Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування закладу вищої освіти)

Навчально-науковий інститут інформаційних технологій і робототехніки
(повне найменування інституту, назва факультету (відділення))

Кафедра автоматики, електроніки та телекомунікацій
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи

магістр
(ступінь вищої освіти)

на тему **РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ ТА ЙОГО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В УМОВАХ ЄВРОІНТЕГРАЦІЇ**

Виконав: студент 6 курсу, групи 601ТТ
спеціальності 172 «Телекомунікації та радіотехніка
(шифр і назва напряму підготовки, спеціальності)

Карпук В.Ю.
(прізвище та ініціали)
Керівник Жученко О.С.
(прізвище та ініціали)
Рецензент Штомпель М.А.
(прізвище та ініціали)

Полтава - 2022 рік

Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Інститут Навчально-науковий інститут інформаційних технологій і робототехніки
Кафедра Автоматики, електроніки та телекомунікацій
Ступінь вищої освіти Магістр
Спеціальність 172 «Телекомунікації та радіотехніка»

ЗАТВЕРДЖУЮ

Завідувач кафедри автоматики, електроніки та телекомунікацій

_____ О.В. Шефер
“___” _____ 2022 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Карпуку Владиславу Юрійовичу

1. Тема проекту (роботи) **«Розробка протоколу телеметрії для дистанційно-керованої техніки та його програмної реалізації в умовах євроінтеграції»**
керівник проекту (роботи) Жученко Олександр Сергійович, к.т.н., доцент
затверджена наказом вищого навчального закладу від “12 ” 08 2022 року № 544 фа

2. Строк подання студентом проекту (роботи) 07.12.2022 р.

3. Вихідні дані до проекту (роботи) Принципи розробки мережевих протоколів прикладного рівня. Аналізатор мережевого трафіку Wireshark, мова програмування C#, мультимедійна платформа Unity.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Основні теоретичні відомості про мережеві протоколи. Аналіз внутрішньої будови сучасних протоколів телеметрії. Методи розробки прикладних протоколів передачі даних. Реалізація протоколу телеметрії для дистанційно-керованої техніки. Дослідження роботи протоколу телеметрії.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових плакатів):
1) Структурна схема пакету протоколу.
2) Діаграми встановлення та завершення зв’язку.
3) Схеми стану передачі телеметричних даних.
4) Схеми стану передачі команд керування.
5) Розгляд типів пакетів протоколу.

6) Схема взаємодії моделі тестування протоколу.
7) Графічний інтерфейс вікна огляду будови пакетів.
8) Інтерфейс керування у моделі налагоджування.

6. Дата видачі завдання 01.09.2022 р.

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів магістерської роботи	Термін виконання етапів роботи			Примітка (плакати)
1	Огляд теорії прикладних протоколів передачі даних	13.09.22		15%	Пл. 1
2	Аналіз внутрішньої будови та класифікація сучасних протоколів телеметрії	27.09.22	I	30%	Пл. 2
3	Методи та принципи розробки прикладних протоколів передачі даних	10.10.22		40%	Пл. 4
4	Розробка специфікації протоколу телеметрії	17.10.22		50 %	Пл. 5
5	Програмна реалізація протоколу телеметрії	25.10.22	II	60%	Пл. 6
6	Створення моделі для тестування та дослідження роботи протоколу телеметрії	07.11.22		70%	Пл.7,8
7	Оформлення магістерської роботи	07.12.22	III	100%	

Магістрант _____ Карпук В.Ю.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Жученко О.С.
(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

БПА – безпілотний апарат

ОС – операційна система

IDE – інтегроване середовище розробки

GUI – графічний інтерфейс користувача

ШІМ – широтно-імпульсна модуляція

ВСТУП

У сучасному світі все більше використовується дистанційне керування технікою: у розумному будинку, де потрібно вмикати чи вимикати різне обладнання, на виробництві де потрібно дистанційно керувати різними промисловими процесами чи у воєнній сфері для віддалено керованої зброї чи розвідки. Усі це різноманіття обладнання та сценаріїв його використання поєднує одне – наявність протоколу передачі даних для дистанційного керування.

