мережеві карти, які містять порти для різних каналів зв'язку. Кожен пристрій має параметри, такі як: назва, IP-адреса та інші. Пристрої повинні з'єднуватись за допомогою каналів зв'язку. Пристрої можуть спілкуватись один з одним через канали зв'язку.

В симуляторі комп'ютерних мереж повинна бути реалізована функціональність, яка б дозволила відстежувати процеси спілкування пристроїв між собою.

Головна функція симулятора комп'ютерних мереж полягає в тому, щоб дати користувачам можливість підключатись до мереж один одного. Користувач може відкрити доступ до своєї мережі з можливістю для перегляду або редагування в реальному часі.

Підсумовуючи вищесказане, відзначимо основні етапи для реалізації поставленого завдання:

1. Постановка задачі

1.1. Визначення основної функціональності симулятора комп'ютерних мереж.

1.2. Вибір архітектури симулятора.

1.3. Підбір оптимальних теоретичних відомостей про мережеві властивості для найбільшого наближення симуляції до реальних умов.

2. Аналіз вимог

2.1. Користувач може додавати до мережі пристрої.

2.2. Пристрої можна з'єднувати між собою за допомогою каналів зв'язку.

2.3. Пристрої можуть спілкуватись один з одним за допомогою спеціальної функціональності для створення трафіку.

2.4. Необхідно дати можливість для відстежування процесу спілкування пристроїв між собою.

2.5. Користувач може відкривати доступ до своєї мережі для перегляду або редагування в реальному часі.

3. Планування

3.1. Проаналізувати способи програмної реалізації симулятора комп'ютерних мереж.

3.2. Спроектувати загальну концепцію взаємодії класів системи для симуляції мережі.

3.3. Спроектувати загальну концепцію взаємодії класів системи для надання доступу іншим користувачам до своєї мережі.

3.4. Реалізувати додаток відносно спроектованої архітектури.

3.5. Розробити зручний дизайн для роботи з програмою, який буде використовувати технології WPF.

4. Реалізація

4.1. Побудова та налаштування мережі

4.1.1. Написати базові класи для:

- мережевих пристроїв;
- каналів зв'язку;

4.1.2. Реалізувати класи для роботи з пристроями мережі:

- Класи OpenSystemBuilder та MediaBuilder для створення пристроїв та каналів зв'язку;

- класи OpenSystemManager та MediaManager для роботи з пристроями та каналами зв'язку;
- класи для роботи з пристроями та каналами зв'язку відносно користувацького інтерфейсу;

4.2. Симуляція роботи пристроїв

4.2.1. Написати класи та методи для:

- комунікації між пристроями;
- обробки прийнятих повідомлень;
- збереження інформації про процеси, які відбуваються під час симуляції;
- відображення результатів симуляції на користувацькому інтерфейсі;

4.3. Надання доступу іншим користувачам до своєї мережі

4.3.1. Написати класи та методи для:

- надання доступу до своєї мережі з правами на перегляд або редагування;
- комунікації між додатками для відстежування змін в реальному часі;

Результат реалізації поставленого завдання – реалізований Windows додаток на мові програмування C# з використанням технологій WPF та WCF, який дозволяє користувачу створювати комп'ютерні мережі, відстежувати процеси, які відбуваються в них та взаємодіяти з іншими користувачами за допомогою функціональності по наданню доступу до своєї мережі.

1.2. Аналіз аналогів програмного продукту

Студенти, які навчаються за спеціальностями, що входять до галузі 12. Інформаційні технології, мають перелік обов'язкових дисциплін для вивчення. Однією з таких дисциплін є «Комп'ютерні мережі», в результаті вивчення якої студенти повинні вміти пояснювати основи функціонування комп'ютерних мереж. Щоб досягти цієї мети навчання, студентам доведеться

					Державний університет «Житомирська політехніка».20.123.14.000 - ПЗ	
						12

витратити багато годин на практичне проектування, налаштування та впровадження комп'ютерної мережі. Ефективність в цьому напрямку досягається лише тоді, коли є в наявності достатньо якісного обладнання, щоб забезпечити можливість індивідуальної практики налаштування програмного забезпечення та обладнання.

Альтернативний підхід полягає у наданні студентам програмного забезпечення для моделювання мережі, хоча це не покликане повністю замінити обладнання. Однак, "це представляє корисний та економічно ефективний підхід для розуміння понять комп'ютерних мереж, протоколів та додатків краще, ніж традиційні засоби" [2].

Для цього досить часто використовують в освітньому процесі так звані симулятори. Розглянемо їх детально.

Cisco Packet Tracer - це мережевий симулятор, який може бути використаний як студентами, так й викладачами та адміністраторами мережі. Це програмне забезпечення пропонує широкий спектр комутаторів і маршрутизаторів Cisco, що працюють на IOS 12 та IOS 15, бездротових пристроїв від Linksys та декількох кінцевих пристроїв, таких як ПК та сервери з командним рядком [3].

Packet Tracer має дві робочі області - логічну та фізичну (див. рис. 1). Логічна робоча область дозволяє користувачам будувати топології логічних мереж шляхом розміщення, підключення та кластеризації пристроїв віртуальної мережі. Фізична робоча область забезпечує графічний фізичний вимір логічної мережі, що дає відчуття масштабу та розміщення того, як мережеві пристрої, такі як маршрутизатори, комутатори та хости, виглядатимуть у реальному середовищі.

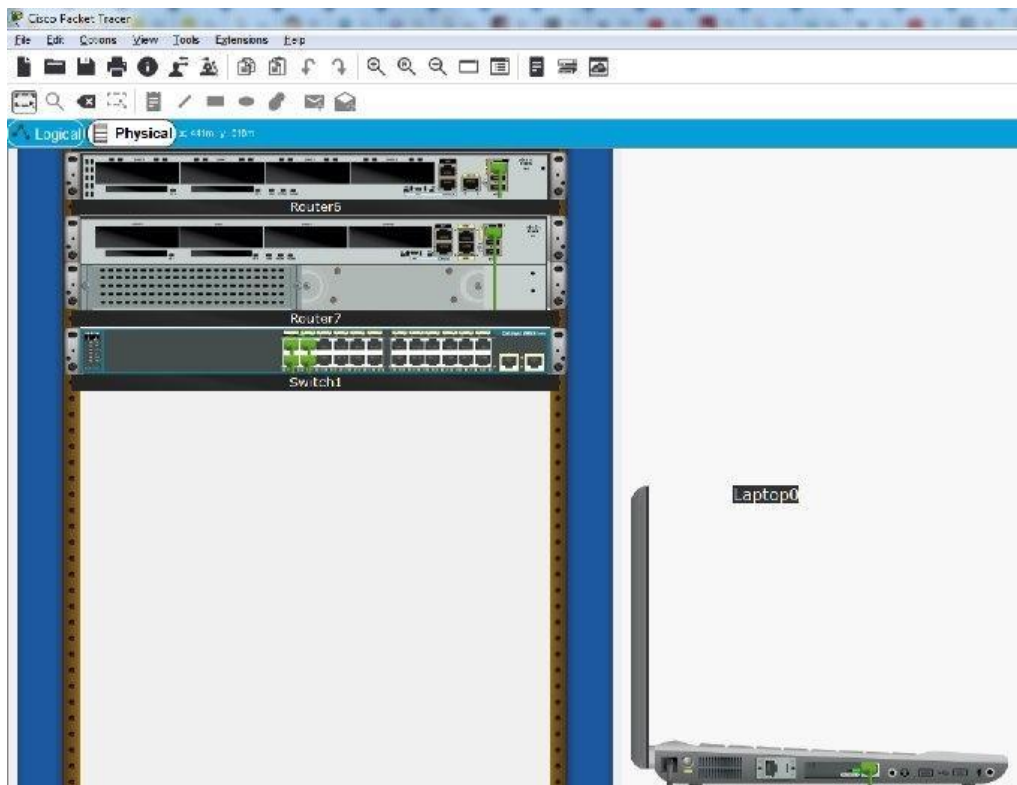


Рис. 1.1. Фізична робоча область в програмі CiscoPacketTracer

Packet Tracer має добре реалізовану імітацію роботи з налаштуванням обладнання. Для віртуальних комп'ютерів доступні всі основні мережеві налаштування, є доступ до емулятора командного рядка і потрібних мережевих утиліт. Віртуальні роутери підтримують налаштування з терміналу із можливістю використання всіх необхідних команд. Багато команд і опцій також продубльовані і в графічному інтерфейсі. Це зручно, коли доводиться мати справу з великою кількістю віртуального обладнання, оскільки це заощаджує час на виконання однотипних операцій.

Packet Tracer підтримує всі основні протоколи. Перевірити спроектовану мережу в роботі можна як в режимі «реального часу», так і включивши покроковий режим. Підтримується як спільна демонстрація (індикатори на каналах зв'язку), так і більш тонкі тести нижчого рівня, наприклад певним чином сконфігурованих пакетів між віртуальними пристроями. В результаті можна відносно швидко змодельовати і перевірити в роботі досить складні мережеві конфігурації.

До недоліків можна віднести орієнтацію виключно на обладнання Cisco - в додаток включені образи тільки обладнання Cisco. Якогось виходу в реальний світ не передбачено, проекти можна переносити між програмами тільки у вигляді файлів внутрішнього формату.

Graphical Network Simulator (GNS3) - використовується сотнями тисяч мережових інженерів по всьому світу для емуляції, налаштування, тестування та усунення несправностей з віртуальними та реальними мережами [4]. Це підтвердили і результати краудфандінгової компанії, в ході якої розробник GNS3 швидко зібрав цілком пристойну суму на розвиток програмного пакета.

Даний пакет складається з набору окремих самостійних програм, склад яких можна змінювати в ході інсталяції (див. рис. 2). Окремі додатки відповідають за власні блоки функцій. Наприклад, для імітації віртуальних маршрутизаторів використовується Dynamips, а емуляції комп'ютерів-хостів - VirtualBox. GNS3 грає роль платформи, що дозволяє зручно пов'язати їх в цілісну наочну середу. Доступні плагіни, розширення та інші опції (TracerouteNG, PortScanner, NetworkConfiguration, Real-TimeNetFlowAnalyzer). Свіжі версії необхідних компонентів завантажуються з мережі автоматично в ході установки пакета GNS3.

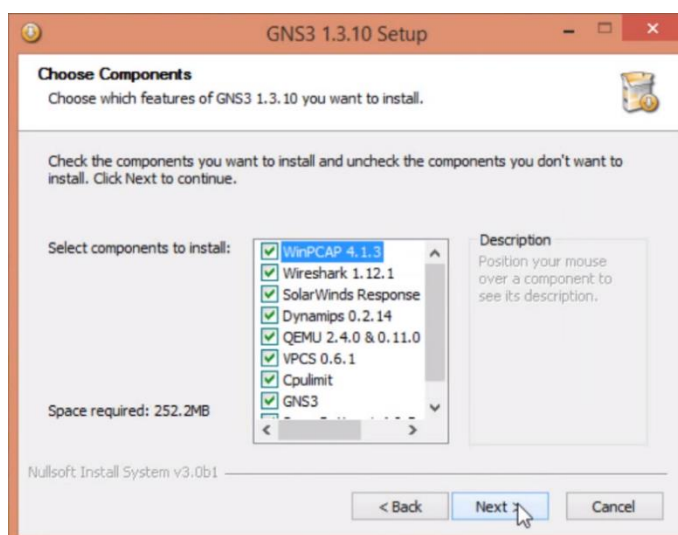


Рис. 1.2. Список програм, які встановлюються з GNS3

За замовчуванням в поставку GNS3 входять моделі тільки найпростіших мережових пристроїв, причому умовних, не прив'язаних до конкретних брендів.

Це пов'язано з тим, що розробники GNS3 прямого відношення до виробників мережевого обладнання не мають, відповідно, прав включати образи програмного забезпечення фізичних пристроїв до складу свого пакета у них немає. Тому шукати і встановлювати образи справжніх маршрутизаторів доведеться окремо і самостійно. Однак, такий підхід підвищує точність моделювання за рахунок використання реальних образів, а не умовних імітацій.

Одна з особливостей GNS3 - можливість зв'язати віртуальні пристрої з реальними. Наприклад, в базу пакета входять додатки Wireshark і WinPcap, що дозволяють маніпулювати даними, переданими в реальних мережах. З'являється можливість роботи не тільки в локальній віртуальній пісочниці, але і в реальній мережі зі справжніми пристроями. В результаті GNS3 можна використовувати не тільки для моделювання, але і для аналізу роботи існуючих реальних комп'ютерних мереж. Більш того, можна змішати в одному проекті віртуальні та фізичні пристрої. Іншими словами, реальний комп'ютер зможе отримати справжній трафік і обробити його за допомогою віртуального мережевого пристрою, віддавши в мережу імітацію відповідної відповіді.

У GNS3 є деякі недоліки, про які потрібно знати. Хоча теоретично не так важливо, образи якого обладнання будуть використовуватись, реально знайти, встановити і отримати консультацію по використанню найлегше для образів пристроїв Cisco. Крім цього не варто забувати, що GNS3 має високі системні вимоги. Оскільки GNS3 буде повноцінні віртуальні моделі, без підробок і імітацій, у пакета на більш-менш складних проектах з'являються серйозні системні вимоги до комп'ютера, на якому йде моделювання.

1.3. Обґрунтування вибору інструментальних засобів

Для реалізації симулятора комп'ютерних мереж обрано мову програмування C# та технології WPF та WCF в поєднанні зі зручним візуальним редактором коду Visual Studio. Далі буде описано ці технології та переваги, які вплинули на вибір саме цих технологій.

C# - типобезпечна, об'єктно-орієнтована мова програмування, що дозволяє розробникам створювати безпечні і надійні додатки, що працюють на .NETFramework. C# можна використовувати для створення клієнтських додатків Windows, XML-веб-служб, розподілених компонентів, додатків клієнт-сервер, додатків баз даних та багато іншого.[5]

До ключових переваг C# можемо віднести:

- простота розробки;
- швидкість розробки;
- об'єктно-орієнтованість;
- компонентно-орієнтованість;
- масштабованість і оновлюваність;
- структурованість.

Windows Presentation Foundation (WPF) - це UI платформа, яка входить до складу .NET, дозволяє створювати клієнтські додатки для настільних систем. Платформа розробки WPF підтримує широкий набір функцій розробки додатків, включаючи модель програми, ресурси, елементи керування, графіку, макет, прив'язку даних, документи та безпеку. WPF використовує Extensible Application Markup Language (XAML) для надання декларативної моделі програмування додатків [6].

Переваги використання WPF:

- можливість декларативного визначення графічного інтерфейсу за допомогою спеціальної мови розмітки XAML;
- велика кількість вбудованих елементів керування;
- функціональність орієнтована на використання паттерну MVVM;
- апаратне прискорення графіки, що збільшує продуктивність додатків та робить графіку більш пальною.

Windows Communication Foundation (WCF) - це платформа для створення додатків, орієнтованих на служби. За допомогою WCF можна передавати дані у вигляді асинхронних повідомлень з однієї кінцевої точки в іншу. Кінцева точка служби може входити в постійно доступну службу, яка повинна розміщуватись

в IIS, або представляти службу, яка повинна розміщуватись в додатку. Кінцева точка може бути клієнтом служби, яка запитує дані від кінцевої точки. Повідомлення можуть представляти одиночний символ або одне слово, що відправляється в форматі XML, або мати вигляд складного потоку двійкових даних [7].

Ключові переваги використання WCF:

- орієнтованість на служби;
- безпека та надійність (вона надається у вигляді аудиту, аутентифікації, авторизації, конфіденційності та цілісності повідомлень);
- висока швидкість передачі даних;
- висока продуктивність;
- простота налаштування за допомогою конфігураційного файлу;
- різні способи розміщення (Self-hosting, Windows Service, IIS, WAS).

VisualStudio – це потужне інтегроване середовище розробки (IDE), яке можна використовувати для редагування, налагодження, створення коду та подальші публікації додатку. Інтегроване середовище розробки (IDE) - це багатофункціональна програма, яка може бути використана для багатьох аспектів розробки програмного забезпечення. Окрім стандартного редактора та відладчика, яких надають більшість IDE, Visual Studio включає компілятори, інструменти для завершення коду, графічні дизайнери та багато інших функцій для полегшення процесу розробки програмного забезпечення [8].

1.4. Опис властивостей комп'ютерних мереж, пристроїв на протоколів.

Еталонна модель взаємодії відкритих систем (Open System Interconnection - OSI) служить в якості набору самих основних елементів комп'ютерних мереж з моменту створення в 1984 році. Модель OSI базується на пропозиції, розробленій Міжнародною організацією зі стандартизації (International Standards Organization - ISO). Початкова мета моделі OSI полягала у створенні набору стандартів дизайну для виробників обладнання, щоб вони могли спілкуватися один з одним.

Модель OSI визначає ієрархічну архітектуру, яка логічно розділяє функції, необхідні для підтримки зв'язку між системами.

