

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет прикладної математики та інформатики

(повне найменування назва факультету)

Кафедра дискретного аналізу та інтелектуальних систем

(повна назва кафедри)

Кваліфікаційна (бакалаврська) робота

ВИКОРИСТАННЯ ВЛАСНОГО ПРОТОКОЛУ ПЕРЕДАЧІ ДАНИХ У СИСТЕМАХ ВІДЕОСПОСТЕРЕЖЕННЯ

Виконав: студент групи ПМі-45с
спеціальності

122 «Комп'ютерні науки»

(шифр і назва спеціальності)

Пастушук О.В.

(підпис)

(прізвище та ініціали)

Керівник доц. Коковська Я.В.

(підпис)

(прізвище та ініціали)

Рецензент

(підпис)

(прізвище та ініціали)

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Факультет Прикладної математики та інформатики _____

Кафедра Дискретного аналізу та інтелектуальних систем _____

Спеціальність 122 «Комп'ютерні науки»
(шифр і назва)

«ЗАТВЕРДЖУЮ»

Завідувач кафедри _____

"30 "серпня 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Пастушук Ользі Валеріївні

(прізвище, ім'я, по батькові)

1. Тема роботи Використання власного протоколу передачі даних у системах відеоспостереження

керівник роботи Коковська Ярина Володимирівна, доцент кафедри дискретного аналізу та інтелектуальних систем

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені Вченою радою факультету від **"27" вересня 2023 року № 24**

2. Строк подання студентом роботи 14.06.2024 р.

3. Вихідні дані до роботи мова програмування Python, середовище розробки Visual Studio Code,

4. Зміст дипломної роботи (перелік питань, які потрібно розробити) Дослідження проблем протоколів передачі даних, вивчення конкурентів, вивчення складових протоколу передачі даних, написання протоколу та симулятивної програми, експерименти з апаратним забезпеченням

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Схема елементарної системи відеоспостереження, таблиці функцій та методів коду власного протоколу, знімки уривків коду.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання **30 серпня 2023 р.**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи (бакалаврської)	Строк виконання етапів роботи	Примітка
1.	Вивчення предметної області та індустрії	24.10.2023-29.10.2023	Виконано
2.	Вивчення конкурентів наявних протоколів	05.11.2023-10.11.2023	Виконано
3.	Вивчення складових протоколу	30.11.2023-03.12.2023	Виконано
4.	Написання власного протоколу	05.01.2024-25.04.2024	Виконано
5.	Експерименти з апаратним забезпеченням	26.04.2024-04.05.2024	Виконано
6.	Написання симулятивної програми	05.05.2024-17.05.2024	Виконано
7.	Представлення результатів	20.05.2024-25.05.2024	Виконано
8.	Оформлення дипломної роботи	28.05.2024-06.05.2024	Виконано

Студент _____ **Пастушок О.В.**
 (підпис) (прізвище та ініціали)

Керівник роботи _____ **доц. Коковська Я.В.**
 (підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП	6
Актуальність теми	7
Мета	7
Постановка задачі.....	8
1 ОГЛЯД ЛІТЕРАТУРИ ТА АНАЛІЗ НАЯВНИХ РІШЕНЬ	9
1.1 Історія розвитку систем відеоспостереження	9
1.1.1 1940-1960 рр.: Ранні етапи	9
1.1.2 1970-1980 рр.: Розширення можливостей	9
1.1.3 1990-і роки: Цифрова революція	10
1.1.4 2000-ні - сьогодні: Інтелектуальні системи	10
1.1.5 Перспективи розвитку	10
1.2 Сучасні стандарти та протоколи передачі даних	11
1.2.1 RTSP (Real-Time Streaming Protocol).....	11
1.2.2 ONVIF (Open Network Video Interface Forum)	11
1.2.3 H.264 та H.265 (HEVC).....	12
1.3 Порівняльний аналіз популярних протоколів	12
1.4 Недоліки та обмеження наявних протоколів.....	14
2 МЕТОДОЛОГІЯ	16
2.1 Вимоги до протоколу	16
2.2 Методи стиснення даних	18
2.3 Механізми обробки помилок	18
2.4 Методи шифрування	19
3 СИСТЕМНА ІНТЕГРАЦІЯ	21
3.1 Вимоги до обладнання.....	21

3.2 Інтеграція програмного забезпечення.....	22
3.3 Конфігурація мережі.....	23
4 РОЗРОБКА ВЛАСНОГО ПРОТОКОЛУ	24
4.1 Структура проекту.....	24
4.2 Файл protocol.py	25
4.3 Файл memory.py.....	29
4.4 Файл endpoints.py	32
4.5 Файли server.py та client.py	34
5 ДЕМОНСТРАЦІЯ РОБОТИ.....	36
Висновок	39
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	40

ВСТУП

За останні кілька десятиліть відеоспостереження стало невід'ємною частиною безпеки в багатьох місцях, від приватних будинків до великих комерційних будівель і громадських місць. Еволюція цих систем відбувалася завдяки технологічному прогресу, що призвело до розробки й виготовлення камер із вищою роздільною здатністю, більш розгалуженої мережевої інфраструктури чи з вбудованим штучним інтелектом. Очевидно, поряд із цими досягненнями, потреба в ефективних і безпечних протоколах передачі даних стає все більшою.

Традиційні протоколи передачі даних, зокрема HTTP (протокол передачі гіпертексту), FTP (протокол передачі файлів) і RTP (протокол передачі даних у режимі реального часу), не задовольняли суворих вимог програм відеоспостереження. Ці протоколи мають різні обмеження, наприклад, затримки, неефективності пропускну здатності та вразливість до порушень безпеки. У результаті може бути важко забезпечити високоякісний потік відео в режимі реального часу, необхідні для сучасних систем відеоспостереження, особливо у великих масштабах, наприклад, на підприємствах чи в офісах.

Крім того, з інтеграцією технології відеоспостереження у взаємопов'язані системи охорони важливість безпеки даних неможливо переоцінити. Відеопотоки, що містять конфіденційну інформацію, потребують надійного шифрування та механізмів автентифікації, щоб запобігти несанкціонованому доступу та втручанню.

Метою цієї дипломної роботи є розробка та впровадження такого спеціального протоколу передачі даних для систем відеоспостереження. І, з заглибленням у технічні тонкощі розробки та інтеграції протоколів, зробити внесок у розвиток технології відеоспостереження, та зрештою посприяти безпеці та ефективності практик спостереження в різних сферах.

Актуальність теми

Оскільки системи відеоспостереження відіграють важливу роль у забезпеченні безпеки в різних сферах життя, зростання кількості відеокамер та підвищення вимог до якості відеоматеріалів потребує удосконалення протоколів передачі даних. Сучасні протоколи, зокрема RTSP, ONVIF та інші, мають певні обмеження, зокрема ненадійність, безпеку та ефективність використання мережевих ресурсів.

Крім того, з появою нових технологій, зокрема штучного інтелекту та аналізу відео в режимі реального часу, виникає необхідність у протоколах, що можуть забезпечити низькі затримки передачі та високу якість обслуговування. Це особливо важливо в умовах міських систем безпеки та критичної інфраструктури, де навіть незначна затримка або втрата даних може мати серйозні наслідки.

Власний протокол може врахувати специфічні потреби конкретних систем відеоспостереження та забезпечити більш гнучке та оптимізоване рішення, ніж загальноприйняті стандарти.