Для початку потрібно розібратися у основних поняттях, що відносяться до заданої теми.

Мережевий протокол зв'язку – набір правил для декількох пристроїв про те як саме вони будуть обмінюватися даними. Якщо декілька приладів мають підтримку одного протоколу, то це дозволяє їм бути сумісними між собою [1].

Зазвичай сучасні мережеві протоколи будуються із оглядом на дві мережеві моделі передачі даних – OSI (7 рівнів) та TCP/IP (4 рівні). Дані моделі виставляють рівні абстракцій обробки даних. Протоколи зв'язку можуть бути реалізовані програмно чи апаратно у вигляді мікросхем, останній спосіб використовується для низькорівневих протоколів, що б підвищити продуктивність обробки даних [1, 2].

У загальному випадку мережевий протокол являє собою опис структури пакету даних, що має двійковий вигляд (байти). Ця структура має заголовок для ідентифікації протоколу, службові дані що потрібно для коректної роботи та власне деяку корисну інформацію. Окрім цього протокол також описує алгоритм встановлення сеансу зв'язку та його завершення.

Дистанційне керування – це такий вид керування пристроєм чи технікою при якому команди керування надходять із фізично віддаленого від пристрою місця. Зазвичай таке передача команд керування відбувається через радіо канал, електричними сигналами через кабель, інфрачервоним чи ультразвуковим випромінюванням.

Телеметричні дані (телеметрія) – такий потік даних, що передається із віддаленого пристрою на пульт оператора, що керує даною технікою. Ці дані можуть включати у себе різні величини: виміряні показники із датчиків, аналогові

та цифрові величини, відео та аудіо потоки. Телеметричні дані дозволяють провести обробку та аналіз для практичного використання кінцевим користувачем, отже таким чином відслідковувати стан дистанційно керованого об'єкту.

Актуальність даної роботи обумовлюється великою кількістю різноманітних дистанційно-керованих приладів. У сучасному світі кожен день з'являються нові прилади, що потребують дистанційного керування. Це можна побачити із розвитком багатьох сфер людства: медичної (телеметрична медицина), воєнної (безпілотники), промислової (маніпулятори), розумні будинки та інше. Всі ці сфери вже використовують чи будуть використовувати дистанційно-керовану техніку. Для забезпечення все більш наростаючої потреби дистанційно-керованих об'єктів необхідно розробляти нові протоколи для взаємодії із ними, тобто протоколи телеметрії.

Метою даної роботи є опис та програмна реалізація протоколу для дистанційного керування технікою. Цей протокол дозволяє передавати типові для дистанційно керованої техніки види даних: команди керування, телеметрії із датчиків, аудіо-відео потоки та службові дані.

Об'єкт дослідження даної роботи є протокол прикладного рівню для телеметрії та дистанційного керування технікою.

Задачі даної кваліфікаційної роботи:

- аналіз внутрішньої будови сучасних протоколів телеметрії;
- виведення методів та принципів розробки прикладних протоколів передачі даних;
- вибір інструментів для розробки протоколу;
- розробка специфікації для протоколу телеметрії, опис його внутрішньої будови;
- розробка моделі тестування та дослідження роботи протоколу.

1 ТЕОРІЯ ТА АНАЛІЗ ПРОТОКОЛІВ ДИСТАНЦІЙНО КЕРОВАНОЇ ТЕХНІКИ

1.1 Загальні відомості про протоколи передачі даних

1.1.1 Мережевий протокол зв'язку

Мережевий протокол зв'язку це – опис правил чи узгоджень обміну інформацією. Такі домовленості дозволяють обмінюватися різним приладам чи програмам повідомленнями із заздалегідь відомою структурою [1]. Отже це необхідний набір правил який утворює формат даних.