Модель OSI має сім рівнів, кожен з яких має різний рівень абстракції і виконує чітко визначену функцію. Наступні принципи, застосовані для досягнення семи рівнів (Фейг) [9]:

- рівень повинен бути створений там, де потрібен різний рівень абстракції;
- кожен рівень повинен виконувати чітко визначену функцію.
- функцію кожного рівня слід вибирати з урахуванням визначення міжнародних стандартизованих протоколів;
- межі рівня слід вибирати для мінімізації потоку інформації через інтерфейси;
- кількість рівнів має бути достатньо великою, щоб різні функції не потрібно було перекидати в один і той же рівень із необхідних, і достатньо малий, щоб архітектура не стала громіздкою.

Сім рівнів OSI, починаючи з найнижчого:

1. Фізичний рівень – відповідає за передачу та прийом потоків біт через фізичне середовище.
2. Канальний рівень – відповідає за надійну передачу кадрів даних між двома вузлами, з'єднаними на фізичному рівні.
3. Мережевий рівень – відповідає за структурування та управління багатовузловою мережею, включаючи адресацію, маршрутизацію та контроль трафіку.
4. Транспортний рівень – відповідає за надійну передачу сегментів даних між точками в мережі, включаючи сегментацію, підтвердження та мультиплексування.
5. Сеансовий рівень – відповідає за встановлення зв'язку, підтримку сеансів, аутентифікацію, а також забезпечує безпеку.
6. Представницький рівень – відповідає за переміщення даних між мережевою службою та додатком; включає кодування символів, стиснення даних, шифрування та дешифрування.

7. Прикладний рівень – надає набір інтерфейсів для додатків для отримання доступу до мережеслужб.

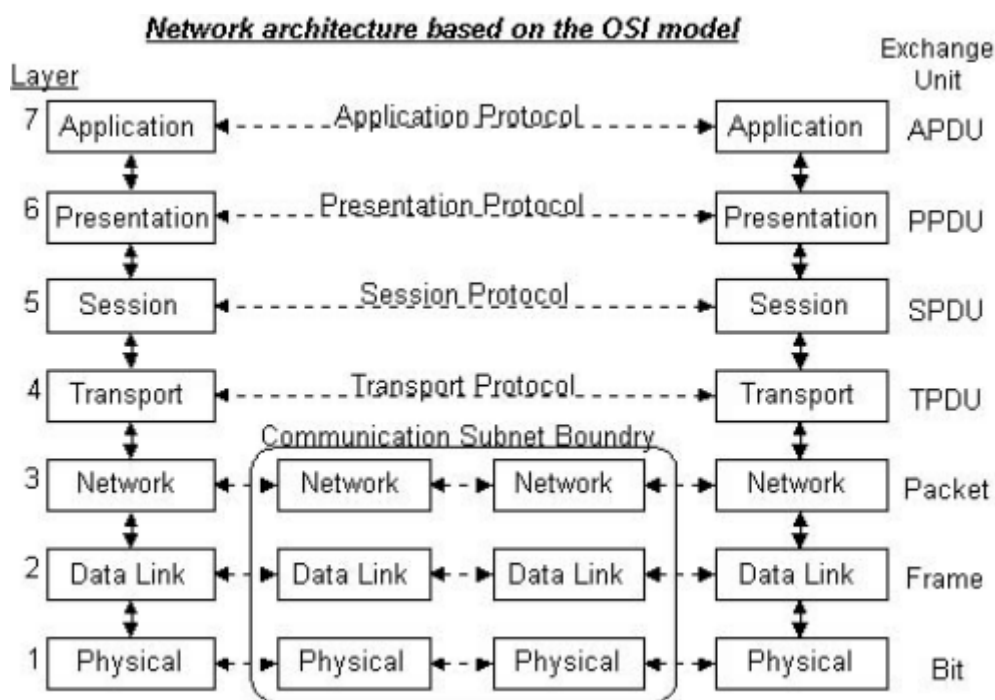


Рис. 1.3. Мережева архітектура на основі моделі OSI

Контролер мережевого інтерфейсу (NIC) – це основне обладнання, яке використовується для підключення до мережі. Незважаючи на те, що NIC традиційно асоціюються з ПК, ноутбуками та серверами, вони можуть існувати майже на будь-якому мережевому пристрої, включаючи принтери, телефони та сканери. У деяких мережевих апаратних засобах, таких як комутатори, що використовуються для мережевих масивів зберігання даних, є змінні модулі, які дозволяють використовувати різні типи з'єднання. Ці модулі також технічно є NIC.

NIC конвертує пакети даних між двома різними технологіями передачі даних. Кінцевий пристрій використовує технологію паралельної передачі даних для передачі даних між її внутрішніми частинами, тоді як носій, що забезпечує зв'язок між різними пристроями, використовує технологію послідовного передавання даних.

Зазвичай, сучасні мережеві пристрої мають вбудовані NIC. Якщо потрібні додаткові NIC, вони також доступні як додаткові пристрої окремо. Для

настільних або серверних систем вони доступні у формі адаптера, який можна підключити до доступних слотів.

Існує два типи NIC:

1. На основі середовища передачі даних NIC використовуються відповідно до типу середовища передачі даних. Для підключення різних типів середовищ використовуються різні типи NIC. Для підключення конкретного типу носія ми повинні використовувати NIC, який спеціально створений для цього типу середовища.
2. На основі дизайну мережі - специфічний дизайн мережі потребує спеціальну NIC. Наприклад, FDDI, Token Ring і Ethernet мають свій характерний тип карт NIC. Вони не можуть використовувати інші типи карт NIC.

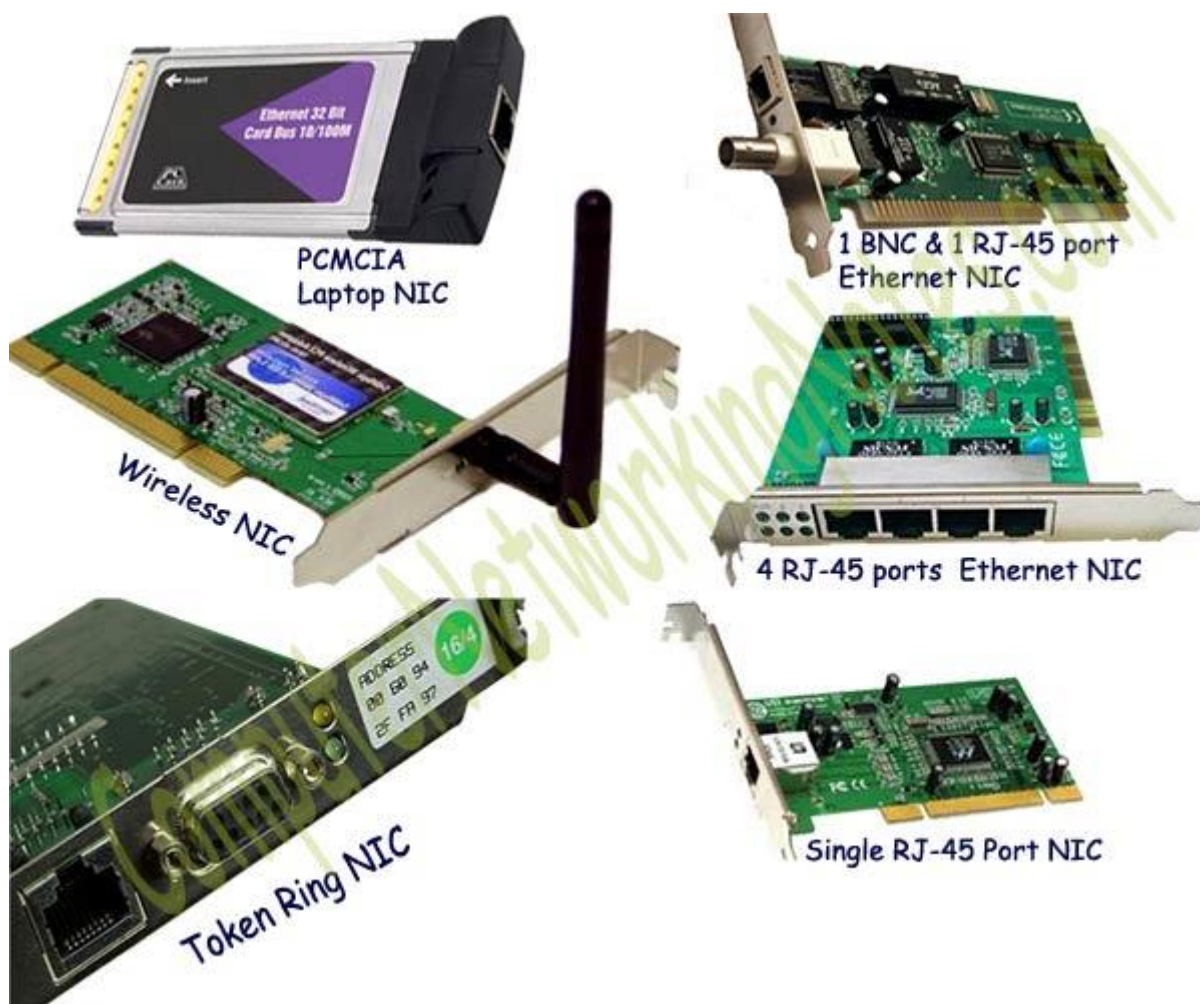


Рис. 1.4. Поширені типи мережевих карт

Стек протоколів TCP/IP— це набір протоколів мережі Інтернет. Назва походить від назви стрижневих протоколів мережі Інтернет — IP (Internet

Protocol — «міжмережевий протокол») і TCP (Transmission Control Protocol — «протокол керування передаванням»). Фактично це систематизований стек протоколів, що поділяється на чотири рівні, які корелюються з еталонною моделлю OSI.

Рівні TCP/IP:

1. Прикладний рівень
2. Транспортний рівень
3. Міжмережевий рівень
4. Канальний рівень

Канальний рівень описує спосіб кодування даних для передачі пакета даних на фізичному рівні (тобто спеціальні послідовності біт, що визначають початок і кінець пакету даних, а також забезпечують перешкодостійкість).

Приклади протоколів канального рівня: Ethernet, IEEE 802.11 WLAN, SLIP, Token Ring, ATM і MPLS.

Міжмережевий рівень визначає протоколи, які відповідають за логічну передачу даних по всій мережі.

Основними протоколами, що знаходяться на цьому шарі, є:

1. IP (Internet Protocol) - відповідає за доставку пакетів від вихідного хоста до цільового вузла, переглядаючи IP-адреси в заголовках пакетів. IP має 2 версії - IPv4 та IPv6.
2. ICMP (Internet Control Message Protocol) - інкапсульований в межах дейтаграм IP і відповідає за надання хостам інформації про проблеми в мережі.
3. ARP (Address Resolution Protocol) – відповідає за знаходження апаратної адреси хоста за відомою IP-адресою. ARP має кілька типів: Reverse ARP, Proxy ARP, Gratuitous ARP and Inverse ARP.

Транспортний рівень несе відповідальність за безперервне спілкування та безперешкодну доставку даних. Він захищає додатки верхнього рівня від складності даних.

Два основних протоколи, присутні в цьому шарі:

1. TCP (Transmission Control Protocol) – він забезпечує надійну та безаварійну комунікацію між кінцевими системами. Він відповідає за послідовність і сегментацію даних. Він також має функцію підтвердження і контролює потік даних через механізм управління потоком.
2. UDP (User Datagram Protocol) - протокол передачі датаграм без встановлення з'єднання. UDP не надає можливості упевнитися про доставку повідомлення одержувачу, а також можливого перемішування пакетів. У додатках, що вимагають гарантованої передачі даних, використовується протокол TCP. UDP зазвичай використовується в таких додатках, як потокове відео і комп'ютерні ігри, де допускається втрата пакетів, а повторний запит утруднений або невиправданий, або в додатках виду запит-відповідь (наприклад, запити до DNS), де створення з'єднання займає більше ресурсів, ніж повторна відправка.

Рівень додатків включає протоколи, які використовуються більшістю програм для надання послуг користувача або обміну даними програми через мережеві з'єднання, встановлені протоколами нижчого рівня.

Приклади протоколів рівня додатків включають протокол передачі гіпертексту (HTTP), протокол передачі файлів (FTP), протокол простої передачі пошти (SMTP) та протокол конфігурації динамічного хоста (DHCP).

Висновки до розділу 1

В цьому розділі представлена мета, на підставі якої були проаналізовані та представлені основні завдання та методи їх вирішення. Була описана основна функціональність та представлений план реалізації цієї функціональності для створення даного програмного продукту.

Крім цього були проаналізовані аналоги симуляторів та емуляторів комп'ютерних мереж, а саме їх переваги та недоліки.

Описано мову програмування та технології, за допомогою яких реалізовано додаток. Наведено переваги, які вплинули на вибір саме цих технологій.

Наведено та описано властивості комп'ютерних мереж та мережевих пристроїв, які використано для реалізації додатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ СИМУЛЯТОРА КОМП'ЮТЕРНИХ МЕРЕЖ

2.1. Визначення варіантів використання та структури системи.

Симулятор комп'ютерних мереж та основних мережевих протоколів має надавати певні можливості. А саме проектування комп'ютерних мереж та дослідження їх роботи.

Проаналізувавши вимоги користувачів, виділимо наступні варіанти використання:

1. Створення проекту мережі.
2. Вибір та створення мережевих пристроїв.
3. Вибір та створення з'єднань між пристроями.
4. Налаштування IP-адресації для створених пристроїв.
5. Перегляд мережевого логу пристроїв.
6. Створення та відправка трафіку між пристроями.
7. Можливість відкрити доступ до свого проекту.
8. Можливість приєднатися до іншого проекту.



Рис. 2.1. Варіанти використання

Функціональні вимоги:

1. Побудова комп'ютерної мережі: в системі повинен бути набір запрограмованих пристроїв та з'єднань, за допомогою яких можна створювати та розширювати комп'ютерну мережу.
2. Робота декількох людей над одним проектом: користувач має можливість відкрити доступ до свого проекту.

Аналіз функціональних вимог дозволив виділити наступні сутності, за допомогою яких реалізуємо програмний продукт.

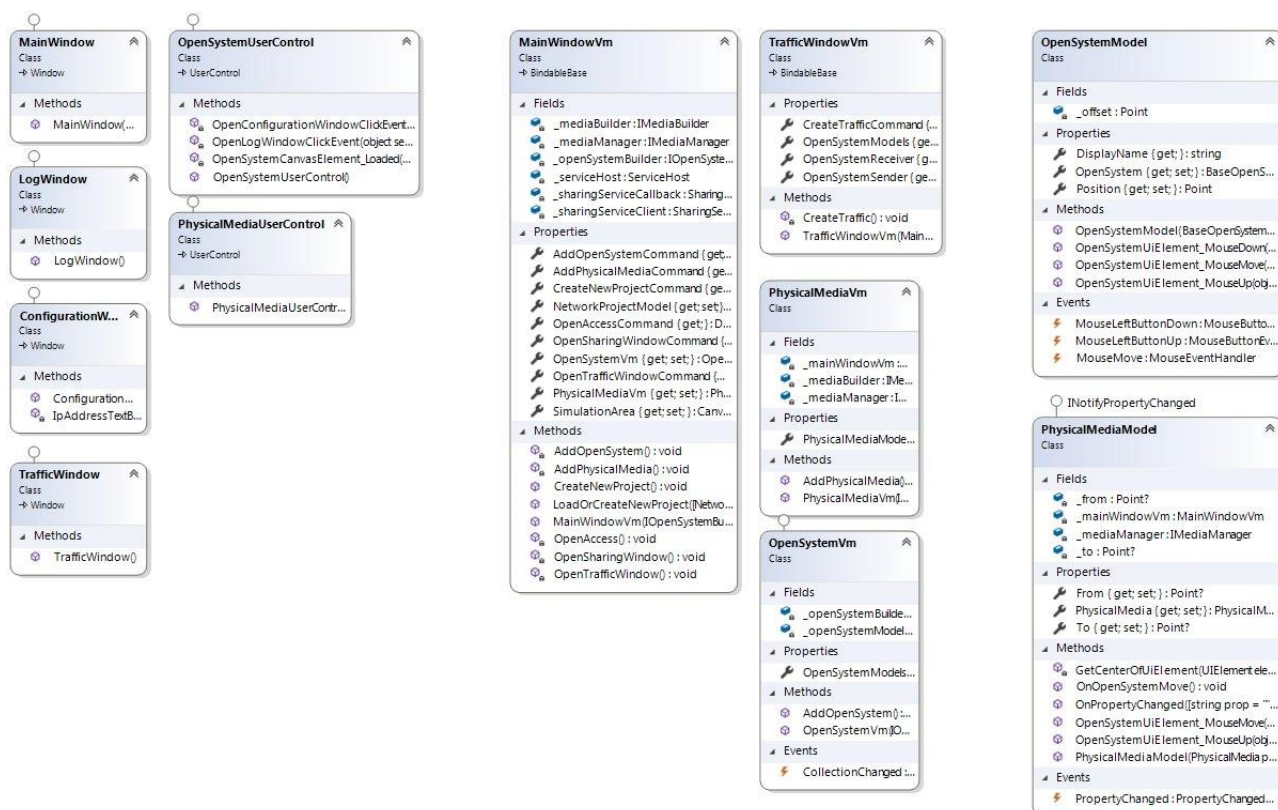


Рис. 2.2. Діаграма класів, які відповідають за роботу з графічним інтерфейсом за допомогою паттерну MVVM

Model – View – ViewModel (MVVM) – це архітектурний шаблон, який дозволяє відокремити логіку додатку від графічного інтерфейсу.