Таким чином, розробка та впровадження власного протоколу передачі даних у системах відеоспостереження є надзвичайно актуальною задачею та відповідає сучасним тенденціям розвитку інформаційних технологій та систем безпеки.

Мета

Метою цього дослідження є спроба подолати недоліки наявних протоколів передачі даних у системах відеоспостереження шляхом розробки та впровадження власного протоколу передачі даних. Цей протокол повинен забезпечити високу надійність, безпеку та ефективність використання мережевих ресурсів, а також задоволення специфічних вимог сучасних систем відеоспостереження. Основною метою є подолання недоліків наявних протоколів передачі даних шляхом створення спеціалізованого протоколу, оптимізованого для передачі в режимі реального часу, низької затримки та надійної безпеки.

ПОСТАНОВКА ЗАДАЧІ

Спершу буде досліджено, наскільки наявні протоколи передачі даних відповідають вимогам сучасних систем відеоспостереження, а потім проаналізовано основні причини їхньої неефективності. Наступним завданням буде створення протоколу передачі даних, що враховуватиме унікальні характеристики та вимоги, зокрема мінімізацію затримки, оптимізацію використання пропускної здатності та посилення заходів безпеки.

Метою цієї дипломної роботи є зробити внесок у розвиток технологій відеоспостереження, і таким чином посприяти безпечнішим, ефективнішим і надійнішим практикам спостереження в різних сферах та дати можливість розкрити весь потенціал систем відеоспостереження.

1 ОГЛЯД ЛІТЕРАТУРИ ТА АНАЛІЗ НАЯВНИХ РІШЕНЬ

1.1 Історія розвитку систем відеоспостереження

Історія розвитку систем відеоспостереження починається в середині 20 століття і містить кілька ключових етапів, що визначили еволюцію технологій та їх впровадження у різних сферах життя.

1.1.1 1940-1960 рр.: Ранні етапи

Перші системи відеоспостереження з'явилися у 1940-х роках. У 1942 році німецька компанія Siemens AG розробила систему для моніторингу запуску ракет V-2. Ця система була закритою (CCTV - Closed Circuit Television), тобто відеосигнал передавався через кабелі без можливості широкого розповсюдження.

У 1960-х роках CCTV-системи почали використовуватися у США для забезпечення безпеки в комерційних та урядових будівлях, банках та інших важливих об'єктах. Технологія залишалася дорогою та складною у встановленні, тому її застосування було обмеженим. [1]

1.1.2 1970-1980 рр.: Розширення можливостей

У 1970-х роках розвиток електроніки сприяв зменшенню розмірів і вартості компонентів CCTV-систем. Камери стали більш компактними та доступними, що збільшило кількість встановлень у приватних та комерційних об'єктах.

У 1980-х роках почали з'являтися системи з можливістю запису відео на магнітні стрічки, що дозволило зберігати відеоматеріали для подальшого перегляду та аналізу. Відеомагнітофони (VCR) стали стандартним обладнанням для систем відеоспостереження, що суттєво підвищило їх ефективність. [1]

1.1.3 1990-і роки: Цифрова революція

У 1990-х роках відбувся перехід від аналогових до цифрових систем відеоспостереження. З'явилися цифрові відеореєстратори (DVR), які дозволяли зберігати відео на жорстких дисках з вищою якістю зображення та зручністю управління записами. Цифрові технології також спростили процес пошуку та аналізу відеоматеріалів. [2]

З розвитком Інтернету та комп'ютерних мереж були створені IP-камери, що передавали відео мережею, завдяки чому можна було віддалено спостерігати за об'єктами у режимі реального часу. Це відкрило нові можливості для інтеграції систем відеоспостереження з іншими інформаційними системами.

1.1.4 2000-ні - сьогодні: Інтелектуальні системи

На початку 2000-х років технології відеоспостереження продовжили стрімко розвиватися. З'явилися мережеві відеореєстратори (NVR) та розподілені системи зберігання даних, а Інтернет дозволив використовувати хмарні сервіси для зберігання та обробки відеоматеріалів.

Сучасні системи відеоспостереження оснащені функціями відеоаналітики, які використовують алгоритми штучного інтелекту для виявлення та аналізу подій у режимі реального часу. Наприклад, розпізнавання облич, визначення підозрілої поведінки, підрахунок кількості людей та інші функції допомагають автоматизувати процеси безпеки та підвищують ефективність моніторингу.

1.1.5 Перспективи розвитку

Сьогодні технології відеоспостереження продовжують еволюціонувати. Інтеграція з Інтернетом речей (IoT), розвиток технологій обробки великих даних та вдосконалення алгоритмів машинного навчання обіцяють ще більшу автоматизацію та інтелектуалізацію систем. Нові протоколи передачі даних разом з власними рішеннями, що спрямовані на підвищення надійності, безпеки та ефективності систем відеоспостереження, стають важливими елементами цього розвитку.

Таким чином, історія систем відеоспостереження демонструє постійний розвиток технологій, які адаптуються до нових викликів і потреб суспільства, та підвищують рівень безпеки та ефективності управління об'єктами.

1.2 Сучасні стандарти та протоколи передачі даних

Сучасні системи відеоспостереження використовують різноманітні стандарти та протоколи передачі даних для забезпечення надійної та ефективної комунікації між компонентами системи, наприклад камерами, відеореєстраторами та програмним забезпеченням для моніторингу. Основні стандарти та протоколи, що використовуються в відеоспостереженні - це RTSP, ONVIF та H.264/H.265.

1.2.1 RTSP (Real-Time Streaming Protocol)

RTSP є мережевим протоколом, розробленим для передачі медіаданих у режимі реального часу. Він дозволяє керувати відеопотоками між медіасерверами та клієнтами, забезпечує такі функції, як відтворення, пауза, зупинка та запис відео. Цей протокол забезпечує мінімальну затримку при передачі відео, що важливо для систем відеоспостереження в режимі реального часу, а також підтримує різні медіаформати та кодеки, що дозволяє використовувати його з широким спектром обладнання, та легко інтегрується з іншими мережевими протоколами, наприклад RTP (Real-Time Transport Protocol) та RTCP (Real-Time Control Protocol), для забезпечення високої якості відеопотоків.

1.2.2 ONVIF (Open Network Video Interface Forum)

ONVIF – це глобальний стандарт, що забезпечує сумісність між мережевими пристроями відеоспостереження, незалежно від виробника, що дозволяє створювати гнучкі та масштабовані системи відеоспостереження. Його використання спрощує інтеграцію нових пристроїв у наявні системи. Також, ONVIF підтримує шифрування, автентифікацію користувачів та інші механізми захисту даних.

1.2.3 H.264 та H.265 (HEVC)

H.264 та його наступник H.265 (High Efficiency Video Coding) є стандартами відеокодування, що використовуються для стиснення відеопотоків та забезпечують високу якість зображення при зменшенні обсягу переданих даних. Порівняно з H.264, H.265 забезпечує до 50% більш ефективне стиснення, що дозволяє передавати відео високої роздільної здатності з меншими вимогами до пропускної здатності мережі. Обидва стандарти підтримують різні рівні якості зображення, від стандартної роздільної здатності до 4K та вище, і підтримуються більшістю сучасних пристроїв відеоспостереження, що робить їх універсальними рішеннями для стиснення відео.

1.3 Порівняльний аналіз популярних протоколів

Для систем відеоспостереження використовуються різні протоколи та стандарти, які забезпечують передачу, обробку та зберігання відеоданих. Як було вказано у попередньому розділі, до найбільш популярних належать RTSP, ONVIF та стандарти відеокодування H.264/H.265. Порівняльний аналіз цих протоколів дозволяє виявити їх переваги та недоліки для вибору оптимального рішення.