Модель (Model) описує використовувані в додатку дані. Моделі можуть містити логіку, пов'язану з цими даними, наприклад, логіку валідації властивостей моделі. У той же час модель не повинна містити ні якої логіки, пов'язаної з відображенням даних і взаємодією з візуальними елементами управління.

Нерідко модель реалізовує інтерфейси INotifyPropertyChanged або

INotifyCollectionChanged, які дозволяють повідомляти систему про зміни властивостей моделі. Завдяки цьому полегшується прив'язка до представлення, хоча пряма взаємодія між моделлю і представленням відсутня.

Представлення (View) визначає візуальний інтерфейс, через який користувач взаємодіє з додатком. Стосовно до WPF представлень - це код в xaml, який визначає інтерфейс у вигляді кнопок, текстових полів та інших візуальних елементів.

Хоча вікно (клас Window) в WPF може містити як інтерфейс в xaml, так і прив'язаний до нього код C#, проте в ідеалі код C# не повинен містити ніякої логіки, крім конструктора, який викликає метод InitializeComponent і виконує початкову ініціалізацію вікна. Однак в файлі коду може знаходитися певна логіка, яку важко реалізувати в рамках паттерна MVVM у ViewModel.

Модель представлення (ViewModel) пов'язує модель і представлення через механізм прив'язки даних. Якщо в моделі змінюються значення властивостей, при реалізації моделлю інтерфейсу INotifyPropertyChanged, автоматично йде зміна відображуваних даних в поданні, хоча безпосередньо модель і уявлення не пов'язані.

Модель представлення також містить логіку по отриманню даних з моделі, які потім передаються в уявлення. Також модель представлення визначає логіку по оновленню даних в моделі.

Оскільки елементи представлення, тобто візуальні компоненти типу кнопок, не використовують події, то представлення взаємодіє з моделлю представлення за допомогою команд.

Наприклад, користувач хоче зберегти введені в текстове поле дані. Він натискає на кнопку і тим самим відправляє команду до моделі представлення. А модель представлення вже отримує передані дані і відповідно до них оновлює модель.

Підсумком застосування патерну MVVM є функціональний розподіл програми на три компонента, які простіше розробляти і тестувати, а також в

подальшому модифікувати і підтримувати.

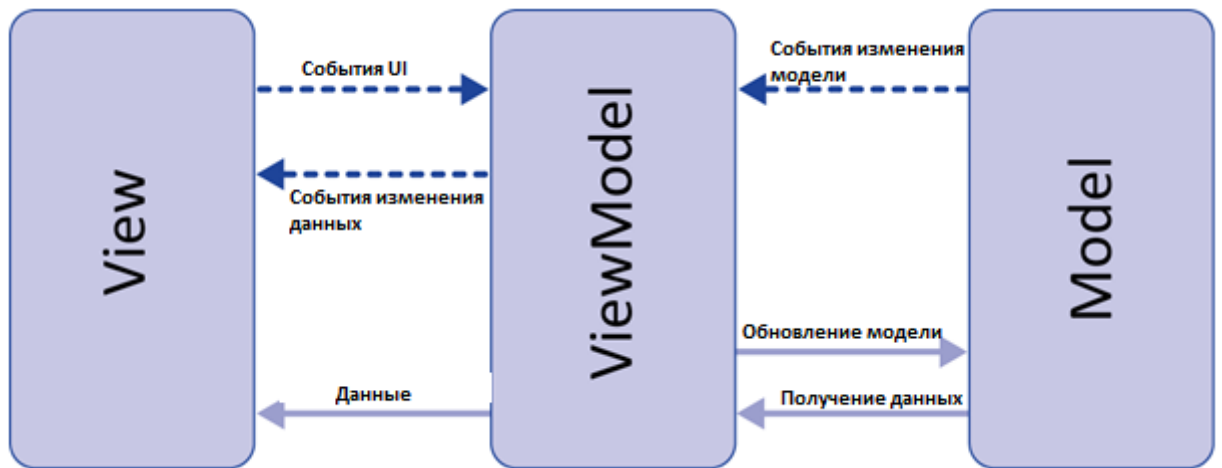


Рис. 2.3. Паттерн MVVM

На діаграмі, зображеній на рис. 2.2, можна побачити 4 класи вікон та 2 класи графічних елементів – MainWindow, LogWindow, ConfigurationWindow, TrafficWindow, OpenSystemUserControl, PhysicalMediaUserControl.

Клас MainWindow відповідає за відображення основних елементів взаємодії з користувачем. Також клас MainWindow відображає пристрої та з'єднання, додані користувачем.

Клас LogWindow відповідає за відображення логів пристрою, за допомогою якого можна простежити що відбувалося на пристрої.

Клас ConfigurationWindow відповідає за відображення інформації пристрою та редагування його IP-адресації.

Клас TrafficWindows відповідає за створення та відправку штучного трафіку від одного пристрою до іншого.

Класи OpenSystemUserControl та PhysicalMediaUserControl відповідають за графічне відображення пристроїв та з'єднань, а також взаємодію з користувачем на вікні MainWindow.

Також на діаграмі зображено 4 класи моделей представлень (MainWindowVm, TrafficWindowVm, PhysicalMediaVm, OpenSystemVm), які містять в собі логіку взаємодії з користувачем, та 2 моделі (OpenSystemModel, PhysicalMediaModel), які зберігають інформацію про пристрої та з'єднання.

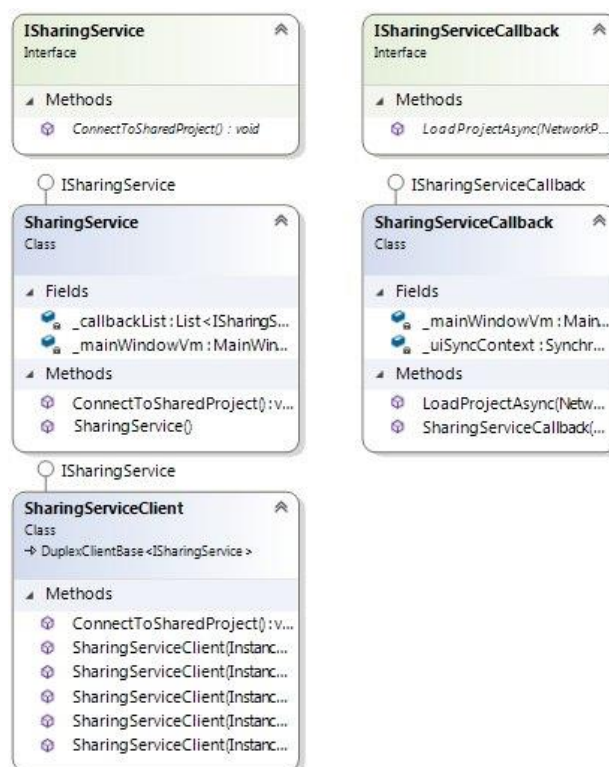


Рис. 2.3. Діаграма класів для комунікації додатків один з одним

На діаграмі, зображеній на рис. 2.3, можна побачити класи для комунікації додатку з іншим додатком такого типу. Класи реалізовано за допомогою платформи WCF. Вони спроектовані та реалізовані таким чином, щоб додаток міг працювати у клієнт-серверному режимі.

Клас `SharingService` містить функціональність, яка відповідає за роботу додатку у серверному режимі.

Клас `SharingServiceClient` містить функціональність, яка відповідає за роботу додатку у клієнтському режимі.

Клас `SharingServiceCallback` містить функціональність, яка допомагає серверу взаємодіяти з клієнтом.

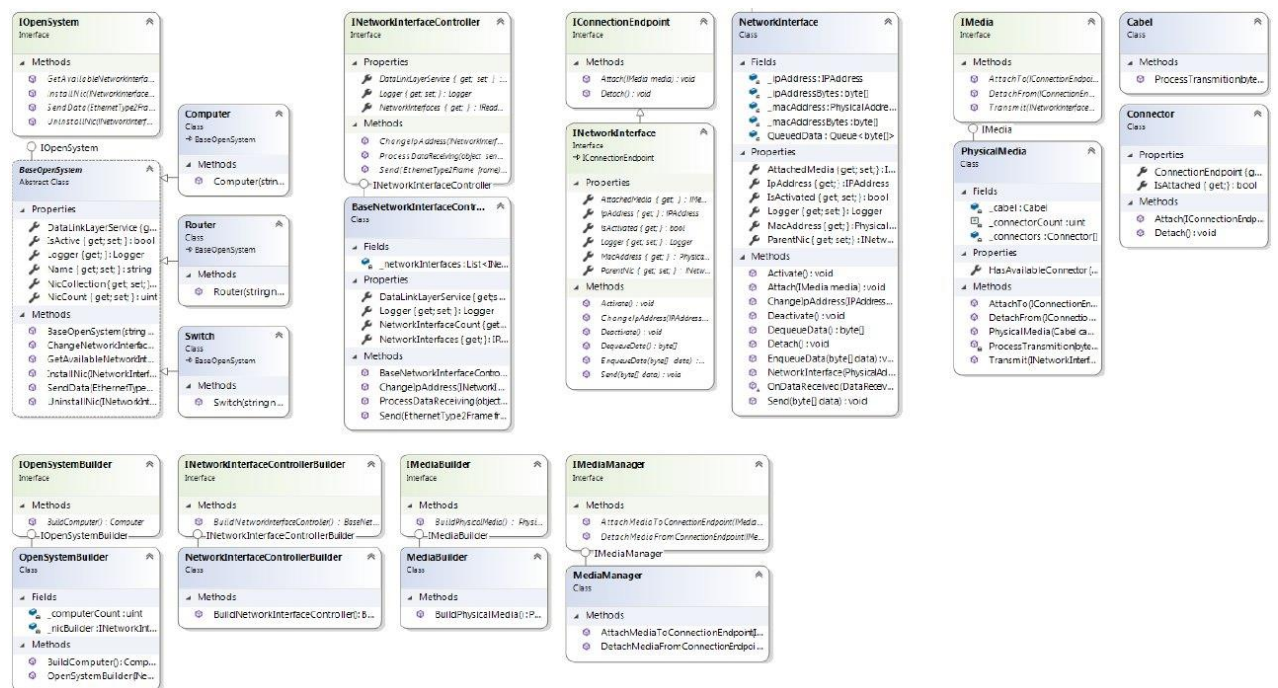


Рис. 2.4. Діаграма класів, які представляють собою мережеве обладнання

На діаграмі, зображеній на рис. 2.4, видно класи, які представляють собою мережеві пристрої: BaseOpenSystem, Computer, Router, Switch.

Клас BaseOpenSystem є батьківським відношенням до класів Computer, Router та Switch. Він містить в собі функціональність та дані, які є базовими для всіх мережевих пристроїв.

Класи Computer, Router та Switch перевизначають функціональність, реалізовану в базовому класі.

На діаграмі зображено класи BaseNetworkInterfaceController та NetworkInterface для роботи з мережевими картами та мережевими інтерфейсами. В цих класах реалізовано функціональність для відправки, отримання та обробки даних.

Також на діаграмі зображено класи PhysicalMedia, Cabel та Connector, в яких реалізовано функціональність для з'єднання мережевих пристроїв та передачу даних.

Крім цього, на діаграмі можна побачити 3 класи для створення мережевого обладнання OpenSystemBuilder, NetworkInterfaceControllerBuilder, PhysicalMediaBuilder та клас PhysicalMediaManager для управління мережевими з'єднаннями, а саме підключення до мережевих пристроїв.

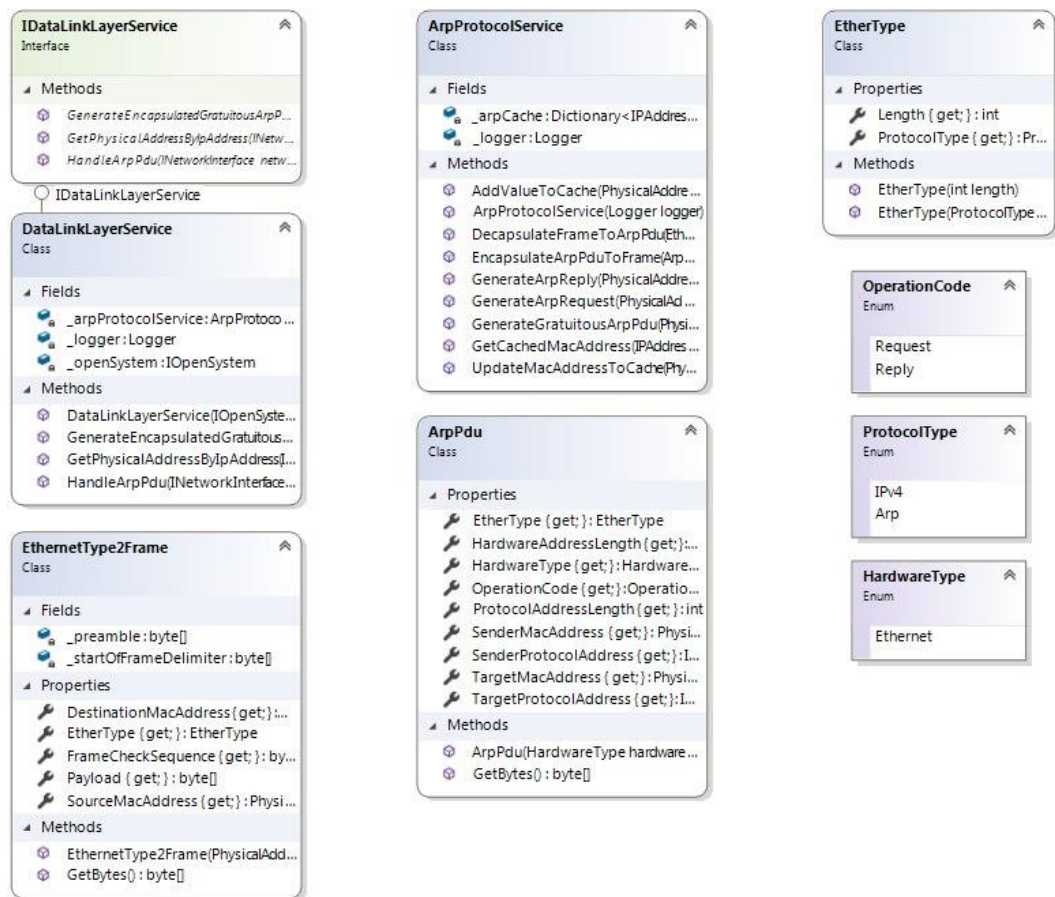


Рис. 2.5. Діаграма класів для роботи з Ethernet кадрами та протоколами канального рівня

На діаграмі, зображених на рис. 2.5, зображено класи, в яких реалізовано роботу з канальним рівнем та протоколом ARP.

В класі `DataLinkLayerService` реалізовано функціональність для роботи з протоколами канального рівня. Метод `HandleArpPdu` містить основну функціональність по обробці ARP пакетів.

Клас `ArpProtocolService` містить функціональність для роботи з протоколом ARP. В класі створено поле `_arpCache`, яке зберігає закешовані пари MAC-адреса IP-адрес.

Методи, реалізовані в цьому класі:

- `AddValueToCache` – додає пару до кешу, якщо такої пари нема;
- `DecapsulateFrameToArpPdu` – декапсулює отриманий фрейм в ARP пакет;
- `EncapsulateArpPduToFrame` – інкапсулює ARP пакет у фрейм;

- `GenerateArpReply` – створює ARP пакет на основі вхідних параметрів, автоматично заповнює значенням `Reply` поле `OperationCode`, яке відповідає за визначення типу пакету (`Request`, `Reply`);
- `GenerateArpRequest` – створює ARP пакет на основі вхідних параметрів, автоматично заповнює значенням `Request` поле `OperationCode`, яке відповідає за визначення типу пакету (`Request`, `Reply`);
- `GenerateGratuitousArpPdu` – створює ARP пакет на основі вхідних параметрів, пакет створюється таким чином, щоб в результаті отримати `GratuitousARP` пакет;
- `GetCachedMacAddress` – шукає значення закешованої MAC-адреси за допомогою IP-адреси. Якщо закешованого значення не додано, повертається значення `null` для подальшої обробки;
- `UpdateMacAddressToCache` – шукає значення закешованої MAC-адреси за допомогою IP-адреси та оновлює її значення.

Клас `EthernetType2Frame` представляє Ethernet кадр, який використовується для комунікації між пристроями. Переглянувши рис. 2.6, можна співставити поля класу з полями кадру Ethernet.

Клас `ArpPdu` представляє ARP пакет, який, при передачі на рівень нижче, інкапсулюється в Ethernet кадр. Переглянувши рис. 2.7, можна співставити поля класу з полями пакету ARP.

Клас `EtherType` та перелічення `OperationCode`, `ProtocolType`, `HardwareType` є складовими частинами класів `EthernetType2Frame` та `ArpPdu`.

802.3 Ethernet packet and frame structure

Layer	Preamble	Start of frame delimiter	MAC destination	MAC source	802.1Q tag (optional)	Ethertype (Ethernet II) or length (IEEE 802.3)	Payload	Frame check sequence (32-bit CRC)	Interpacket gap
	7 octets	1 octet	6 octets	6 octets	(4 octets)	2 octets	46-1500 octets	4 octets	12 octets
Layer 2 Ethernet frame	← 64-1522 octets →								
Layer 1 Ethernet packet & IPG	← 72-1530 octets →								← 12 octets →

Рис. 2.6. Формат Ethernet кадру на фізичному та каналному рівнях

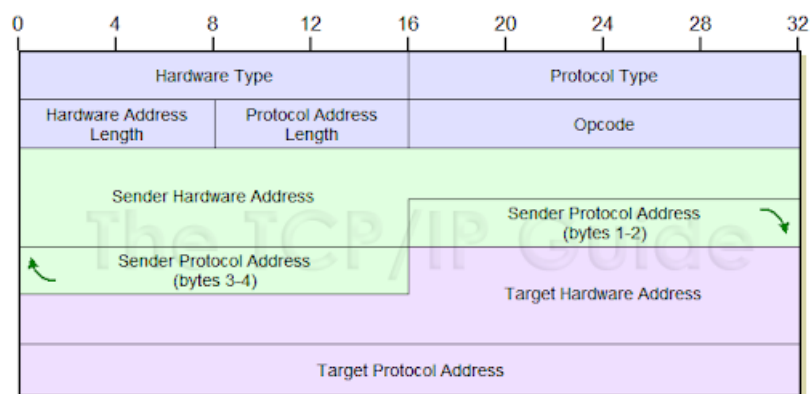


Рис. 2.7 Формат ARP пакету

2.2. Проектування та реалізація алгоритмів роботи системи

Після запуску програми (рис.2.8) симулятора комп'ютерних мереж користувач має можливість створити новий проект, додати до нього пристрої, додати з'єднання між пристроями, створити трафік з одного пристрою до іншого, відкрити доступ до проекту або підключитися до іншого проекту. Також після створення пристрою, при натисканні правої клавіші миші на пристрої, з'являється допоміжний список з вибором – відкрити вікно налаштування або вікно логу цього пристрою.

При виборі варіанту створення нового проекту, створюється команда, яка очищає всі поля, які були заповнені раніше та ініціалізує їх новими значеннями для нового проекту.

Варіант додання пристрою до проекту створює команду, яка ініціалізує новий пристрій та додає його в колекцію створених пристроїв. Ця колекція зв'язана з представленням головного вікна, тобто при оновленні (додання або видалення) колекції, на головному вікні будуть з'являтися графічні елементи, з якими можна взаємодіяти.

Додання з'єднання працює схожим принципом як і додання пристрою. В проекті повинно бути хоча б 2 пристрої, щоб мати можливість з'єднати їх.

При виборі варіанту створення трафіку, відкривається нове вікно, в якому можна обрати з якого пристрою буде надсилатися трафік, до якого пристрою цей трафік повинен прийти та пакети якого протоколу повинні надсилатися. Після

заповнення усіх полів користувач натискаю клавішу відправки, яка генерує команду запуску процесу відправки трафіку.

Після створення пристрою, користувач може відкрити вікно налаштування цього пристрою, де показується його основна інформація та таблиця мережевих інтерфейсів, де можна побачити MAC-адресу та IP-адресу кожного інтерфейсу. Також для кожного інтерфейсу можна змінити IP-адресу.

Після створення пристрою та деякої його роботи, наприклад, з'єднанні його з іншим пристроєм або відправки трафіку, можна відкрити вікно логу, в якому зображено список дій пристрою.

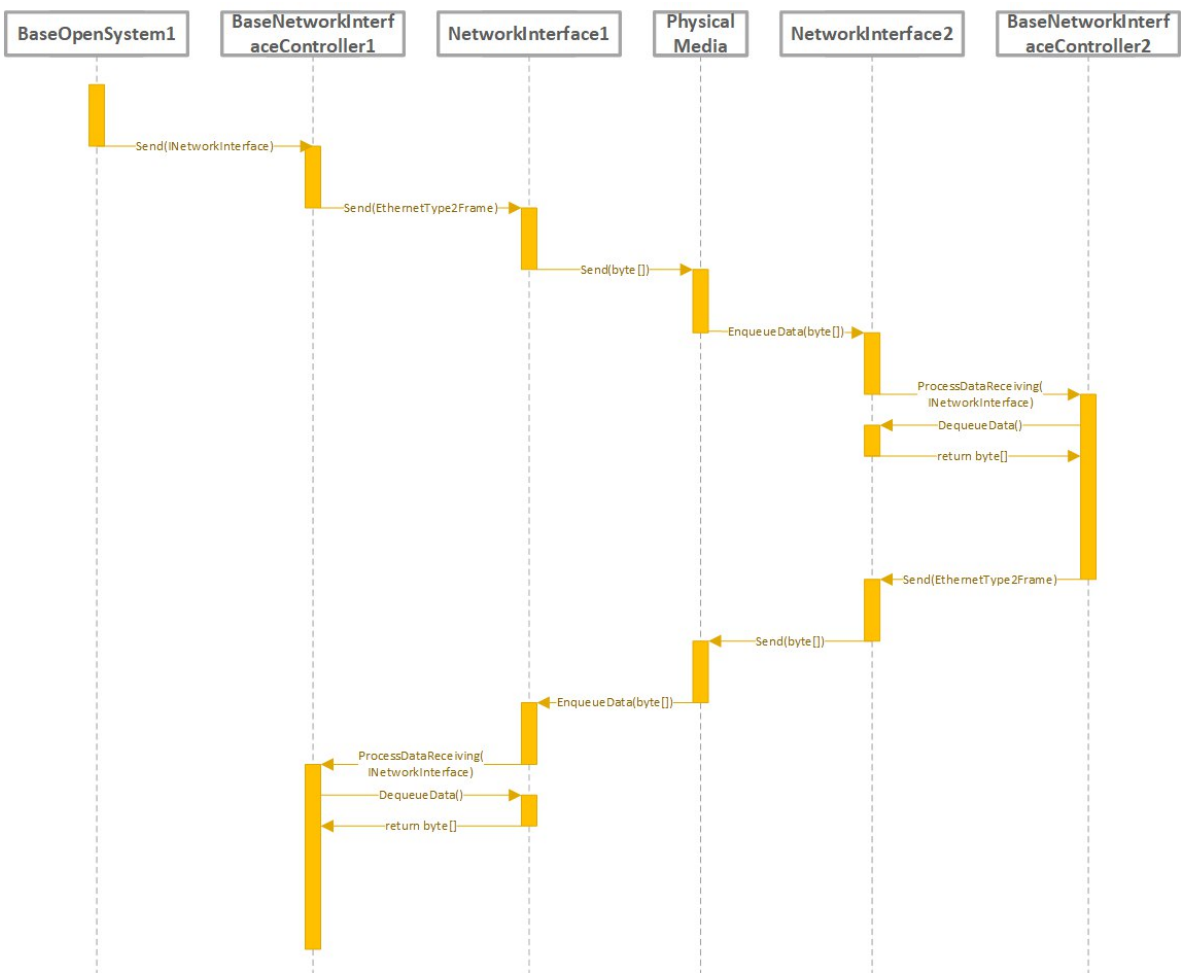


Рис. 2.8. UMLSequence діаграма процесу відправки даних та отримання відповіді

Один з основних процесів симулятора комп'ютерних мереж – це відправка даних з одного пристрою до іншого (рис. 2.8) при умові, що вони з'єднані один з одним. Процес відправки даних починається в класі BaseOpenSystem в методі SendData. Цей метод приймає в параметрах екземпляр класу

EthernetType2Frame, і знаючи з якої MAC-адреси повинні надіслатися дані, обирає мережевий інтерфейс з відповідною MAC-адресою та викликає метод Send цього інтерфейсу. Метод Send мережевого інтерфейсу, отримавши дані перетворює їх в набір байтів, після чого надсилає дані далі по ланцюгу до екземпляру класу PhysicalMedia, де відбувається імітація проходження даних через канал зв'язку, після чого ці дані відправляється до наступного пристрою, який приймає, а потім обробляє вхідні дані.

Отримавши дані, мережевий інтерфейс надсилає їх в мережеву карту (клас BaseNetworkInterfaceController), до якої належить. Мережева карта, отримавши дані, перетворює їх з набору байтів в екземпляр класу EthernetType2Frame, на основі якого відбувається обробка. За обробку отриманого кадру відповідає спеціальний сервіс (клас DataLinkLayerService), який на основі поля EtherType, обирає сервіс, який буде обробляти дані.

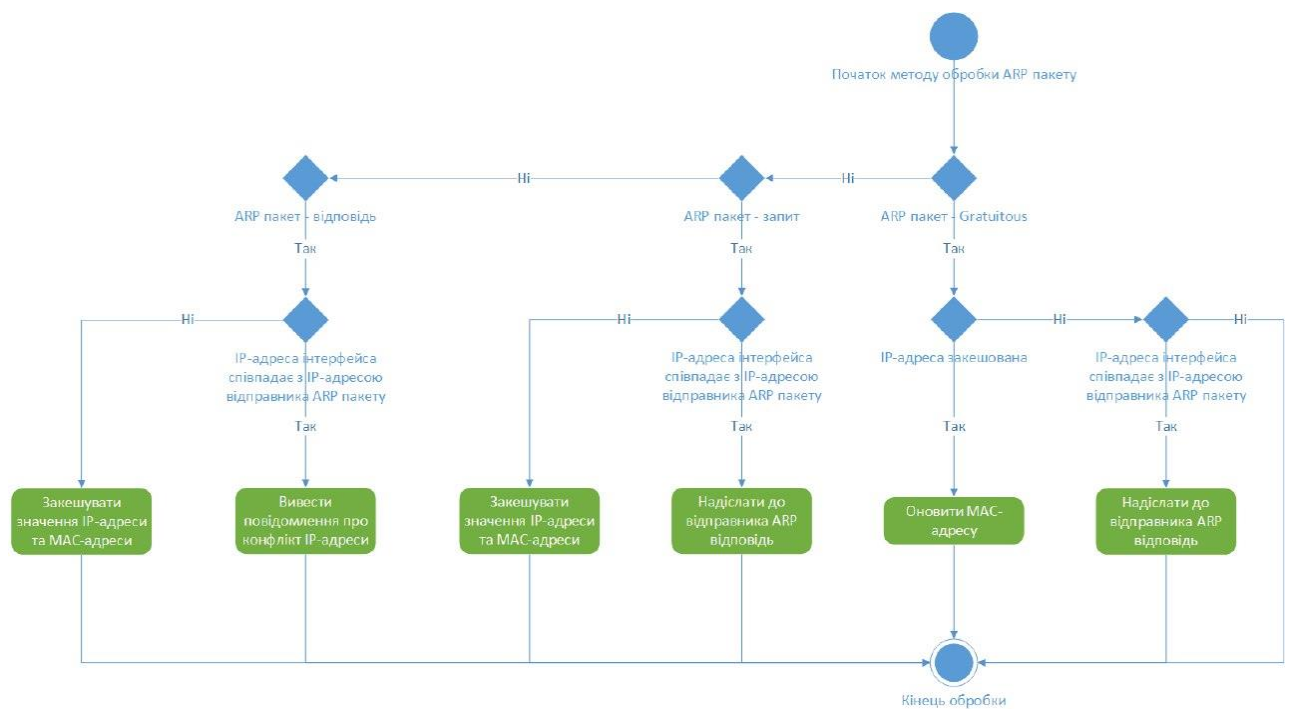


Рис. 2.9. UMLActivity діаграма, яка показує процес обробки ARP пакету

Отримавши кадр, мережева карта декапсулює його та передає в метод HandlePdu класу DataLinkLayerService, який в свою чергу, за допомогою допоміжного сервісу ArpProtocolService, обробляє пакет, як зображено на рис. 2.9.

2.3. Проектування та реалізація міждодаткової комунікації

Ключовий клас, за допомогою відбувається хостинг та комунікація додатку – `ServiceHost`. Він є частиною простору імен `System.ServiceModel`. При ініціалізації екземпляру класу `ServiceHost`, створюється служба, за допомогою якої здійснюється комунікація на більш низьких рівнях. Екземпляр класу `ServiceHost` ініціалізується інформацією про тип послуги, одну або кілька кінцевих точок служби, необов'язкові базові адреси та сервіси, які регулюють те, як платформа обробляє запити до служби.

Комунікація між додатками відбувається за допомогою платформи WCF. Основні вимоги, які виконує WCF – це реалізація комунікації додатку як в клієнтському, так і в серверному режимі, тобто додаток не вимагає сторонніх програм для комунікації. Перше обмеження, яке з'являється при такій архітектурі - додатки можуть спілкуватися лише в тому разі, якщо вони знаходяться в одній мережі і мають доступ один до одного.

Для комунікації було спроектовано та реалізовано 3 класи (рис. 2.3):

1. Клас `SharingService` – відповідає за комунікацію серверної частини. Містить список екземплярів класу `SharingServiceCallback`, тобто список приєднаних клієнтів, за допомогою яких сервер може відправляти оновлення для клієнтів. В класі реалізовано метод `ConnectToSharedProject`, який збирає всі дані, які необхідні для створення проекту та відправляє їх до клієнту. Клієнт в свою чергу на основі отриманих даних створює та заповнює проект.
2. Клас `SharingServiceClient` – відповідає за комунікацію клієнтської частини. Використовується для віддаленого визову методів серверу.
3. Клас `SharingServiceCallback` – інкапсулює дані про клієнту. Містить методи для зворотного виклику на сторону серверу. Містить метод `LoadProjectAsync`, за допомогою якого відправляється оновлення проекту для всіх приєднаних клієнтів.

Висновки до розділу 2

Симулятор комп'ютерних мереж реалізує функціональність для проектування та дослідження роботи мережевих пристроїв. В цьому розділі представлено варіанти використання для користувача мета та функціональні вимоги. Також в розділі описано структуру та архітектуру системи.

В розділі описано класи, в яких реалізовано функціональність для взаємодії з користувачем. Описано класи для роботи внутрішньої логіки додатку, а саме роботи пристроїв та протоколів. Також описано класи для взаємодії додатків один з одним.

Крім цього були спроектовані алгоритми для комунікації пристроїв та роботи протоколів. Були представлені деякі теоретичні відомості мережевих властивостей.

Була спроектована та описана структура частини системи, яка відповідає за комунікації між додатками.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИМУЛЯТОРОМ КОМП'ЮТЕРНИХ МЕРЕЖ

3.1. Порядок встановлення та налаштування системи.

Настільний додаток симулятору комп'ютерних мереж – це графічний додаток для настільних комп'ютерних операційних систем з використанням платформи .NET. Для того, щоб він функціонував як настільний додаток використовується платформа WPF, що включає в себе різноманітні візуальні компоненти. Для комунікації з іншими додатками використовується платформа WCF. Грунтуючись на тому, що в проекті використовується .NET Framework 4.7.2, підведемо мінімальні вимоги.

Мінімальні вимоги до системи:

- Процесор 1 ГГц;
- ОЗУ 512 МБ;
- Мінімальне місце на диску 120МБ або 4.5 ГБ у разі необхідності встановлення платформи .NET Framework 4.7;
- Платформа .NET Framework 4.7;
- Графічний пристрій з підтримкою DirectX 9.

Підтримувані клієнтські операційні системи:

- Windows 10;
- Windows 8,1;
- Windows 8;
- Windows 7 SP1.

Для роботи з додатком знадобиться встановлена платформа .NET Framework 4.7. Платформа .NET вимагає права адміністратора для встановлення.

3.2. Структура інтерфейсу та порядок роботи з системою.

Після завантаження додатку, з'являється вікно, на якому є меню для взаємодії користувача з додатком та простір для побудови комп'ютерної мережі та взаємодії з нею (рис. 3.1).