З метою кращого та більш явного розуміння переваг та недоліків, інформація про них подана у вигляді таблиці 1.1 та взята з документацій [3-5].

Таблиця 1.1 - Порівняння популярних протоколів

Параметр	RTSP	ONVIF	H.264	H.265
Затримка передачі	Низька	Середня	Залежить від налаштувань	Залежить від налаштувань
Сумісність	Висока (підтримка багатьох форматів)	Висока (між різними виробниками)	Висока (широка підтримка)	Висока (широка підтримка)
Якість зображення	Залежить від використовуваного кодека	Залежить від пристроїв	Висока	Вища ніж у H.264
Ефективність стиснення	Залежить від кодека	Не застосовується	Висока	Дуже висока
Безпека	Потребує додаткових налаштувань	Вбудовані механізми	Потребує додаткових налаштувань	Потребує додаткових налаштувань
Простість налаштування	Середня	Висока	Середня	Середня
Обчислювальна потужність	Низька-середня	Не застосовується	Середня	Висока
Ліцензійні витрати	Немає	Немає	Немає	Можливі

Кожен з розглянутих протоколів та стандартів має свої переваги та недоліки, які слід враховувати при виборі конкретної системи відеоспостереження. RTSP забезпечує гнучкість та низьку затримку, що важливо для спостереження в режимі реального часу, але потребує додаткових налаштувань для безпеки. ONVIF полегшує інтеграцію пристроїв різних виробників, проте може не підтримувати всі функції камер. H.264 та H.265 забезпечують високу якість зображення та ефективність стиснення, проте H.265 вимагає більше обчислювальних ресурсів і може бути пов'язаний з ліцензійними витратами. Вибір протоколу залежить від конкретних вимог замовника, однак поєднати їх усі можливо лише шляхом розробки власного протоколу.

1.4 Недоліки та обмеження наявних протоколів

Згідно з заданою метою дослідження, а саме розробкою протоколу передачі даних, спершу варто розглянути загальні обмеження уже наявних протоколів, щоб зможти їх вирішити.

Складність інтеграції є однією з найважливіших проблем: інтеграція різних протоколів у єдину мережеву систему може бути складною через різні вимоги до налаштувань та сумісності, як це часто трапляється під час використання однієї з провідних систем безпеки - Hikvision. У користувачів неодноразово виникали проблеми з під'єднанням кількох камер відеоспостереження до одного реєстратора через несумісність обробки різних протоколів передачі даних. Інтерфейс керування цієї системи безпеки є загальнодоступним, однак підказки та інструкції практично відсутні, а під час виникнення помилки підключення такому користувачу самотійно розібратися та зрозуміти, що саме він робить не так, не вдасться. Тому є доцільним використовувати єдиний власний протокол передачі даних, щоб мінімізувати кількість причин неправильного підключення камер до мережі.

Високоякісне відео під час його запису на сервер та відображення у клієнті потребує великої пропускної здатності мережі, що може бути проблематичним в умовах обмеженої інфраструктури, особливо якщо мова йтиме про системи з більшою кількістю пристроїв у них. Деякі протоколи можуть неефективно працювати у великих масштабованих системах через обмеження в обробці великої кількості потоків або пристроїв.

Аналогічно важливим аспектом є й забезпечення надійного захисту даних, що потребує додаткових заходів та може ускладнювати загальну систему. Незалежно від цілі використання відеоспостереження (вдома чи в офісних приміщеннях) важливо зберігати конфіденційність отримуваних даних, оскільки витоки інформації є порушенням недоторканості приватного життя та можуть спричинити низку соціальних і не тільки проблем. Террі Данлеп, колишній американський хакер та співзасновник компанії ReFirm Labs, у своїй статті розповідає, як його команда

отримала модель камери Dahua, другого провідного виробника камер відеоспостереження у Китаї, відразу після Hikvision, для перевірки на безпечність. Провідний реверсивний інженер ReFirm Labs, Крейг Хефнер, виявив у цій камері програмну лазівку, що була доволі глибоко вбудована у її мікропрограму [6].

У своєму остаточному звіті Крейг стверджує, що вразливість не є результатом якихось випадкових логічних помилок або неправильного програмування, а є результатом саме навмисного впровадження цієї лазівки постачальником. Оскільки багато інших продуктів Dahua містять схожого типу лазівки, Крейг категорично не рекомендує використовувати продукти Dahua у системах відеонагляду та охорони в критично важливих або вразливих мережах.

2 МЕТОДОЛОГІЯ

Під час ознайомлення з наявними протоколами було виявлено низку недоліків, які потрібно подолати під час розробки власного протоколу. Сперш ніж приступати до програмної реалізації, варто окреслити вимоги до протоколу, поділені за категоріями, та ознайомитися з методами стиснення та шифрування даних.

2.1 Вимоги до протоколу

Після аналізу недоліків та обмежень наявних протоколів передачі даних врахування низки критичних вимог, які забезпечать його ефективність, безпеку та надійність. Основні вимоги до протоколу можна розділити на кілька категорій: функціональні, технічні, безпекові та експлуатаційні.

1) Функціональні вимоги:

- a) Низька затримка передачі: потрібно забезпечувати мінімальну затримку при передачі відеопотоків, щоб моніторинг у режимі реального часу працював коректно;
- b) Підтримка високої якості відео: протокол повинен підтримувати передачу відео високої роздільної здатності (HD, Full HD, 4K та вище).
- c) Масштабованість: робота в умовах великої кількості пристроїв та користувачів повинна бути ефективною, забезпечувати стабільність і надійність при збільшенні навантаження;
- d) Гнучкість та сумісність: обов'язкова підтримка різних форматів відео та аудіо, а також бути сумісним з іншими стандартами та протоколами, наприклад RTSP, ONVIF, H.264/H.265;

2) Технічні вимоги:

- a) Ефективність використання пропускну здатності: необхідно оптимізувати використання пропускну здатності мережі для забезпечення плавного та безперебійного відеопотоку, навіть у мережах з обмеженими ресурсами;

- b) Підтримка різних мережевих середовищ: протокол повинен ефективно працювати в різних мережевих середовищах, тобто з локальними мережами (LAN), широкомасштабними мережами (WAN) та бездротовими мережами;
- c) Вбудовані механізми відновлення: потрібно додати механізми відновлення даних у випадку втрати пакетів або перебоїв у зв'язку, що забезпечуватимуть високу надійність передачі;

3) Безпекові вимоги:

- a) Шифрування даних: необхідно забезпечити постійне та безперервне шифрування переданих відео та аудіо даних для захисту від несанкціонованого доступу;
- b) Автентифікація та авторизація: протокол повинен мати механізми автентифікації користувачів та пристроїв, а також контроль доступу для запобігання несанкціонованому доступу до системи відеоспостереження;
- c) Захист від атак: розробити механізми стійкості до різних типів атак, зокрема, від атаки типу «відмова в обслуговуванні» (DoS), підслуховування (eavesdropping) та маніпуляції даними;

4) Експлуатаційні вимоги:

- a) Легкість налаштування та використання: протокол має бути простим у налаштуванні та використанні, з інтуїтивно зрозумілим інтерфейсом та документацією, що полегшують впровадження та експлуатацію;
- b) Моніторинг та діагностика: протокол повинен мати інструменти для моніторингу стану системи та діагностики проблем, що дозволить оперативно реагувати на будь-які збої або несправності;

Врахування усіх перелічених вище вимог дозволить створити власний протокол передачі даних, що відповідатиме не лише сучасним стандартам та потребам систем відеоспостереження, а й специфічним потребам користувачів, забезпечивши ефективну роботу в різних умовах та середовищах.