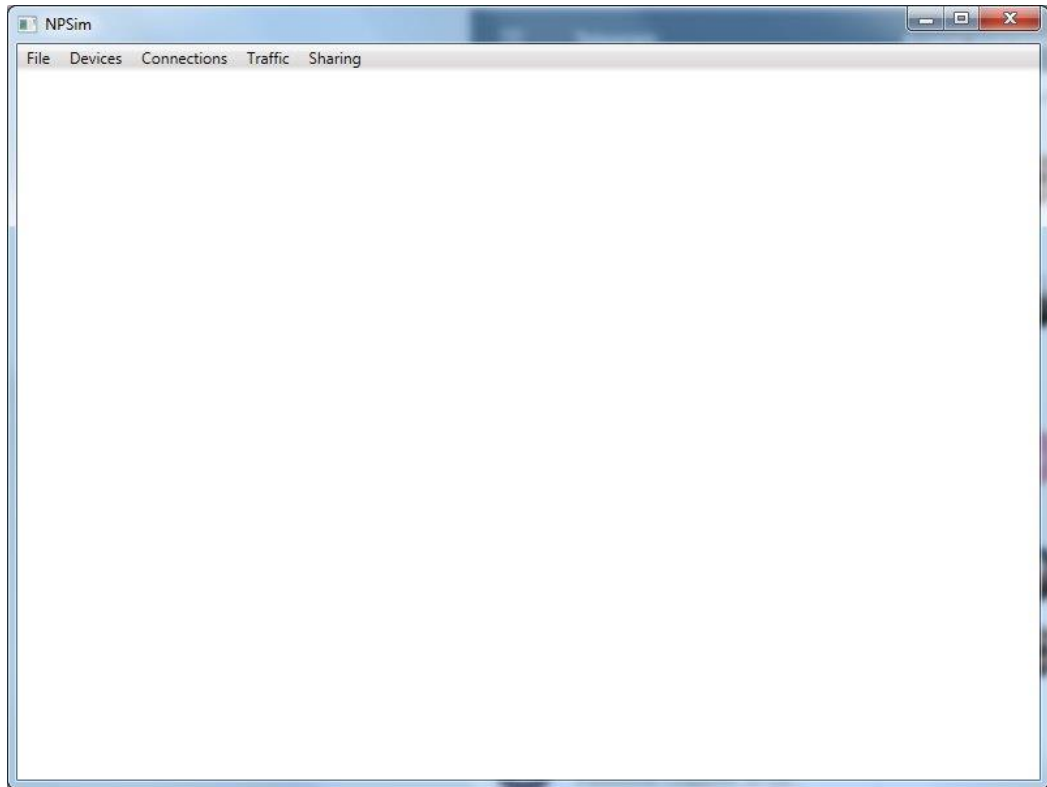


Рис. 3.1. Головне вікно додатку

Для створення проекту мережі, в якому можна буде будувати мережу, треба натиснути на пункт «File» - «Create project» (рис. 3.2).



Рис. 3.2. Пункт меню «File» - «Create project»

Після створення проекту, користувач може додавати пристрої, натиснувши на пункт «Devices» та обравши один з варіантів «Computer», «Router» або «Switch» (рис. 3.3).

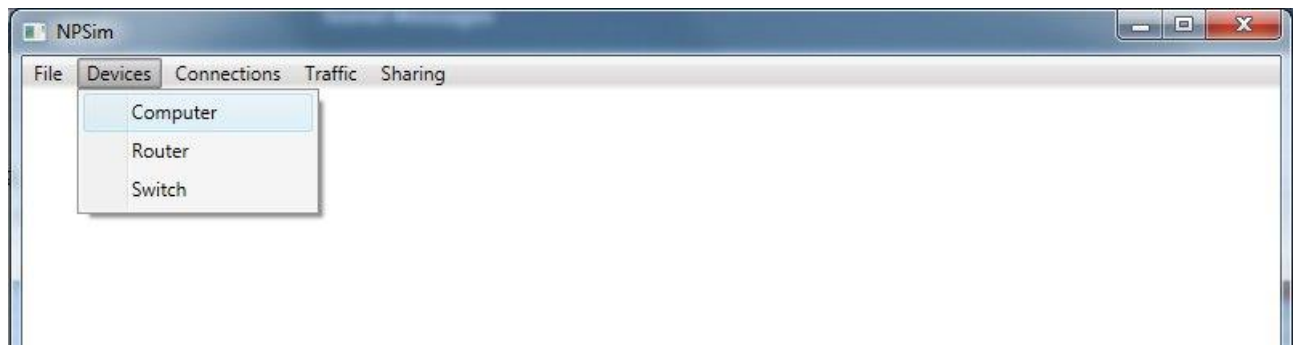


Рис. 3.3. Пункт меню «Devices» для створення пристрою в проєкті

Створивши як мінімум 2 пристрої, користувач може з'єднати їх за допомогою «Connections» - «PhysicalMedia». Натиснувши на пункт «PhysicalMedia» користувач повинен натиснути на 2 пристрої, які він хоче з'єднати (рис. 3.4).

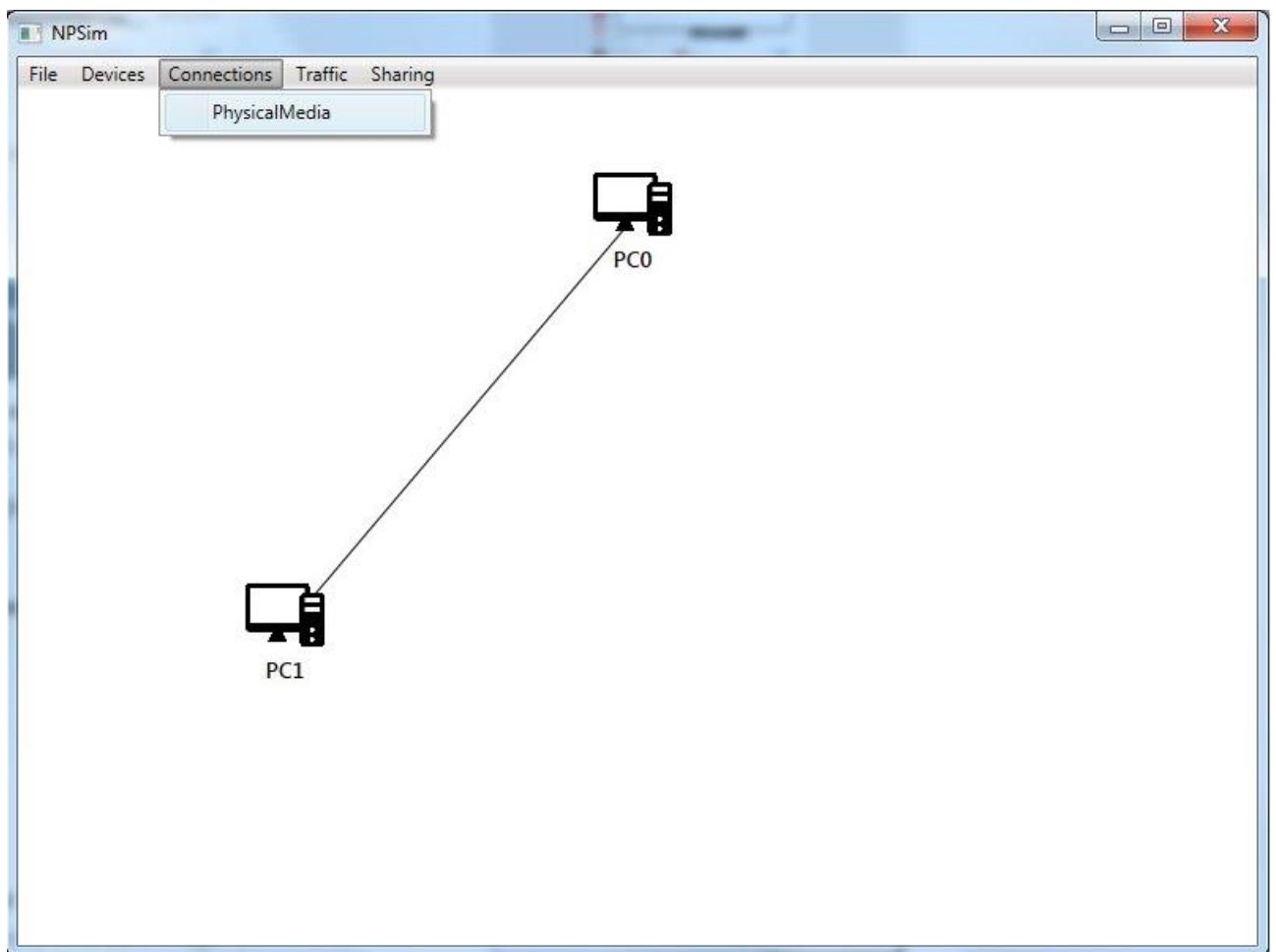


Рис. 3.4. З'єднання пристроїв

Для створення трафіку, необхідно відкрити вікно створення трафіку за допомогою «Traffic» - «Send data» (рис. 3.5). У вікні треба обрати пристрій з

якого відправляться дані, пристрій до якого вони повинні будуть дійти та протокол якого типу буде інкапсульований (рис. 3.6).

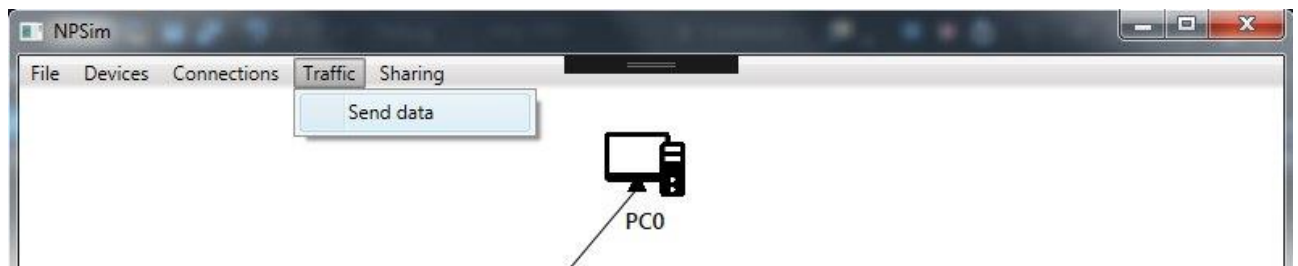


Рис. 3.5. Пункт меню «Traffic» - «Send data» для відкриття вікна створення трафіку

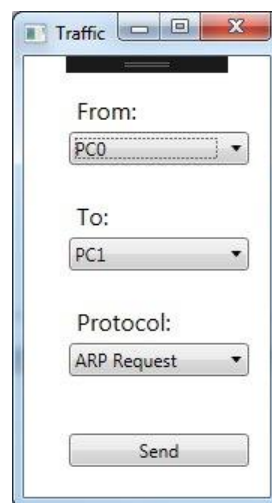


Рис. 3.6. Вікно створення трафіку

Для комунікації з іншими додатками користувач може відкрити доступ до проекту за допомогою пункту «Sharing» - «Open Access», щоб інші користувачі могли приєднатись до його проекту. Для приєднання до чужого проекту, користувач повинен обрати пункт «Sharing» - «Connect to shared project» (рис. 3.7).

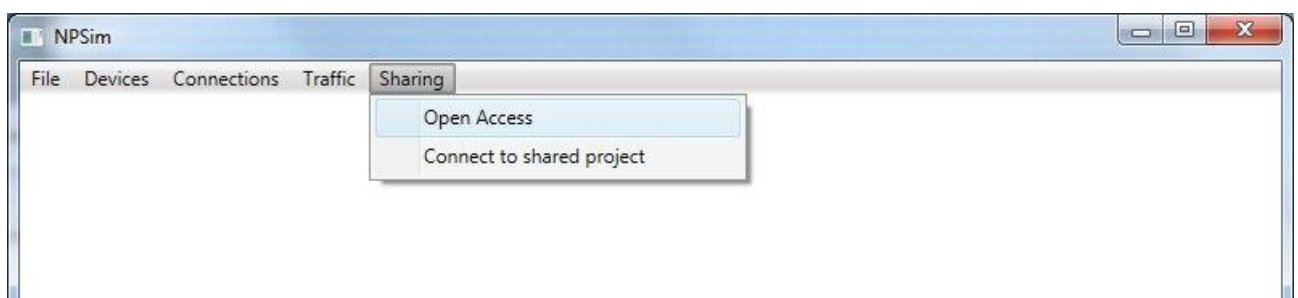


Рис. 3.7. Пункти меню для комунікації між додатками

Крім меню, користувач може взаємодіяти з пристроями в головному вікні. Користувач може викликати контекстне меню пристрою, натиснувши правою

клавішою миші на ньому. В контекстному меню є два варіанти для вибору – «Configuration», який відкриває вікно налаштування пристрою та «Log», який відкриває вікно мережевого логу пристрою (рис. 3.8).

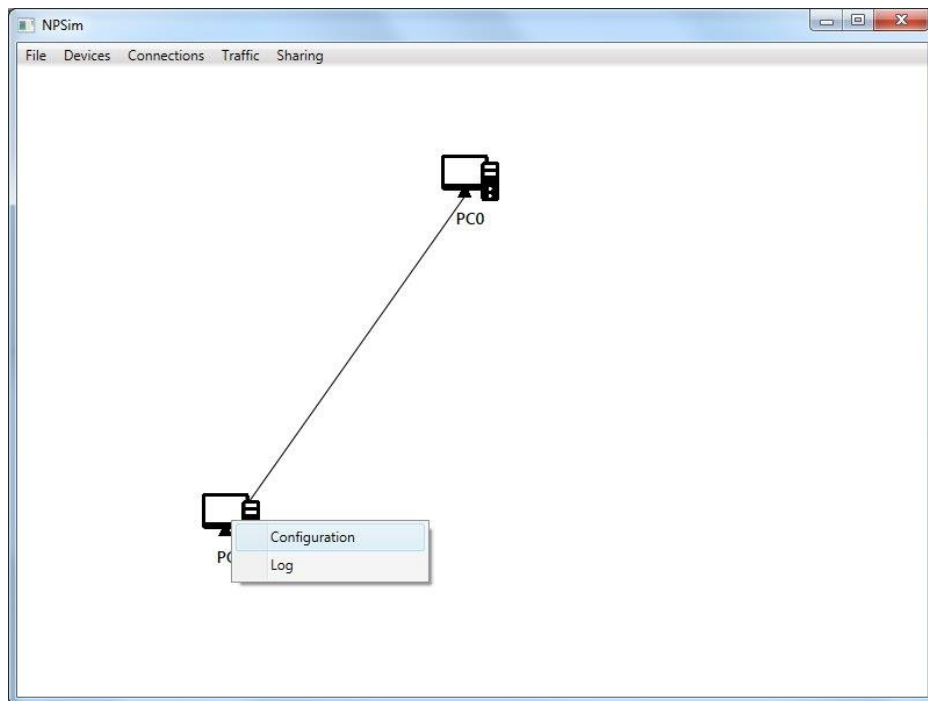


Рис. 3.8. Контекстне меню пристрою

Відкривши вікно налаштування пристрою, користувач може побачити статичні налаштування пристрою та його мережеві інтерфейси. Користувач може редагувати IP-адресу мережевого інтерфейсу (рис. 3.9).

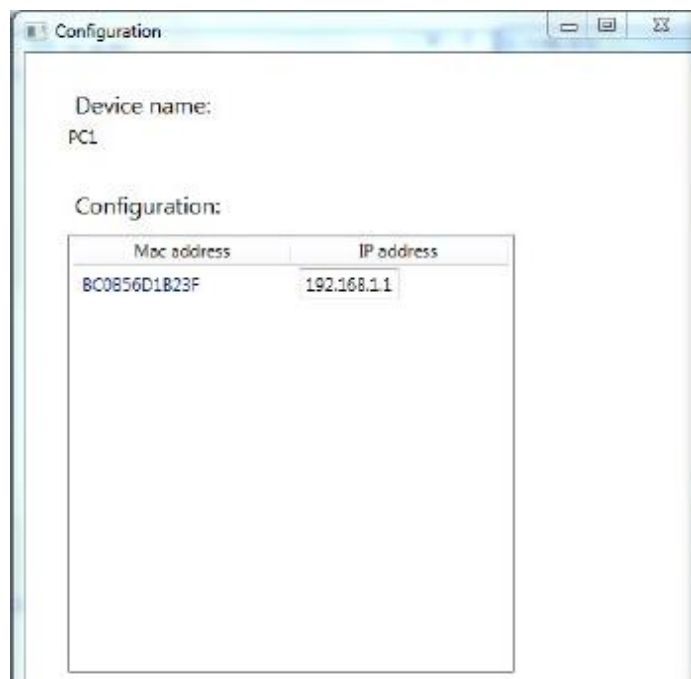


Рис. 3.9. Вікно налаштування пристрою

У вікні мережевого логу пристрою зображено дії, які відбувалися в контексті пристрою (рис. 3.10).

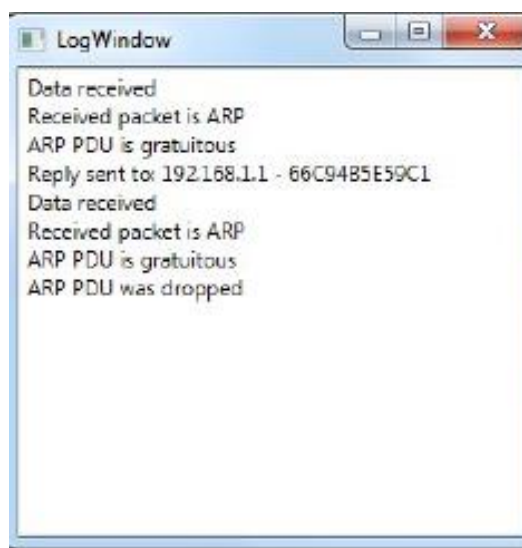


Рис. 3.10. Вікно мережевого логу пристрою PC0

3.3. Дослідження роботи системи.