2.2 Методи стиснення даних

Оскільки забезпечення мінімальної затримки при передачі відеопотоків є важливою вимогою, потрібно вдосконалити методи стиснення даних для оптимізації використання пропускної здатності та зменшення накладних витрат на передачу. Серед цих методів:

- 1) Підтримку стандартних алгоритмів стиснення відео, зокрема H.264, H.265 (HEVC) або VP9, щоб зменшити розмір відеопотоків без шкоди для якості зображення.
- 2) Використання дельта-стиснення для передачі лише змін (дельта) між послідовними відеокадрами, що зменшить надмірність і збереже уже наявну пропускну здатність.
- 3) Механізми стиснення заголовків, що мінімізують накладні витрати, пов'язані з заголовками пакетів, що оптимізує використання пропускної здатності для невеликих корисних даних.
- 4) Використання адаптивного потокового передавання бітів для динамічного налаштування бітрейту відеопотоків на основі умов мережі, що забезпечить плавне відтворення та ефективне використання пропускної здатності.

Під час написання методів стиснення пакетів буде використана бібліотека `zlib` в Python, що використовується для стиснення і розпакування даних. Вона базується на алгоритмі DEFLATE, який комбінує методи стиснення LZ77 і кодування Хаффмана.

2.3 Механізми обробки помилок

Щоб забезпечити надійну передачу даних, протокол містить надійні механізми обробки помилок для виявлення та виправлення помилок, які виникають під час передачі. До таких механізмів належать:

- Пряме виправлення помилок (FEC): Технології FEC використовуються для додавання надлишкових даних до переданих пакетів, що дозволяє одержувачам відновлюватися після втрати або пошкодження пакетів без необхідності повторної передачі;
- Автоматичний запит на повторення (ARQ): протоколи ARQ реалізовані для запиту повторної передачі втрачених або пошкоджених пакетів та забезпечують цілісність і повноту даних;
- Нумерація послідовності: кожному пакету присвоюється унікальний порядковий номер, що дозволяє одержувачам виявляти пакети, що не відповідають порядку, і збирати їх у правильному порядку;
- Перевірка контрольної суми: контрольні суми або циклічні надлишкові перевірки (CRC) обчислюються для кожного пакета, щоб перевірити цілісність даних і виявити помилки передачі.

2.4 Методи шифрування

Безпека є головною проблемою в системах відеоспостереження, тому протокол передбачає надійні методи шифрування для захисту відеоданих від несанкціонованого доступу та перехоплення. До методів шифрування відносять:

- Симетричне шифрування: симетричні алгоритми шифрування, зокрема AES (Advanced Encryption Standard), використовуються для шифрування відеоданих перед передачею та забезпечують конфіденційність і приватність;
- Асиметричне шифрування: алгоритми асиметричного шифрування, наприклад, RSA (Рівест–Шамір–Адлеман), використовуються для обміну ключами та автентифікації, що забезпечує безпечний зв'язок між сторонами без попереднього використання спільних ключів;
- Управління ключами. Для безпечного створення, розповсюдження та зберігання ключів шифрування реалізовано надійні методи керування ключами, що мінімізує ризик зламу ключа або несанкціонованого доступу;

- Цифрові підписи: цифрові підписи використовуються для автентифікації переданих даних і перевірки їх походження, забезпечення цілісності даних і запобігання фальсифікації або підробці.

3 СИСТЕМНА ІНТЕГРАЦІЯ

Для тестування як наявних, так і власних протоколів передачі даних, необхідне спеціалізоване програмне середовище, яке підтримує роботу з відеокамерами, а також відповідне апаратне забезпечення. Важливо забезпечити наявність необхідних серверів, мережевих комутаторів і маршрутизаторів, що можуть підтримувати високі навантаження та забезпечувати надійну передачу відеоданих. Окрім цього, потрібно вміти налаштовувати програмне забезпечення, щоб воно відповідало специфікаціям протоколу та вимогам до якості обслуговування. Необхідно також мати навички конфігурації мережі для оптимізації пропускну здатності і мінімізації затримок під час передачі відеопотоків.

3.1 Вимоги до обладнання

Перший етап системної інтеграції передбачає визначення необхідних апаратних компонентів для створення системи відеоспостереження з метою тестування власного протоколу передачі даних. Основні вимоги до апаратного забезпечення:

- Камери: необхідні IP-камери високої чіткості, здатні записувати та кодувати відеопотоки в режимі реального часу. Обрані камери також повинні підтримувати протоколи галузевого стандарту, наприклад RTSP (протокол потокової передачі в режимі реального часу) або ONVIF (форум відкритого мережевого відеоінтерфейсу) для взаємодії з настроюваним протоколом;
- Сервери запису: потрібні сервери, устатковані достатніми обчислювальною потужністю та об'ємом пам'яті для запису та зберігання відеозапису з кількох камер. Сервери повинні підтримувати конфігурації RAID (надлишкового масиву незалежних дисків) для резервування даних і відмовостійкості;
- Мережеве обладнання: також необхідні комутатори, маршрутизатори та інші компоненти мережевої інфраструктури, здатні підтримувати

високошвидкісну передачу даних і зв'язок із малою затримкою між камерами, серверами запису та консолями моніторингу. Необхідно підтримувати функції якості обслуговування (QoS), щоб визначати пріоритет відеотрафіку над іншим мережевим трафіком;

- Консолі моніторингу: Робочі станції або спеціальні консолі моніторингу, оснащені програмним забезпеченням керування відео для моніторингу, відтворення та аналізу відеопотоків у режимі реального часу. Програмне забезпечення має підтримувати інтеграцію з власним протоколом передачі даних для безперебійного зв'язку з камерами та серверами запису.

Оскільки програмне забезпечення на консолях моніторингу залежить переважно від виробника та постачальника відеокамер, відшукати камеру з можливістю інтеграції власного протоколу передачі даних є непростим завданням. Відтак, для виконання цієї роботи було прийняте рішення про симуляцію дії власного протоколу передачі даних за допомогою клієнт-серверної програми, написаної мовою програмування Python.

3.2 Інтеграція програмного забезпечення

На цьому етапі передбачається налаштування та інтеграція компонентів програмного забезпечення для підтримки власного протоколу передачі даних у системі відеоспостереження. До ключових аспектів інтеграції програмного забезпечення належать:

- 1) Реалізація власного протоколу передачі даних у програмному забезпеченні для керування відео, серверах запису та інших відповідних компонентах;
- 2) Інтеграція API або комплектів розробки програмного забезпечення (SDK), наданих виробниками камер або сторонніми постачальниками, для забезпечення взаємодії з налаштованим протоколом;
- 3) Керування конфігурацією налаштувань і параметрів у програмному забезпеченні керування відео, серверах запису та мережевій інфраструктурі

для оптимізації продуктивності, надійності та безпеки. дозволів користувача та мережових політик;

- 4) Проведення комплексного тестування та перевірки інтеграції програмного забезпечення для забезпечення сумісності, функціональності та продуктивності.

3.3 Конфігурація мережі

Етап конфігурації мережі передбачає налаштування мережових налаштувань і параметрів для підтримки спеціального протоколу передачі даних і забезпечення ефективного зв'язку між камерами, серверами запису та консолями моніторингу.

Основні аспекти конфігурації мережі містять:

- 1) Призначення статичних або динамічних IP-адрес камерам, серверам запису та іншим мережовим пристроям для полегшення зв'язку та забезпечення постійного з'єднання;
- 2) Сегментація мережі на підмережі або віртуальні локальні мережі (VLAN) для ізоляції відеотрафіку від іншого мережевого трафіку та визначення пріоритетів відеопотоків за допомогою налаштувань якості обслуговування (QoS);
- 3) Налаштування правил переадресації портів і параметрів брандмауера, щоб дозволити вхідний і вихідний трафік для спеціального протоколу передачі даних;
- 4) Відстеження показників продуктивності мережі, зокрема затримки, пропускної здатності і втрати пакетів, для виявлення й вирішення потенційних вузьких місць або проблем.

Завдяки систематичному вирішенню вимог до апаратного забезпечення, інтеграції програмного забезпечення та конфігурації мережі система відеоспостереження може бути ефективно інтегрована з користувальницьким протоколом передачі даних, завдяки чому забезпечити надійну, ефективну та безпечну передачу відеоданих у системах спостереження.

4 РОЗРОБКА ВЛАСНОГО ПРОТОКОЛУ

Програмна реалізація передбачає розробку власного протоколу передачі даних з використанням мови програмування Python та програмного середовища Visual Studio Code. Основна увага приділяється створенню ефективного коду, який забезпечуватиме мінімальні затримки і високу надійність передачі відеопотоків. Для цього необхідно використовувати сучасні бібліотеки та інструменти Python, такі `zlib` для стиснення даних.

4.1 Структура проєкту

Проєкт складається з кількох файлів, які розподілені по їх функціоналу, що зображено на рисунку 4.1:

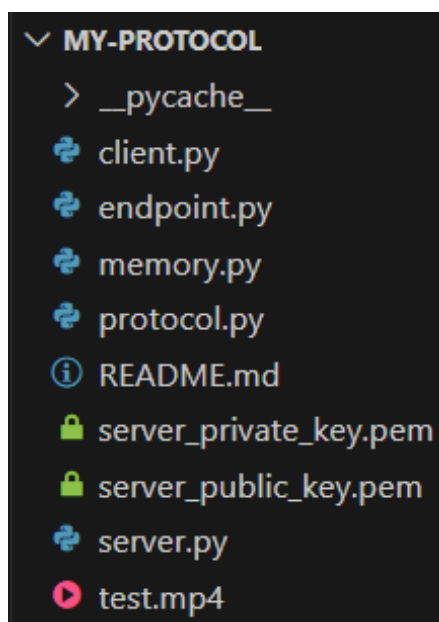


Рисунок 4.1 - Список файлів

У файлі `protocol.py` реалізовано сам клас `Protocol` та допоміжний йому клас `Package`, що обгортає клас протоколу, підтримує перевірку цілісності даних, розділення `Protocol` на кілька частин, повертає інформацію щодо прогресу виконання і дозволяє налаштувати обробку помилок.

Файл `endpoint.py` містить реалізований клас `Endpoint`, що дозволяє створювати бізнес-логіку для програм, написаних з бібліотекою `Flask` та схожих до неї.

У файлі `memory.py` здійснюється управління внутрішньою пам'яттю та зберіганням даних.

У файлах `server.py` та `client.py` реалізовано простий інтерфейс сервера та клієнта відповідно для наочної демонстрації роботи протоколу за відсутності необхідного апаратного забезпечення для демонстрації у максимально реальних умовах.

4.2 Файл `protocol.py`

За допомогою класів `Protocol` та `Package` здійснюється обробка потоків даних та управління протоколами передачі даних. Основними компонентами цього файлу є ці два класи (`Protocol` та `Package`) і допоміжні функції, що будуть наведені далі.

Клас `Protocol` забезпечує методи для роботи з потоками даних, а саме методи для збереження та завантаження даних, обробки метаданих та передачі даних по потоках. Методи цього класу відображені у наступній таблиці 4.1:

Таблиця 4.1 – Методи класу Protocol

Назва	Опис
<code>__init()</code>	Ініціалізує об'єкт протоколу з параметрами мети, розширення файлу, методу кодування та розміру буфера.
<code>__str__()</code>	Повертає рядкове представлення об'єкта протоколу.
<code>update()</code>	Оновлює дані заголовка протоколу.
<code>upmeta()</code>	Змінює метадані.
<code>head()</code>	Повертає заголовок даних.
<code>code()</code>	Декодує метадані.
<code>json()</code>	Конвертує метадані в об'єкт JSON.
<code>pack()</code>	Повертає дані з заголовками та метаданими.
<code>unpack()</code>	Розпаковує дані з вхідного байтового рядка.
<code>write()</code>	Записує дані в потік.
<code>read()</code>	Читає дані з потоку.
<code>readbit()</code>	Читає дані з потоку побітово.
<code>seek()</code>	Переміщує позицію зчитування/запису в потоці.
<code>load_stream()</code>	Завантажує дані з потоку.
<code>create_stream()</code>	Створює потік.
<code>ignore_stream()</code>	Ігнорує потік.
<code>save_stream_io()</code>	Зберігає потік у вказаний файл.
<code>conver_full_stream_io()</code>	Конвертує весь потік.
<code>stream_until()</code>	Обробляє потік.
<code>parse_stream_head()</code>	Розбирає заголовок потоку.
<code>parse_static_head()</code>	Розбирає статичний заголовок.
<code>make_head()</code>	Генерує заголовок.
<code>recv_stream_only_by_extn()</code>	Отримує потік лише за певним розширенням.

Клас `Package` відповідає за ініціалізацію, запуск, завершення та обробку відправлення та отримання даних через потоки. Цей клас використовує функцію `safecode` для генерації безпечних кодів, що використовуються у потоках передачі даних. Методи класу наведені у таблиці 4.2 :

Таблиця 4.2 - Методи класу `Package`

Назва	Опис
<code>__init__(sender, recver, buffer, handle, file_buffer)</code>	Конструктор класу, ініціалізує атрибути об'єкта.
<code>encrypt(data)</code>	Здійснює шифрування з використанням схеми ОАЕР
<code>decrypt(encrypted_data)</code>	Здійснює розшифрування
<code>start()</code>	Запускає потоки відправлення й отримання даних.
<code>error(func)</code>	Встановлює функцію, яка буде викликана у випадку помилки.
<code>close()</code>	Закриває з'єднання.
<code>block(ifok)</code>	Блокує виконання, доки не буде виконано умову.
<code>abort(safecode)</code>	Скасовує виконання дії за безпечним кодом.
<code>send(data, code)</code>	Відправляє дані.
<code>recv(safecode)</code>	Отримує дані.

Особливо важливими є методи `encrypt()` та `decrypt()`, за допомогою яких реалізовується шифрування. Ці методи використовують асиметричне шифрування з використанням приватного та публічного ключів. В основі цих алгоритмів лежать RSA та ОАЕР.