Для дослідження можна виділити основні процеси:

- Створення пристроїв;
- З'єднання пристроїв в мережу;
- Налаштування пристроїв;
- Відправка даних між пристроями;
- Підключення додатку до іншого додатку.

Створивши пристрої, з'єднавши їх в мережу та налаштувавши їх так, щоб в них були однакові IP-адреси, відправляються ARP Gratuitous пакети, за допомогою чого пристрої дізнаються про конфлікт IP-адреси (рис. 3.11).

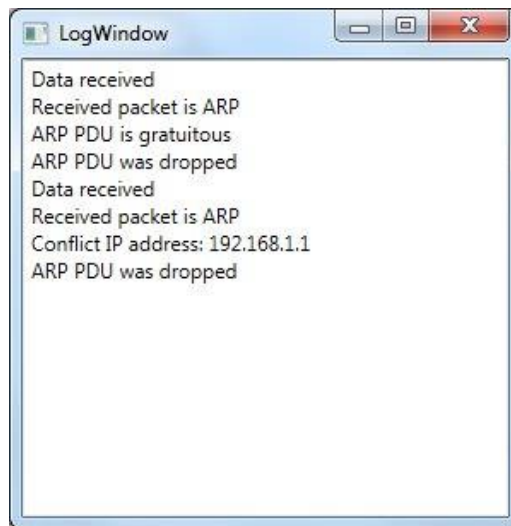


Рис. 3.11. Вікно мережевого логу пристрою PC1

На рис. 3.12 зображено додаток, який виступає в ролі серверу. В додатку було створено проект та додано 2 комп'ютери, які були з'єднані в мережу. На рис. 3.13 зображено додаток, який виступає в ролі клієнту та було приєднано до додатку, зображеного на рис. 3.12. Після приєднання до додатку-серверу, додаток-клієнт отримує дані про проект та відтворює цей проект в просторі побудови мережі. Також додаток-клієнт, отримавши дані, може прослідити дії, які відбувалися в пристрої додатку-серверу за допомогою логу.

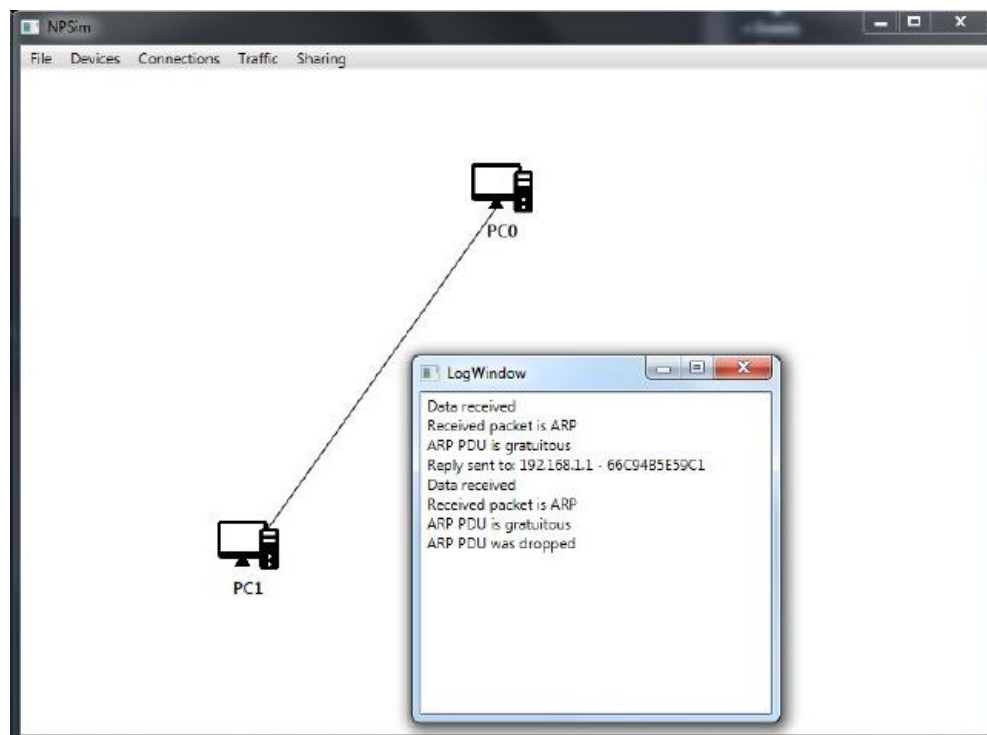


Рис. 3.12. Додаток, який виступає в ролі серверу

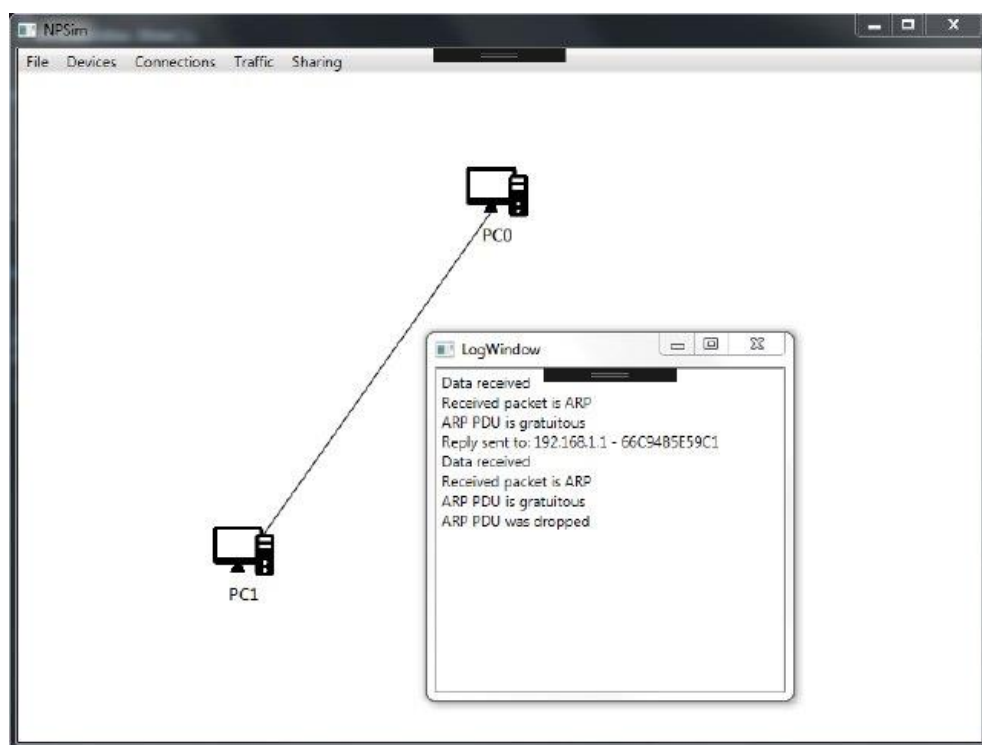


Рис. 3.12. Додаток, який виступає в ролі клієнту

Висновки до розділу 3

В даному розділі визначено мінімальні вимоги до системи для повноцінної роботи програми.

Було описано структуру інтерфейсу та ключові кроки взаємодії з інтерфейсом. Було описано головне вікно та його елементи, вікно створення трафіку, вікно налаштування пристрою та вікно логу пристрою.

Крім цього в даному розділі описано роботу з додатком, а саме процеси створення трафіку, перегляду логу та комунікації між пристроями.

ВИСНОВКИ

В першому розділі була представлена мета, проаналізовані завдання та методи їх вирішення, та сформовано план реалізації додатку. Були проаналізовані програмні аналоги та представлені їх переваги та недоліки. Також було визначено необхідні засоби та структурні елементи, які необхідні для реалізації додатку. Крім цього були наведені теоретичні відомості про мережеві властивості пристроїв та протоколів.

В другому розділі були сформовані варіанти використання для користувача та функціональні вимоги. Було спроектовано та розроблено структуру програмного коду та основну архітектуру системи. В розділі описано класи для взаємодії з користувачем та класи для внутрішньої роботи додатку. Описано класи для комунікації між додатками.

В третьому розділі було описано методологію користування створеним додатком. Були описані та показані процеси створення проекту, створення пристроїв, з'єднання їх в мережу, створення трафіку між пристроями, перегляду логів та комунікації між пристроями.

В результаті виконання курсового проекту отримано програмний продукт, який виконує поставлені завдання.

Програмне забезпечення представляє папку, в якій міститься файл з розширенням «exe» та файли сторонніх бібліотек з розширенням «dll».

Спроектоване програмне забезпечення відповідає поставленим вимогам та є достатньо швидкодіючим не займаючи великих ресурсів комп'ютера, на якому воно запущено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cabarkapa, D., Application of Cisco Packet Tracer 6.2 in teaching of advanced computer networks. – 2015. – ResearchGate: 6.
2. Lee, W. (2013). The evolution of the networking skills gap in Asia/Pacific white paper, External Publication of IDC Information and Data Retrieved 18th February [Електронний ресурс] – Режим доступу до ресурсу: http://www.cisco.com/assets/csr/pdf/IDC_Skills_Gap_-_AsiaPacific.pdf
3. Jesin A., Packet Tracer Network Simulator. - 2014. – Preface.
4. Andrew, C. (2020). Getting Started with GNS3 [Електронний ресурс] - Режим доступу до ресурсу: https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html
5. Genevieve W. (2015). Introduction to the C# language and the .NET Framework [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework>
6. JacobsM. (2018). Getstarted with WPF [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/visualstudio/designers/getting-started-with-wpf?view=vs-2019>
7. Aymeric A. (2017). What Is Windows Communication Foundation [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/dotnet/framework/wcf/whats-wcf>
8. JacobsM. (2019). Welcome to the Visual Studio IDE [Електронний ресурс] - Режим доступу до ресурсу: <https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019>
9. Feig R. Computer Networks: The OSI Reference Model [Електронний ресурс] - Режим доступу до ресурсу: <http://www.rad.com/networks/1994/osi/intro.htm>

ДОДАТКИ

					Державний університет «Житомирська політехніка».20.123.14.000 - ПЗ	
						48

Фрагмент лістингу додатку

UnityConfig.cs

```
using System;
using NPSim.Domain.Builders;
using NPSim.Domain.PhysicalLayer;
using NPSim.ViewModels;
using Unity;

namespace NPSim.Composition
{
    public static class UnityConfig
    {
        public static IUnityContainer GetContainer()
        {
            return _container.Value;
        }

        private static readonly Lazy<IUnityContainer> _container = new Lazy<IUnityContainer>(() =>
        {
            var container = Build();
            return container;
        });

        private static IUnityContainer Build()
        {
            var container = new UnityContainer();

            container.RegisterSingleton<IOpenSystemBuilder, OpenSystemBuilder>();
            container.RegisterType<IMediaManager, MediaManager>();
            container.RegisterType<IMediaBuilder, MediaBuilder>();
            container.RegisterType<INetworkInterfaceControllerBuilder, NetworkInterfaceControllerBuilder>();

            #region ViewModels

            container.RegisterSingleton<MainWindowVm>();

            #endregion

            return container;
        }
    }
}
```

```

    }
}
}

```

IMediaBuilder.cs

```
using NPSim.Entities.PhysicalLayer.Media;
```

```

namespace NPSim.Domain.Builders
{
    public interface IMediaBuilder
    {
        PhysicalMedia BuildPhysicalMedia();
    }
}

```

INetworkInterfaceControllerBuilder.cs

```
using NPSim.Entities.PhysicalLayer.Nic;
```

```

namespace NPSim.Domain.Builders
{
    public interface INetworkInterfaceControllerBuilder
    {
        BaseNetworkInterfaceController BuildNetworkInterfaceController();
    }
}

```

IOpenSystemBuilder.cs

```
using NPSim.Entities;
```

```

namespace NPSim.Domain.Builders
{
    public interface IOpenSystemBuilder
    {
        Computer BuildComputer();
    }
}

```

MediaBuilder.cs

```
using NPSim.Entities.PhysicalLayer;
using NPSim.Entities.PhysicalLayer.Media;

namespace NPSim.Domain.Builders
{
    public class MediaBuilder : IMediaBuilder
    {
        public PhysicalMedia BuildPhysicalMedia()
        {
            var physicalMedia = new PhysicalMedia(new Cabel(), new[] { new Connector(), new Connector() });

            return physicalMedia;
        }
    }
}
```

NetworkInterfaceControllerBuilder.cs

```
using System.Collections.Generic;
using NPSim.Entities.PhysicalLayer.Nic;

namespace NPSim.Domain.Builders
{
    public class NetworkInterfaceControllerBuilder : INetworkInterfaceControllerBuilder
    {
        public BaseNetworkInterfaceController BuildNetworkInterfaceController()
        {
            var macAddress1 = PhysicalAddressHelper.GeneratePhysicalAddress();
            var networkInterface1 = new NetworkInterface(macAddress1);

            var nic = new BaseNetworkInterfaceController(new List<INetworkInterface> { networkInterface1 });

            return nic;
        }
    }
}
```

OpenSystemBuilder.cs

```
using NPSim.Entities;
```

```

namespace NPSim.Domain.Builders
{
    public class OpenSystemBuilder : IOpenSystemBuilder
    {
        private uint _computerCount = 0;

        private readonly INetworkInterfaceControllerBuilder _nicBuilder;

        public OpenSystemBuilder(INetworkInterfaceControllerBuilder nicBuilder)
        {
            _nicBuilder = nicBuilder;
        }

        public Computer BuildComputer()
        {
            var nic = _nicBuilder.BuildNetworkInterfaceController();

            var computer = new Computer($"PC{_computerCount}", 2);
            computer.InstallNic(nic);

            _computerCount++;

            return computer;
        }
    }
}

```

ArpProtocolService.cs

```

using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Domain.DataLinkLayer
{
    [DataContract]

```

```

public class ArpProtocolService
{
    //TODO: add clearing after time out
    [DataMember]
    private readonly Dictionary<IPAddress, PhysicalAddress> _arpCache;

    private readonly Logger _logger;

    public ArpProtocolService(Logger logger)
    {
        _logger = logger;

        _arpCache = new Dictionary<IPAddress, PhysicalAddress>();
    }

    public void AddValueToCache(PhysicalAddress physicalAddress, IPAddress ipAddress)
    {
        if (!_arpCache.ContainsKey(ipAddress))
        {
            _arpCache.Add(ipAddress, physicalAddress);
        }
    }

    public void UpdateMacAddressToCache(PhysicalAddress physicalAddress, IPAddress ipAddress)
    {
        _arpCache.Remove(ipAddress);

        AddValueToCache(physicalAddress, ipAddress);
    }

    public PhysicalAddress GetCachedMacAddress(IPAddress ipAddress)
    {
        return _arpCache.SingleOrDefault(p => p.Key == ipAddress).Value;
    }

    public EthernetType2Frame EncapsulateArpPduToFrame(ArpPdu pdu, PhysicalAddress destinationMacAddress)
    {
        return new EthernetType2Frame(destinationMacAddress ?? PhysicalAddressHelper.Broadcast,
            pdu.SenderMacAddress, new EtherType(ProtocolType.Arp), pdu.GetBytes());
    }

    public ArpPdu DecapsulateFrameToArpPdu(EthernetType2Frame frame)

```

```

{
    using (var stream = new MemoryStream(frame.Payload))
    {
        var formatter = new BinaryFormatter();
        return (ArpPdu)formatter.Deserialize(stream);
    }
}

public ArpPdu GenerateGratuitousArpPdu(PhysicalAddress senderMacAddress, IPAddress
senderProtocolAddress)
{
    return new ArpPdu
    (
        HardwareType.Ethernet,
        new EtherType(ProtocolType.IPv4),
        OperationCode.Request,
        senderMacAddress,
        senderProtocolAddress,
        PhysicalAddressHelper.Blank,
        senderProtocolAddress
    );
}

public ArpPdu GenerateArpReply(PhysicalAddress senderMacAddress, IPAddress senderProtocolAddress,
PhysicalAddress destinationMacAddress, IPAddress destinationProtocolAddress)
{
    return new ArpPdu
    (
        HardwareType.Ethernet,
        new EtherType(ProtocolType.IPv4),
        OperationCode.Reply,
        senderMacAddress,
        senderProtocolAddress,
        destinationMacAddress,
        destinationProtocolAddress
    );
}

public ArpPdu GenerateArpRequest(PhysicalAddress senderMacAddress, IPAddress senderProtocolAddress,
IPAddress destinationProtocolAddress)
{
    return new ArpPdu

```

```

        (
            HardwareType.Ethernet,
            new EtherType(ProtocolType.Ipv4),
            OperationCode.Request,
            senderMacAddress,
            senderProtocolAddress,
            PhysicalAddressHelper.Blank,
            destinationProtocolAddress
        );
    }
}