Функція `encrypt()` приймає байтові дані та шифрує їх, використовуючи публічний ключ. Функція `decrypt()` приймає зашифровані байтові дані та розшифровує їх, використовуючи приватний ключ. Для детальнішого ознайомлення наведений код на рисунку 4.2:

```

401     def encrypt(self, data:bytes) -> bytes:
402         try:
403             encrypted_data = self.public_key.encrypt(
404                 data,
405                 padding.OAEP(
406                     mgf=padding.MGF1(algorithm=hashes.SHA256()),
407                     algorithm=hashes.SHA256(),
408                     label=None
409                 )
410             )
411             return encrypted_data
412         except ValueError as e:
413             print(e)
414             raise
415
416     def decrypt(self, encrypted_data:bytes) -> bytes:
417         decrypted_data = self.private_key.decrypt(
418             encrypted_data,
419             padding.OAEP(
420                 mgf=padding.MGF1(algorithm=hashes.SHA256()),
421                 algorithm=hashes.SHA256(),
422                 label=None
423             )
424         )
425         return decrypted_data

```

Рисунок 4.2 – Реалізація методів encrypt та decrypt

Допоміжні функції файлу подані у наступній таблиці 4.3:

Таблиця 4.3 – Допоміжні функції для протоколу

Назва	Опис
extract_text(text, start_char, end_char)	Витягує всі підрядки між start_char і end_char із поданого тексту.
safecode(length)	Генерує випадковий алфавітно-цифровий рядок заданої довжини.
bytes_format(value, space, point)	Форматує значення байта в рядок з одиницею вимірювання.
calc_divisional_range(size, chunk)	Обчислює діапазон поділу.
split_bytes_into_chunks(data, chunk_size)	Розділяє рядок байтів на фрагменти заданого розміру.

4.3 Файл memory.py

У цьому файлі здійснюється управління внутрішньою пам'яттю та зберіганням даних. Основні компоненти цього файлу: клас MemoryLimit, функції управління пам'яттю та класи RowIterator, Database і Dictionary.

Клас MemoryLimit визначає межі використання пам'яті (а саме, м'яку та жорстку межі) і дозволяє отримувати ці значення за допомогою відповідних властивостей «soft» і «hard».

Функції для роботи зі сховищем подані у наступній таблиці 4.4:

Таблиця 4.4 – Функції для роботи зі сховищем

Назва	Опис
init()	Ініціалізує сховище даних у пам'яті
set(key, value)	Встановлює пару ключ-значення в сховищі даних у пам'яті.
get(key, df)	Отримує значення зі сховища даних у пам'яті.
items()	Отримує всі пари ключ-значення в сховищі даних у пам'яті.
keys()	Отримує всі ключі в сховищі даних у пам'яті.
pop(key)	Видаляє та повертає значення зі сховища даних у пам'яті.
clear()	Очищає сховище даних у пам'яті.
setdefault(key)	Встановлює значення за умовчанням для ключа, якщо він не існує.
setlist(keys)	Встановлює значення за замовчуванням для списку ключів, якщо вони не існують.

Клас RowIterator забезпечує ітерацію по рядках у таблиці нашого сховища, використовуючи для цього курсор SQLite, його методи відображені у таблиці 4.5:

Таблиця 4.5 - Методи класу RowIterator

Назва	Опис
<code>__init__(curs, table, query)</code>	Ініціалізує об'єкт класу з заданими параметрами.
<code>__enter__()</code>	Він виконує SQL-запит з використанням курсора та повертає об'єкт класу для подальшого використання.
<code>__exit__()</code>	Використовується для очищення ресурсів та обробки винятків.
<code>__iter__()</code>	Дозволяє використовувати об'єкт класу як ітератор.
<code>__next__()</code>	Повертає наступний рядок результату SQL-запиту.

Клас Database виконує роль управління операціями з базою даних SQLite. Він містить методи для підключення до бази даних, виконання запитів, перевірки наявності колонок у таблиці, створення таблиць, оновлення даних та закриття з'єднання. Функції та методи відображені у наступній таблиці 4.6:

Таблиця 4.6 – Методи класу Database

Назва	Опис
<code>__init__(path)</code>	Ініціалізує об'єкт класу з вказаним шляхом до файлу бази даних.
<code>current_time()</code>	Повертає поточний часовий штамп.
<code>new_uuid()</code>	Генерує новий UUID.
<code>execute_query(query)</code>	Виконує SQL-запит.
<code>check_columns_exist(table, columns)</code>	Перевіряє наявність стовпців у таблиці.
<code>create_table(table_name, fields, key, unique)</code>	Створює таблицю із зазначеними полями.
<code>get_table_rows(table)</code>	Отримує кількість рядків у таблиці.
<code>iter_table_rows(table)</code>	Отримує ітератор над рядками таблиці.
<code>get_all_rows(table)</code>	Отримує всі рядки з таблиці.

Назва	Опис
<code>get_all_rows_one_by_one(table, func)</code>	Отримує всі рядки з таблиці один за одним і застосовує функцію до кожного рядка.
<code>get_table_columns(table)</code>	Отримує назви стовпців таблиці.
<code>get_column_type(table, column)</code>	Отримує тип даних стовпця.
<code>validate_data_types(table, data)</code>	Перевіряє типи даних кортежу щодо схеми таблиці.
<code>insert_data(table, data)</code>	Вставляє дані в таблицю.
<code>convert(table, data)</code>	Перетворює список кортежів на список словників на основі стовпців таблиці.
<code>select_data(table, conditions, iter)</code>	Вибирає дані з таблиці на основі умов.
<code>select_numbler(table, key, start, end, iter)</code>	Вибирає дані з таблиці, де цифровий ключ знаходиться в діапазоні.
<code>delete_data(table, condition)</code>	Видаляє дані з таблиці на основі умов.
<code>build_set_string(data)</code>	Будує рядок SET для запиту UPDATE.
<code>build_condition_string(condition)</code>	Будує рядок умови WHERE для запиту.
<code>update_data(table, data, condition, only)</code>	Оновлює дані в таблиці на основі умов.
<code>close()</code>	Закриває підключення до бази даних.

Клас Dictionary є інтерфейсом для управління даними у вигляді словника з використанням бази даних. Він використовує об'єкт класу Database для зберігання даних у таблиці. Функції та методи наведені у наступній таблиці 4.7:

Таблиця 4.7 – Методи класу Dictionary

Назва	Опис
<code>__init__(db, table)</code>	Ініціалізує об'єкт класу Dictionary.
<code>identify(value)</code>	Визначає тип значення та перетворює його на рядок.
<code>__setitem__(key, value)</code>	Встановлює елемент у словнику.
<code>__getitem__(key)</code>	Отримує елемент у словнику.
<code>__delitem__(key)</code>	Видаляє елемент у словнику.
<code>set(key, value)</code>	Встановлює пару ключ-значення в словнику.
<code>get(key)</code>	Отримує значення зі словника за ключем.
<code>delete(key)</code>	Видаляє зі словника пару ключ-значення.

4.4 Файл endpoints.py

У файлі endpoint.py визначаються класи та функції, які забезпечують обробку мережових запитів за допомогою протоколу Protocol. Ключові компоненти цього файлу – класи Request та Endpoint, що забезпечують зв'язок з мережею, використовуючи протокол Package.

Клас Request представляє мережовий запит, який інкапсулює об'єкт Protocol. Властивості класу подані у таблиці 4.8:

Таблиця 4.8 – Властивості класу Request

Назва	Опис
<code>__init__(data)</code>	Ініціалізує об'єкт класу Request.
<code>code()</code>	Повертає код запиту.
<code>data()</code>	Повертає об'єкт Protocol.
<code>json()</code>	Повертає JSON дані з запиту.
<code>meta()</code>	Повертає метадані з запиту.