```

DataLinkLayerService.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Text;
using System.Threading.Tasks;
using NPSim.Entities;
using NPSim.Entities.PhysicalLayer.Nic;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Domain.DataLinkLayer
{
    [DataContract(IsReference = true)]
    public class DataLinkLayerService : IDataLinkLayerService
    {
        [DataMember]
        private readonly IOpenSystem _openSystem;

        [DataMember]
        private readonly ArpProtocolService _arpProtocolService;

        private readonly Logger _logger;

        public DataLinkLayerService(IOpenSystem openSystem, Logger logger)

```

```

{
    _openSystem = openSystem;

    _logger = logger;

    _arpProtocolService = new ArpProtocolService(logger);
}

public PhysicalAddress GetPhysicalAddressByIpAddress(INetworkInterface networkInterface, IPAddress
ipAddress)
{
    var cachedIpAddress = _arpProtocolService.GetCachedMacAddress(ipAddress);
    if (cachedIpAddress == null)
    {
        var arpPdu = _arpProtocolService.GenerateArpRequest(networkInterface.MacAddress,
networkInterface.IpAddress, ipAddress);

        var frame = _arpProtocolService.EncapsulateArpPduToFrame(arpPdu, PhysicalAddressHelper.Broadcast);

        networkInterface.Send(frame.GetBytes());
    }

    return null;
}

public EthernetType2Frame GenerateEncapsulatedGratuitousArpPdu(INetworkInterface networkInterface,
IPAddress ipAddress)
{
    var gratuitousArpPdu = _arpProtocolService.GenerateGratuitousArpPdu(networkInterface.MacAddress,
ipAddress);
    var ethernetFrame = _arpProtocolService.EncapsulateArpPduToFrame(gratuitousArpPdu,
PhysicalAddressHelper.Broadcast);

    return ethernetFrame;
}

public void HandleArpPdu(INetworkInterface networkInterface, ArpPdu pdu)
{
    if (pdu.OperationCode == OperationCode.Request &&
pdu.SenderProtocolAddress.Equals(pdu.TargetProtocolAddress)) // Is Gratuitous
    {
        _logger.Write($"ARP PDU is gratuitous");
    }
}

```

```

var cachedSenderId = _arpProtocolService.GetCachedMacAddress(pdu.SenderProtocolAddress);
if (cachedSenderId != null)
{
    _arpProtocolService.UpdateMacAddressToCache(pdu.SenderMacAddress, pdu.SenderProtocolAddress);
    _logger.Write($"ARP table updated: {pdu.SenderProtocolAddress} - {pdu.SenderMacAddress}");

    return;
}

if (!pdu.SenderProtocolAddress.Equals(networkInterface.IpAddress)) // not conflict
{
    _logger.Write($"ARP PDU was dropped");
    return;
}

var arpReply = _arpProtocolService.GenerateArpReply(networkInterface.MacAddress,
networkInterface.IpAddress, pdu.SenderMacAddress, pdu.SenderProtocolAddress);
var ethernetFrame = _arpProtocolService.EncapsulateArpPduToFrame(arpReply,
arpReply.TargetMacAddress);

_logger.Write($"Reply sent to: {arpReply.TargetProtocolAddress} - {arpReply.TargetMacAddress}");
networkInterface.Send(ethernetFrame.GetBytes());
}
else if (pdu.OperationCode == OperationCode.Request)
{
    if (!pdu.SenderProtocolAddress.Equals(networkInterface.IpAddress))
    {
        _arpProtocolService.AddValueToCache(pdu.SenderMacAddress, pdu.SenderProtocolAddress);
        _logger.Write($"ARP cached: {pdu.SenderProtocolAddress} - {pdu.SenderMacAddress}");
    }

    var arpReply = _arpProtocolService.GenerateArpReply(networkInterface.MacAddress,
networkInterface.IpAddress, pdu.SenderMacAddress, pdu.SenderProtocolAddress);
var ethernetFrame = _arpProtocolService.EncapsulateArpPduToFrame(arpReply,
arpReply.TargetMacAddress);

_logger.Write($"Reply sent to: {arpReply.TargetProtocolAddress} - {arpReply.TargetMacAddress}");
networkInterface.Send(ethernetFrame.GetBytes());
}
else if (pdu.OperationCode == OperationCode.Reply)
{

```

```

        if (pdu.SenderProtocolAddress.Equals(networkInterface.IpAddress)) // conflict
        {
            _logger.Write($"Conflict IP address: {pdu.SenderProtocolAddress}");
            _logger.Write($"ARP PDU was dropped");
        }
        else
        {
            _arpProtocolService.AddValueToCache(pdu.SenderMacAddress, pdu.SenderProtocolAddress);
            _logger.Write($"Cached: {pdu.SenderProtocolAddress} - {pdu.SenderMacAddress}");
        }
    }
    else
    {
        throw new ArgumentOutOfRangeException(pdu.OperationCode.ToString());
    }
}
}
}

```

IDataLinkLayerService.cs

```

using System.Net;
using System.Net.NetworkInformation;
using NPSim.Entities.PhysicalLayer.Nic;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Domain.DataLinkLayer
{
    public interface IDataLinkLayerService
    {
        PhysicalAddress GetPhysicalAddressByIpAddress(INetworkInterface networkInterface, IPAddress ipAddress);
        EthernetType2Frame GenerateEncapsulatedGratuitousArpPdu(INetworkInterface networkInterface, IPAddress
ipAddress);
        void HandleArpPdu(INetworkInterface networkInterface, ArpPdu pdu);
    }
}

```

IMediaManager.cs

```

using NPSim.Entities.PhysicalLayer.Nic;
using NPSim.Entities.PhysicalLayer.Media;

```

```
namespace NPSim.Domain.PhysicalLayer
```

```
{
```

```
    public interface IMediaManager
```

```
    {
```

```
        void AttachMediaToConnectionEndpoint(IMedia media, IConnectionEndpoint endpoint);
```

```
        void DetachMediaFromConnectionEndpoint(IMedia media, IConnectionEndpoint endpoint);
```

```
    }
```

```
}
```

MediaManager.cs

```
using NPSim.Entities.PhysicalLayer.Nic;
```

```
using NPSim.Entities.PhysicalLayer.Media;
```

```
namespace NPSim.Domain.PhysicalLayer
```

```
{
```

```
    public class MediaManager : IMediaManager
```

```
    {
```

```
        public void AttachMediaToConnectionEndpoint(IMedia media, IConnectionEndpoint endpoint)
```

```
        {
```

```
            media.AttachTo(endpoint);
```

```
            endpoint.Attach(media);
```

```
        }
```

```
        public void DetachMediaFromConnectionEndpoint(IMedia media, IConnectionEndpoint endpoint)
```

```
        {
```

```
            media.DetachFrom(endpoint);
```

```
            endpoint.Detach();
```

```
        }
```

```
    }
```

```
}
```

IPAddressSurrogate.cs

```
using System.Net;
```

```
using System.Runtime.Serialization;
```

```
namespace NPSim.Domain.Surrogates
```

```
{
```

```
    public class IPAddressSurrogate : ISerializationSurrogate
```

```
    {
```

```
        public void GetObjectData(object obj, SerializationInfo info, StreamingContext context)
```

```

{
    var ipAddress = (IPAddress)obj;
    info.AddValue("AddressBytes", ipAddress.GetAddressBytes());
}

public object SetObjectData(object obj, SerializationInfo info, StreamingContext context, ISurrogateSelector
selector)
{
    var ipAddress = new IPAddress((byte[])info.GetValue("AddressBytes", typeof(byte[])));

    return ipAddress;
}
}

```

PhysicalAddressSurrogate.cs

```

using System.Net.NetworkInformation;
using System.Runtime.Serialization;

namespace NPSim.Domain.Surrogates
{
    public class PhysicalAddressSurrogate : ISerializationSurrogate
    {
        public void GetObjectData(object obj, SerializationInfo info, StreamingContext context)
        {
            var physicalAddress = (PhysicalAddress)obj;
            info.AddValue("AddressBytes", physicalAddress.GetAddressBytes());
        }

        public object SetObjectData(object obj, SerializationInfo info, StreamingContext context, ISurrogateSelector
selector)
        {
            var physicalAddress = new PhysicalAddress((byte[])info.GetValue("AddressBytes", typeof(byte[])));

            return physicalAddress;
        }
    }
}

```

Logger.cs

```
using System.Runtime.Serialization;
using System.Text;

namespace NPSim.Domain
{
    [DataContract]
    public class Logger
    {
        [DataMember]
        private StringBuilder _log = new StringBuilder();

        public string Log { get => _log.ToString(); }

        public Logger Write(string message)
        {
            _log.AppendLine(message);

            return this;
        }
    }
}
```

PhysicalAddressHelper.cs

```
using System;
using System.Net.NetworkInformation;

namespace NPSim.Domain
{
    public static class PhysicalAddressHelper
    {
        public static readonly PhysicalAddress None = PhysicalAddress.None;
        public static readonly PhysicalAddress Broadcast = new PhysicalAddress(new byte[6] { 255, 255, 255, 255, 255, 255 });
        public static readonly PhysicalAddress Blank = new PhysicalAddress(new byte[6] { 0, 0, 0, 0, 0, 0 });

        public static PhysicalAddress GeneratePhysicalAddress()
        {
            var random = new Random();
            var octets = new byte[6];
```

```

        random.NextBytes(octets);

        return new PhysicalAddress(octets);
    }
}

```

Cabel.cs

```

namespace NPSim.Entities.PhysicalLayer
{
    public class Cabel
    {
        //TODO: add media transmit simulation
        public byte[] ProcessTransmition(byte[] data)
        {
            return data;
        }
    }
}

```

Connector.cs

```

using System.Runtime.Serialization;
using NPSim.Entities.PhysicalLayer.Nic;

namespace NPSim.Entities.PhysicalLayer.Media
{
    [DataContract(IsReference = true)]
    [KnownType(typeof(NetworkInterface))]
    public class Connector
    {
        public bool IsAttached { get => ConnectionEndpoint != null; }

        [DataMember]
        public IConnectionEndpoint ConnectionEndpoint { get; private set; }

        public void Attach(IConnectionEndpoint connectionEndpoint)
        {
            ConnectionEndpoint = connectionEndpoint;
        }
    }
}

```

```

        public void Detach()
        {
            ConnectionEndpoint = null;
        }
    }
}

```

IMedia.cs

```

using NPSim.Entities.PhysicalLayer.Nic;

namespace NPSim.Entities.PhysicalLayer.Media
{
    public interface IMedia
    {
        void AttachTo(IConnectionEndpoint endpoint);
        void DetachFrom(IConnectionEndpoint endpoint);
        void Transmit(INetworkInterface networkInterface, byte[] data);
    }
}

```

PhysicalMedia.cs

```

using System.Linq;
using System.Runtime.Serialization;
using NPSim.Entities.PhysicalLayer.Nic;

namespace NPSim.Entities.PhysicalLayer.Media
{
    [DataContract(IsReference = true)]
    [KnownType(typeof(PhysicalMedia))]
    public class PhysicalMedia : IMedia
    {
        private const uint _connectorCount = 2;

        public bool HasAvailableConnector { get => _connectors.Any(c => !c.IsAttached); }

        [DataMember]
        private readonly Cabel _cabel;

        [DataMember]
        private readonly Connector[] _connectors;
    }
}

```

```

public PhysicalMedia(Cabel cabel, Connector[] connectors)
{
    _cabel = cabel;
    _connectors = connectors;
}

public void AttachTo(IConnectionEndpoint endpoint)
{
    var availableConnector = _connectors.FirstOrDefault(c => !c.IsAttached);
    availableConnector.Attach(endpoint);
}

public void DetachFrom(IConnectionEndpoint endpoint)
{
    var connector = _connectors.SingleOrDefault(c => c.ConnectionEndpoint == endpoint);
    connector.Detach();
}

public void Transmit(INetworkInterface sender, byte[] data)
{
    data = ProcessTransmition(data);
    var dataReceivers = _connectors.Where(c => c.ConnectionEndpoint != sender).Select(c =>
c.ConnectionEndpoint);
    foreach (INetworkInterface receiver in dataReceivers)
    {
        receiver.EnqueueData(data);
    }
}

private byte[] ProcessTransmition(byte[] data)
{
    return _cabel.ProcessTransmition(data);
}
}

```

BaseNetworkInterfaceController.cs

```

using System;
using System.Collections.Generic;
using System.IO;

```

```

using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Threading.Tasks;
using NPSim.Domain;
using NPSim.Domain.DataLinkLayer;
using NPSim.Domain.Surrogates;
using NPSim.Entities.PhysicalLayer.Nic.EventHandlers;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Entities.PhysicalLayer.Nic
{
    [DataContract]
    [KnownType(typeof(NetworkInterface))]
    [KnownType(typeof(EventHandler<DataReceivedEventArgs>))]
    [KnownType(typeof(DataLinkLayerService))]
    public class BaseNetworkInterfaceController : INetworkInterfaceController
    {
        [DataMember]
        private readonly List<INetworkInterface> _networkInterfaces;

        [DataMember]
        public uint NetworkInterfaceCount { get; private set; }

        public IReadOnlyList<INetworkInterface> NetworkInterfaces { get => _networkInterfaces; }

        [DataMember]
        public IDataLinkLayerService DataLinkLayerService { get; set; }

        public Logger Logger { get; set; }

        public BaseNetworkInterfaceController(List<INetworkInterface> networkInterfaces)
        {
            _networkInterfaces = networkInterfaces;

            foreach (var networkInterface in NetworkInterfaces)
            {
                networkInterface.ParentNic = this;
            }
        }
    }
}

```

```

public void ChangeIpAddress(INetworkInterface networkInterface, IPAddress ipAddress)
{
    networkInterface.ChangeIpAddress(ipAddress);

    var ethernetFrame = DataLinkLayerService.GenerateEncapsulatedGratuitousArpPdu(networkInterface,
ipAddress);
    Send(ethernetFrame);

    //DataLinkLayerService.GetPhysicalAddressByIpAddress(networkInterface, IPAddress.Parse("192.168.1.1"));
}

public void Send(EthernetType2Frame frame)
{
    var sender = NetworkInterfaces.FirstOrDefault(n => n.MacAddress.Equals(frame.SourceMacAddress));

    sender.Send(frame.GetBytes());
}

public void ProcessDataReceiving(object sender, DataReceivedEventArgs e)
{
    var networkInterface = (INetworkInterface)sender;
    var receivedData = networkInterface.DequeueData();
    if (receivedData != null)
    {
        Logger.Write("Data received");

        EthernetType2Frame frame = null;
        using (var stream = new MemoryStream(receivedData))
        {
            var surrogateSelector = new SurrogateSelector();
            surrogateSelector.AddSurrogate(typeof(PhysicalAddress), new
StreamingContext(StreamingContextStates.All), new PhysicalAddressSurrogate());
            var formatter = new BinaryFormatter()
            {
                SurrogateSelector = surrogateSelector
            };
            frame = (EthernetType2Frame)formatter.Deserialize(stream);
        }

        if (frame.DestinationMacAddress.Equals(networkInterface.MacAddress) ||
frame.DestinationMacAddress.Equals(PhysicalAddressHelper.Broadcast))

```


INetworkInterface.cs

```
using System.Net;
using System.Net.NetworkInformation;
using NPSim.Domain;
using NPSim.Entities.PhysicalLayer.Media;

namespace NPSim.Entities.PhysicalLayer.Nic
{
    public interface INetworkInterface : IConnectionEndpoint
    {
        PhysicalAddress MacAddress { get; }
        IPAddress IpAddress { get; }
        bool IsActivated { get; }
        IMedia AttachedMedia { get; }

        INetworkInterfaceController ParentNic { get; set; }
        Logger Logger { get; set; }

        void ChangeIpAddress(IPAddress ipAddress);

        void Activate();
        void Deactivate();
        void Send(byte[] data);
        void EnqueueData(byte[] data);
        byte[] DequeueData();
    }
}
```

INetworkInterfaceController.cs

```
using System.Collections.Generic;
using System.Net;
using NPSim.Domain;
using NPSim.Domain.DataLinkLayer;
using NPSim.Entities.PhysicalLayer.Nic.EventHandlers;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Entities.PhysicalLayer.Nic
{
    public interface INetworkInterfaceController
    {

```

```

        IReadOnlyList<INetworkInterface> NetworkInterfaces { get; }
        IDataLinkLayerService DataLinkLayerService { get; set; }
        Logger Logger { get; set; }

        void ChangeIpAddress(INetworkInterface networkInterface, IPAddress ipAddress);
        void Send(EthernetType2Frame frame);
        void ProcessDataReceiving(object sender, DataReceivedEventArgs e);
    }
}

```

NetworkInterface.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Threading;
using NPSim.Domain;
using NPSim.Entities.PhysicalLayer.Media;
using NPSim.Entities.PhysicalLayer.Nic.EventHandlers;

namespace NPSim.Entities.PhysicalLayer.Nic
{
    [DataContract(IsReference = true)]
    [KnownType(typeof(PhysicalMedia))]
    [KnownType(typeof(EventHandler<DataReceivedEventArgs>))]
    public class NetworkInterface : INetworkInterface
    {
        [DataMember]
        private readonly byte[] _macAddressBytes;
        private readonly PhysicalAddress _macAddress;

        [DataMember]
        private byte[] _ipAddressBytes;
        private IPAddress _ipAddress;

        public PhysicalAddress MacAddress { get => _macAddress ?? new PhysicalAddress(_macAddressBytes); }
        public IPAddress IpAddress { get => _ipAddress ?? new IPAddress(_ipAddressBytes); }

        [DataMember]

```

```

public bool IsActivated { get; private set; }

[DataMember]
public IMedia AttachedMedia { get; private set; }

[DataMember]
public INetworkInterfaceController ParentNic { get; set; }

[DataMember]
private readonly Queue<byte[]> QueuedData = new Queue<byte[]>();

public Logger Logger { get; set; }

public NetworkInterface(PhysicalAddress macAddress)
{
    _macAddress = macAddress;
    _macAddressBytes = MacAddress.GetAddressBytes();

    _ipAddress = IPAddress.None;
    _ipAddressBytes = IpAddress.GetAddressBytes();
}

public void ChangeIpAddress(IPAddress ipAddress)
{
    _ipAddress = ipAddress;
    _ipAddressBytes = IpAddress.GetAddressBytes();
}

public void Attach(IMedia media)
{
    if (AttachedMedia == null)
    {
        AttachedMedia = media;
    }
}

public void Detach()
{
    if (AttachedMedia != null)
    {
        AttachedMedia = null;
    }
}

```

```

    }

    public void Activate()
    {
        IsActivated = true;
    }

    public void Deactivate()
    {
        IsActivated = false;
    }

    public void Send(byte[] data)
    {
        AttachedMedia.Transmit(this, data);
    }

    public void EnqueueData(byte[] data)
    {
        QueuedData.Enqueue(data);

        var e = new DataReceivedEventArgs();
        OnDataReceived(e);
    }

    public byte[] DequeueData()
    {
        if (QueuedData.Any())
        {
            return QueuedData.Dequeue();
        }

        return null;
    }

    protected void OnDataReceived(DataReceivedEventArgs e)
    {
        ParentNic.ProcessDataReceiving(this, e);
    }
}

```

ArpPdu.cs

```
using System;
using System.IO;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using NPSim.Domain.Surrogates;

namespace NPSim.Entities.ProtocolDataUnits
{
    [Serializable]
    public class ArpPdu
    {
        public HardwareType HardwareType { get; }
        public EtherType EtherType { get; }
        public int HardwareAddressLength { get; }
        public int ProtocolAddressLength { get; }
        public OperationCode OperationCode { get; }
        public PhysicalAddress SenderMacAddress { get; }
        public IPAddress SenderProtocolAddress { get; }
        public PhysicalAddress TargetMacAddress { get; }
        public IPAddress TargetProtocolAddress { get; }

        public ArpPdu
        (
            HardwareType hardwareType,
            EtherType etherType,
            OperationCode operationCode,
            PhysicalAddress senderMacAddress,
            IPAddress senderProtocolAddress,
            PhysicalAddress targetMacAddress,
            IPAddress targetProtocolAddress
        )
        {
            HardwareType = hardwareType;
            EtherType = etherType;
            OperationCode = operationCode;
            SenderMacAddress = senderMacAddress;
            SenderProtocolAddress = senderProtocolAddress;
            TargetMacAddress = targetMacAddress;
        }
    }
}
```

```

        TargetProtocolAddress = targetProtocolAddress;
    }

    public byte[] GetBytes()
    {
        using (var stream = new MemoryStream())
        {
            var surrogateSelector = new SurrogateSelector();
            surrogateSelector.AddSurrogate(typeof(PhysicalAddress), new
StreamingContext(StreamingContextStates.All), new PhysicalAddressSurrogate());
            surrogateSelector.AddSurrogate(typeof(IPAddress), new StreamingContext(StreamingContextStates.All),
new IPAddressSurrogate());
            var formatter = new BinaryFormatter()
            {
                SurrogateSelector = surrogateSelector
            };
            formatter.Serialize(stream, this);
            return stream.ToArray();
        }
    }
}

```

EthernetType2Frame.cs

```

using System;
using System.IO;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using NPSim.Domain.Surrogates;

namespace NPSim.Entities.ProtocolDataUnits
{
    [Serializable]
    public class EthernetType2Frame
    {
        private readonly byte[] _preamble = new byte[] { 170, 170, 170, 170, 170, 170, 170 }; // 170 = 10101010
        private readonly byte[] _startOfFrameDelimiter = new byte[] { 171 }; // 171 = 10101011

        public PhysicalAddress DestinationMacAddress { get; }
    }

```

```

public PhysicalAddress SourceMacAddress { get; }
public EtherType EtherType { get; }
public byte[] Payload { get; }
public byte[] FrameCheckSequence { get; }

public EthernetType2Frame(PhysicalAddress destinationMacAddress, PhysicalAddress sourceMacAddress,
EtherType etherType, byte[] payload)
{
    DestinationMacAddress = destinationMacAddress;
    SourceMacAddress = sourceMacAddress;
    EtherType = etherType;
    Payload = payload;
}

public byte[] GetBytes()
{
    using (var stream = new MemoryStream())
    {
        var surrogateSelector = new SurrogateSelector();
        surrogateSelector.AddSurrogate(typeof(PhysicalAddress), new
StreamingContext(StreamingContextStates.All), new PhysicalAddressSurrogate());
        var formatter = new BinaryFormatter()
        {
            SurrogateSelector = surrogateSelector
        };
        formatter.Serialize(stream, this);
        return stream.ToArray();
    }
}
}

```

EtherType.cs

```

using System;

namespace NPSim.Entities.ProtocolDataUnits
{
    [Serializable]
    public class EtherType
    {
        public ProtocolType ProtocolType { get; }
    }
}

```

```

        public int Length { get; }

        public EtherType(ProtocolType protocolType)
        {
            ProtocolType = protocolType;
        }

        public EtherType(int length)
        {
            Length = length;
        }
    }
}

```

HardwareType.cs

```

namespace NPSim.Entities.ProtocolDataUnits
{
    public enum HardwareType
    {
        Ethernet = 0x0001
    }
}

```

BaseOpenSystem.cs

```

using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.NetworkInformation;
using System.Runtime.Serialization;
using NPSim.Domain;
using NPSim.Domain.DataLinkLayer;
using NPSim.Entities.PhysicalLayer.Nic;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Entities
{
    [DataContract]
    [KnownType(typeof(Computer))]
    [KnownType(typeof(BaseNetworkInterfaceController))]
    [KnownType(typeof(DataLinkLayerService))]
}

```

```

public abstract class BaseOpenSystem : IOpenSystem
{
    [DataMember]
    public bool IsActive { get; private set; }

    [DataMember]
    public string Name { get; private set; }

    [DataMember]
    public uint NicCount { get; private set; }

    [DataMember]
    public List<INetworkInterfaceController> NicCollection { get; private set; }

    [DataMember]
    public IDataLinkLayerService DataLinkLayerService { get; private set; }

    [DataMember]
    public Logger Logger { get; private set; }

    public BaseOpenSystem(string name, uint nicCount)
    {
        Name = name;
        NicCount = nicCount;
        NicCollection = new List<INetworkInterfaceController>();
        Logger = new Logger();
        DataLinkLayerService = new DataLinkLayerService(this, Logger);
    }

    public void InstallNic(INetworkInterfaceController nic)
    {
        if (NicCollection.Contains(nic))
        {
            return;
        }

        nic.DataLinkLayerService = DataLinkLayerService;
        nic.Logger = Logger;
        NicCollection.Add(nic);

        foreach (var networkInterface in nic.NetworkInterfaces)
        {

```

```

        networkInterface.Logger = Logger;
    }
}

public void UninstallNic(INetworkInterfaceController nic)
{
    NicCollection.Remove(nic);
}

public void ChangeNetworkInterfaceConfiguration(INetworkInterface networkInterface, IPAddress ipAddress)
{
    networkInterface.ChangeIpAddress(ipAddress);

    var ethernetFrame = DataLinkLayerService.GenerateEncapsulatedGratuitousArpPdu(networkInterface,
ipAddress);
    SendData(ethernetFrame);
}

public void SendData(EthernetType2Frame frame)
{
    var nic = NicCollection.FirstOrDefault(c => c.NetworkInterfaces.Any(n =>
n.MacAddress.Equals(frame.SourceMacAddress)));

    nic.Send(frame);
}

public IEnumerable<INetworkInterface> GetAvailableNetworkInterfaces()
{
    if (NicCollection.Any())
    {
        return NicCollection.SelectMany(c => c.NetworkInterfaces.Where(n => n.AttachedMedia == null));
    }

    return Enumerable.Empty<INetworkInterface>();
}
}

```

IOpenSystem.cs

```

using System.Collections.Generic;
using System.Net.NetworkInformation;

```

```

using NPSim.Entities.PhysicalLayer.Nic;
using NPSim.Entities.ProtocolDataUnits;

namespace NPSim.Entities
{
    public interface IOpenSystem
    {
        void InstallNic(INetworkInterfaceController nic);
        void UninstallNic(INetworkInterfaceController nic);
        void SendData(EthernetType2Frame frame);
        IEnumerable<INetworkInterface> GetAvailableNetworkInterfaces();
    }
}

```

NetworkProjectModel.cs

```

using System.Collections.ObjectModel;
using System.Runtime.Serialization;

namespace NPSim.Models
{
    [DataContract]
    public class NetworkProjectModel
    {
        [DataMember]
        public ObservableCollection<OpenSystemModel> OpenSystemModels { get; set; }

        [DataMember]
        public ObservableCollection<PhysicalMediaModel> PhysicalMediaModels { get; set; }

        public NetworkProjectModel(ObservableCollection<OpenSystemModel> openSystemModels,
ObservableCollection<PhysicalMediaModel> physicalMediaModels)
        {
            OpenSystemModels = openSystemModels;
            PhysicalMediaModels = physicalMediaModels;
        }
    }
}

```

OpenSystemModel.cs

```

using System.Runtime.Serialization;

```

```

using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using NPSim.Entities;

namespace NPSim.Models
{
    [DataContract]
    public class OpenSystemModel
    {
        private Point _offset;

        [DataMember]
        public BaseOpenSystem OpenSystem { get; set; }

        [DataMember]
        public Point Position { get; private set; }

        public string DisplayName { get => OpenSystem.Name; }

        public event MouseButtonEventHandler MouseLeftButtonDown;
        public event MouseButtonEventHandler MouseLeftButtonUp;
        public event MouseEventHandler MouseMove;

        public OpenSystemModel(BaseOpenSystem openSystem)
        {
            OpenSystem = openSystem;
        }

        public void OpenSystemUiElement_MouseDown(object sender, MouseButtonEventArgs e)
        {
            var openSystemUiElement = (UIElement)sender;
            _offset = e.GetPosition(openSystemUiElement);
            openSystemUiElement.CaptureMouse();

            MouseLeftButtonDown?.Invoke(sender, e);
        }

        public void OpenSystemUiElement_MouseUp(object sender, MouseButtonEventArgs e)
        {
            var openSystemUiElement = (UIElement)sender;

```

```

        openSystemUiElement.ReleaseMouseCapture();

        MouseLeftButtonUp?.Invoke(sender, e);
    }

    public void OpenSystemUiElement_MouseMove(object sender, MouseEventArgs e)
    {
        if (e.LeftButton == MouseButtonState.Pressed && Mouse.Captured == sender)
        {
            var openSystemUiElement = (UIElement)sender;
            var simulationCanvas = (Canvas)VisualTreeHelper.GetParent(openSystemUiElement);
            var currentPosition = e.GetPosition(simulationCanvas);

            Position = new Point(currentPosition.X - _offset.X, currentPosition.Y - _offset.Y);

            Canvas.SetLeft(openSystemUiElement, Position.X);
            Canvas.SetTop(openSystemUiElement, Position.Y);
        }

        MouseMove?.Invoke(sender, e);
    }
}

```

PhysicalMediaModel.cs

```

using System;
using System.ComponentModel;
using System.Linq;
using System.Runtime.CompilerServices;
using System.Runtime.Serialization;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;
using System.Windows.Media;
using NPSim.Domain.PhysicalLayer;
using NPSim.Entities.PhysicalLayer.Media;
using NPSim.ViewModels;
using NPSim.Views;

namespace NPSim.Models
{

```

```

[DataContract]
public class PhysicalMediaModel : INotifyPropertyChanged
{
    private readonly IMediaManager _mediaManager;
    private readonly MainWindowVm _mainWindowVm;

    [DataMember]
    private Point? _from;

    [DataMember]
    private Point? _to;

    [DataMember]
    public PhysicalMedia PhysicalMedia { get; private set; }

    public Point? From
    {
        get => _from;
        set
        {
            _from = value;
            OnPropertyChanged("From");
        }
    }

    public Point? To
    {
        get => _to;
        set
        {
            _to = value;
            OnPropertyChanged("To");
        }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    public PhysicalMediaModel(PhysicalMedia physicalMedia, IMediaManager mediaManager, MainWindowVm
mainWindowVm)
    {
        PhysicalMedia = physicalMedia;
        _mediaManager = mediaManager;
    }

```

```

        _mainWindowVm = mainWindowVm;
    }

    public void OnPropertyChanged([CallerMemberName]string prop = "")
    {
        PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(prop));
    }

    public void OnOpenSystemMove()
    {
        throw new NotImplementedException();
    }

    public void OpenSystemUiElement_MouseUp(object sender, MouseButtonEventArgs e)
    {
        if (sender is StackPanel OpenSystemCanvasElement)
        {
            var openSystemModel = (OpenSystemModel)OpenSystemCanvasElement.DataContext;
            var connectionEndpoint =
openSystemModel.OpenSystem.GetAvailableNetworkInterfaces().FirstOrDefault();

            _mediaManager.AttachMediaToConnectionEndpoint(PhysicalMedia, connectionEndpoint);

            if (PhysicalMedia.HasAvailableConnector)
            {
                From = GetCenterOfUiElement(OpenSystemCanvasElement);
                To = From;
            }
            else
            {
                To = GetCenterOfUiElement(OpenSystemCanvasElement);

                var openSystemModels = _mainWindowVm.OpenSystemVm.OpenSystemModels;
                foreach (var item in openSystemModels)
                {
                    item.MouseLeftButtonUp -= OpenSystemUiElement_MouseUp;
                }

                _mainWindowVm.SimulationArea.MouseMove -= OpenSystemUiElement_MouseMove;

                Application.Current.MainWindow.Cursor = Cursors.Arrow;
            }
        }
    }

```

```

    }
}

public void OpenSystemUiElement_MouseMove(object sender, MouseEventArgs e)
{
    if (From.HasValue)
    {
        var simulationArea = (UIElement)sender;
        To = e.GetPosition(simulationArea);
    }
}

private Point GetCenterOfUiElement(UIElement element)
{
    return new Point(Canvas.GetLeft(element) + element.RenderSize.Width / 2, Canvas.GetTop(element) +
element.RenderSize.Height / 2);
}
}
}

```

SharingServiceCallback.cs

```

using System.ServiceModel;
using System.Threading;
using System.Threading.Tasks;
using System.Windows;
using System.Windows.Threading;
using NPSim.Models;
using NPSim.Sharing.Lib;
using NPSim.ViewModels;

namespace NPSim.Sharing.Client
{
    [CallbackBehavior(ConcurrencyMode = ConcurrencyMode.Multiple, UseSynchronizationContext = false)]
    public class SharingServiceCallback : ISharingServiceCallback
    {
        private SynchronizationContext _uiSyncContext;
        private MainWindowVm _mainWindowVm;

        public SharingServiceCallback(SynchronizationContext synchronizationContext, MainWindowVm
mainWindowVm)
        {

```

```

        _uiSyncContext = synchronizationContext;
        _mainWindowVm = mainWindowVm;
    }

    public Task LoadProjectAsync(NetworkProjectModel networkProjectModel)
    {
        return Task.Factory.StartNew(() =>
        {
            _uiSyncContext.Post((c) => _mainWindowVm.LoadOrCreateNewProject((NetworkProjectModel)c),
networkProjectModel);

            });
        }
    }
}

```

SharingServiceClient.cs

```

using System.ServiceModel;
using System.ServiceModel.Channels;
using NPSim.Sharing.Lib;

namespace NPSim.Sharing.Client
{
    public class SharingServiceClient : DuplexClientBase<ISharingService>, ISharingService
    {

        public SharingServiceClient(InstanceContext callbackInstance) :
            base(callbackInstance)
        {
        }

        public SharingServiceClient(InstanceContext callbackInstance, string endpointConfigurationName) :
            base(callbackInstance, endpointConfigurationName)
        {
        }

        public SharingServiceClient(InstanceContext callbackInstance, string endpointConfigurationName, string
remoteAddress) :
            base(callbackInstance, endpointConfigurationName, remoteAddress)
        {
        }
    }
}

```