`_request` – контекстна змінна, яка зберігає поточний об'єкт «Request». Методи для роботи з цією контекстною змінною описані у наступній таблиці 4.9:

Таблиця 4.9 - Методи для роботи з _request

Назва	Опис
set_request(data)	Встановлює поточний запит у контекстній змінній.
get_request()	Повертає поточний запит з контекстної змінної.

Клас Endpoint призначений для обробки комунікації з мережевим вузлом та використовує клас Package. Методи класу наведені у таблиці 4.10 :

Таблиця 4.10 – Методи класу Endpoint

Назва	Опис
__init__(sende, recver, buff, public_key, private_key)	Ініціалізує об'єкт класу Endpoint.
start(thread)	Запускає кінцеву точку для початку обробки запитів.
send_pack(extn, meta)	Надсилає пакет із зазначеним розширенням і метаданими.
route(extn)	Декоратор для реєстрації функцій обробки маршрутів.
__reg_function()	Реєструє функції, які обробляють специфічні маршрути на основі їх імен.
__recv_handle()	Обробляє вхідні запити в циклі до завершення роботи пакета.
__handle_request(extn)	Обробляє запит на основі його розширення.

4.5 Файли server.py та client.py

Оскільки апаратне забезпечення занадто складно отримати, було прийнято рішення про демонстрацію роботи протоколу на клієнт-серверній програмі та написана реалізація цієї програми у двох відповідних файлах client.py та server.py.

У файлі client.py (рисунок 4.3) реалізовано просту клієнтську програму, яка з'єднується з сервером, надсилає файл відео по частинах (chunks) і обробляє відповіді від сервера. Він використовує сокети для з'єднання і бібліотеку endpoint для обробки запитів і відповідей. Нижче поданий код для наочної демонстрації:

```
client.py > ...
1  import socket
2  import endpoint as ep
3  from rich import print
4
5  get_request = ep.get_request
6
7  def client(video_file):
8      sk = socket.socket()
9      sk.connect(('127.0.0.1', 6666))
10     print("[bold green]Connected to the server[/bold green]")
11
12     class rrr(ep.Endpoint):
13         def on_res(self):
14             print("[bold cyan]Server response: recved[/bold cyan]")
15
16     rs = rrr(sk.send, sk.recv, buff=2048)
17     rs.start(thread=True)
18
19     with open(video_file, 'rb') as f:
20         chunk_size = 1024 # Adjust the chunk size as needed
21         while chunk := f.read(chunk_size):
22             rs.send(ep.Protocol(extension='.say', meta=chunk))
23             print(f"[bold magenta]Sent a chunk of size {len(chunk)}[/bold magenta]")
24
25     sk.close()
26
27 if __name__ == "__main__":
28     video_file = input("Enter the path to the video file: ")
29     client(video_file)
30
```

Рисунок 4.3 – вміст файлу client.py

У файлі `server.py` (рисунок 4.4) реалізовано простий сервер, що використовує сокети для зв'язку і бібліотеку `rich` для виводу кольорових повідомлень у терміналі. Сервер приймає підключення до порту 6666, приймає підключення клієнта й обробляє запити з міткою `.say`, після чого відповідає на них за допомогою класу `Endpoint` з бібліотеки `endpoint`. Нижче наведений код для більш наочного розуміння.

```
server.py > ...
1  import socket
2  import endpoint as ep
3  from rich import print
4  from datetime import datetime
5
6  get_request = ep.get_request
7
8  def server():
9      sk = socket.socket()
10     sk.bind(('0.0.0.0', 6666))
11     sk.listen()
12     print("[bold green]Server listening on port 6666...[/bold green]")
13     conn, addr = sk.accept()
14     print(f"[bold blue]Connected by {addr}[/bold blue]")
15
16     end = ep.Endpoint(conn.send, conn.recv)
17
18     @end.route('.say')
19     def say():
20         data = get_request()
21         current_time = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
22         print(f"[bold yellow]{current_time} - Received data of size: {len(data.meta)}[/bold yellow]")
23         return ('.res', 'recved')
24
25     end.start()
26
27     conn.close()
28     sk.close()
29
30 if __name__ == "__main__":
31     server()
32
```

Рисунок 4.4 – вміст файлу `server.py`

5 ДЕМОНСТРАЦІЯ РОБОТИ

Найпростіша система відеоспостереження, на котрій можна було б протестувати та продемонструвати роботу власного розробленого протоколу передачі даних складається з IP-камери, серверу та клієнту, що зображено на рисунку 5.1 для кращого розуміння. Як було описано вище, отримати IP-камеру, вбудована програма якої підтримує не лише зміну протоколу передачі даних, а й інтеграцію власного, складно, тому було прийнято рішення симулювати передачу пакетів між сервером та клієнтом за допомогою написаних власних простих реалізацій клієнта та сервера.

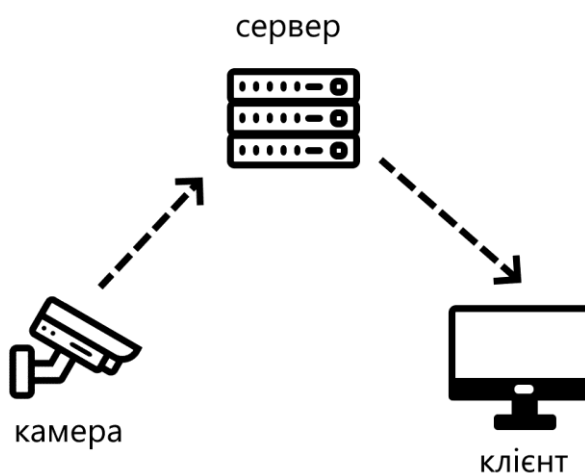


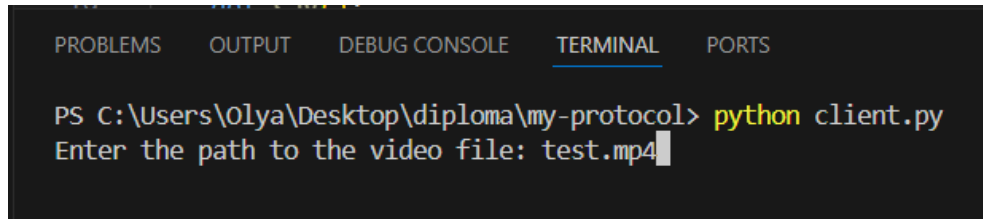
Рисунок 5.1 – Проста система відеоспостереження

У програмному середовищі Visual Studio Code відкриваємо файл `server.py` та виконуємо його, запустивши у першому терміналі. Бачимо повідомлення, що сервер приймає підключення до порту 6666 (рисунок 5.2)

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Olya\Desktop\diploma\my-protocol> python server.py
Server listening on port 6666...
█
```

Рисунок 5.2 – Прослуховування сервером

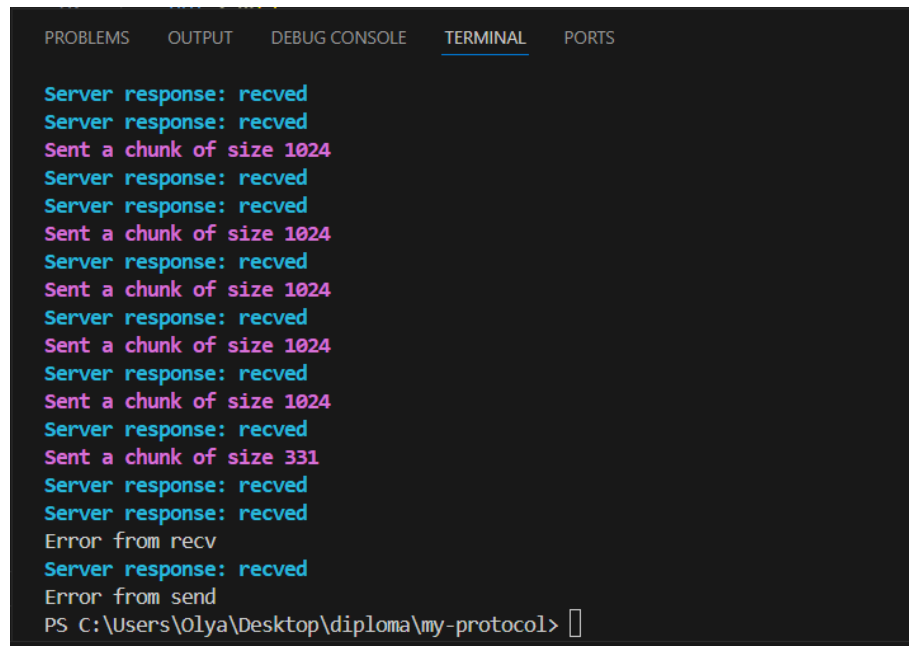
Відкриваємо другий термінал та запускаємо у ньому client.py. Програма виконується та просить ввести назву відеофайлу для відправки на сервер (рисунок 5.3)



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\Olya\Desktop\diploma\my-protocol> python client.py
Enter the path to the video file: test.mp4
```

Рисунок 5.3 – Введення назви відеофайлу

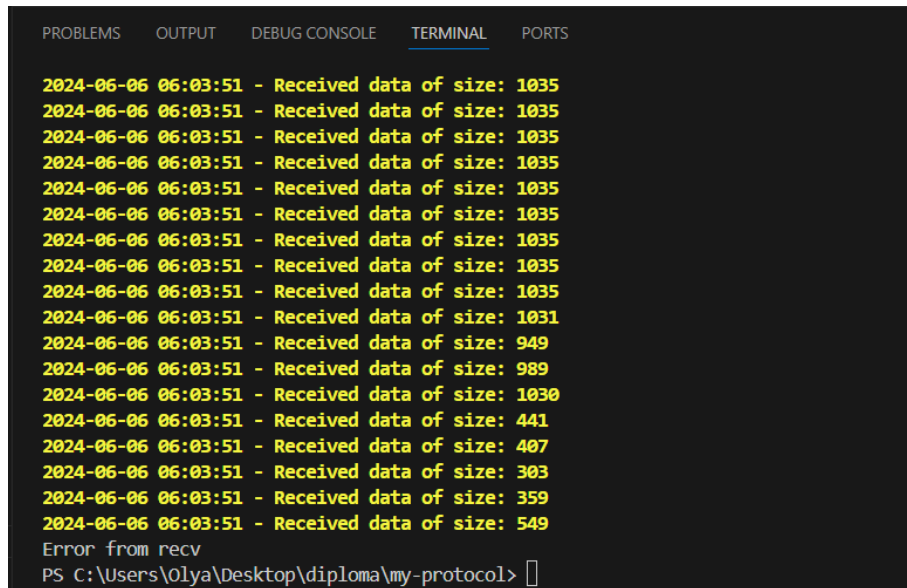
Бачимо, що пакети були успішно відправлені клієнтом на сервер, процес та результат виводиться у терміналі (рисунок 5.4)



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Server response: recved
Server response: recved
Sent a chunk of size 1024
Server response: recved
Server response: recved
Sent a chunk of size 1024
Server response: recved
Sent a chunk of size 1024
Server response: recved
Sent a chunk of size 1024
Server response: recved
Sent a chunk of size 1024
Server response: recved
Sent a chunk of size 331
Server response: recved
Server response: recved
Error from recv
Server response: recved
Error from send
PS C:\Users\Olya\Desktop\diploma\my-protocol>
```

Рисунок 5.4 – Успішна відправка пакетів на сервер

Повертаємося у термінал з відкритим сервером та бачимо успішне отримання пакетів сервером (рисунок 5.5)



The image shows a terminal window with a dark background and yellow text. At the top, there are five tabs: PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (which is selected), and PORTS. The terminal displays a series of log messages indicating data reception. Each message follows the format: '2024-06-06 06:03:51 - Received data of size: [value]'. The values for the size of the received data are: 1035, 1035, 1035, 1035, 1035, 1035, 1035, 1035, 1035, 1031, 949, 989, 1030, 441, 407, 303, 359, and 549. After the last log message, there is a line that says 'Error from recv'. At the bottom of the terminal, the command prompt shows the path 'PS C:\Users\Olya\Desktop\diploma\my-protocol>' followed by a cursor.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1035
2024-06-06 06:03:51 - Received data of size: 1031
2024-06-06 06:03:51 - Received data of size: 949
2024-06-06 06:03:51 - Received data of size: 989
2024-06-06 06:03:51 - Received data of size: 1030
2024-06-06 06:03:51 - Received data of size: 441
2024-06-06 06:03:51 - Received data of size: 407
2024-06-06 06:03:51 - Received data of size: 303
2024-06-06 06:03:51 - Received data of size: 359
2024-06-06 06:03:51 - Received data of size: 549
Error from recv
PS C:\Users\Olya\Desktop\diploma\my-protocol>
```

Рисунок 5.5 – Успішне отримання пакетів сервером

Звідси можемо зробити висновок, що протокол був успішно протестований під час симуляції реальних умов та показав високий результат, відправивши та отримавши пакети з високоефективним кодуванням та швидкістю.

ВИСНОВОК

У цій дипломній роботі було розглянуто та проаналізовано наявні протоколи передачі даних у системах відеоспостереження, зокрема RTSP, ONVIF та стандарти відеокодування H.264/H.265. На основі проведеного порівняльного аналізу було виявлено їхні переваги та недоліки, що дозволило обґрунтувати необхідність розробки власного протоколу передачі даних.

Розроблений власний протокол передачі даних було протестовано у близьких до реальних умов, на створеній симулятивній програмі клієнт-серверу. Протокол продемонстрував високу ефективність та надійність у передачі відеоданих. Було реалізовано додаткові механізми обробки помилок та методи шифрування, що підвищують безпеку переданої інформації.

Загалом, розроблений протокол відповідає сучасним вимогам до систем відеоспостереження, забезпечує високу продуктивність, сумісність з різними типами обладнання та надійність у роботі. Тому подальше покращення та впровадження цього протоколу у реальні системи відеоспостереження може значно підвищити ефективність та безпеку систем відеоспостереження.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. A Brief History of Surveillance Cameras.

URL: <https://www.deepsentinel.com/blogs/home-security/history-of-surveillance-cameras/> (дата звернення: 26.10.2023)

2. J.K. Petersen Handbook of Surveillance Technologies. London : Routledge, 2012;

3. RFC 2326: Real Time Streaming Protocol.

URL: <https://www.rfc-editor.org/rfc/rfc2326> (дата звернення: 07.11.2023)

4. ONVIF Doc Map.

URL: <https://www.onvif.org/specs/DocMap.html> (дата звернення: 08.11.2023)

5. International Telecommunication Union. H.264 : Advanced video coding for generic audiovisual services / Geneva, 2021

6. Taking on the Chinese in Cyberspace.

URL: <https://www.linkedin.com/pulse/taking-chinese-cyberspace-terry-dunlap> (дата звернення: 17.03.2024)