В залежності від поставленої задачі протокол може визначати ряд певних вимог чи властивостей. Перелічимо найбільш типові вимоги для мережевих протоколів зв'язку [1, 2]:

- Розмір пакету даних, що передається.
- Мінімальну і максимальну швидкість передачі та її узгодження.
- Можливість корекції помилок при передаванні.
- Процес встановлення зв'язку, так зване рукостискання (handshake).
- Авторизація абоненту.
- Маршрутизація – прокладання шляху трафіку за критеріями.
- Керування потоком передачі даних.

Між відправником та одержувачем повідомлення складається із дискретних блоків, де кожен окремий пакет даних має свій власний заголовок, основну інформацію та траєкторію маршрутизації. Все це має бути синхронізовано та налагоджено до дрібних деталей.

Існує велика кількість протоколів зв'язку, деякі із них реалізовані апаратно, інші програмно. Вони використовуються в будь-яких цифрових та аналогових комунікаціях. Усіх їх об'єднує те, що вони дозволяють різним приладам обмінюватися повідомленнями, тобто протоколи виконують роль стандартизації у сфері телекомунікацій.

1.1.2 Мережеві моделі OSI та TCP/IP

Сучасні комунікаційні протоколи не працюють ізольовано одне від одного та залежать від інших протоколів, ця залежність показана у багаторівневих моделях. Кожен більш високий рівень у моделі залежить від рівнів, розташованих нижче. Найбільш відомими та популярними моделями на сьогоднішній день є дві мережеві моделі: OSI та TCP/IP [3].

Мережева модель OSI є концептуальною структурою, що декларує мережеві або телекомунікаційні системи у вигляді семи рівнів, кожен з яких виконує свою функцію.

Перелічимо рівні мережевої моделі із зазначенням функцій кожного [2, 3]:

1. Фізичний рівень – тут знаходяться всі фізичні представлення мережі: електричні кабелі, радіочастотні канали, тобто усі фізичні з'єднання та обмін сигналами, що утворюють канал зв'язку.
2. Канальний рівень – цей рівень надає можливість передавати дані між двома кінцевими вузлами та обробку виправлень супутніх помилок при передачі.
3. Мережевий рівень – мережевий рівень займається маршрутизацією пакетів даних, що передаються, тобто прокладає найкращий шлях відповідно до заданих критеріїв.
4. Транспортний рівень – на цьому рівні відбувається координація передачі даних між різними кінцевими системами, наприклад: встановлення швидкості, призначення та розміру даних що відправляються.
5. Рівень сеансу – використовується для встановлення сеансу зв'язку між різними мережевими приладами.
6. Рівень представлення – цей рівень займається перетворенням формату даних рівня сеансу в прикладний та навпаки. Прикладом може бути шифрування та дешифрування потоку передачі.
7. Прикладний рівень – взаємодіє безпосередньо із кінцевим користувачем.

Оскільки модель OSI є ідеалізованої мережевою моделлю, то на практиці використовується спрощена модель TCP/IP, яка має чотири рівні. TCP/IP не відповідає моделі OSI безпосередньо.

TCP/IP або поєднує кілька рівнів OSI в один, або взагалі не використовує деякі рівні. Перелічимо чотири рівні TCP/IP [3]:

1. Канальний рівень – цей рівень визначає спосіб відправлення даних, обробляє фізичний процес відправлення та отримання даних та відповідає за передачу даних між програмами або пристроями в мережі.
2. Мережевий рівень – контролює відправлення та переміщення пакетів по мережі, щоб гарантувати, що вони досягнуть пункту призначення.
3. Транспортний рівень – встановлює та підтримує з'єднання та обмін даними між двома пристроями. Це рівень, на якому дані фрагментуються на пакети та нумеруються для створення послідовності пакетів.
4. Прикладний рівень – обробляє дані на рівні програм які використовують TCP/IP для зв'язку одне з одним. Це рівень, з яким зазвичай взаємодіють користувачі, наприклад, системи електронної пошти чи інтернет-браузер. Він поєднує сеансовий, представлення та прикладний рівні моделі OSI.

Взаємозв'язок між мережевими рівнями моделей OSI і TCP/IP та відповідними протоколами можна побачити на рис.1.

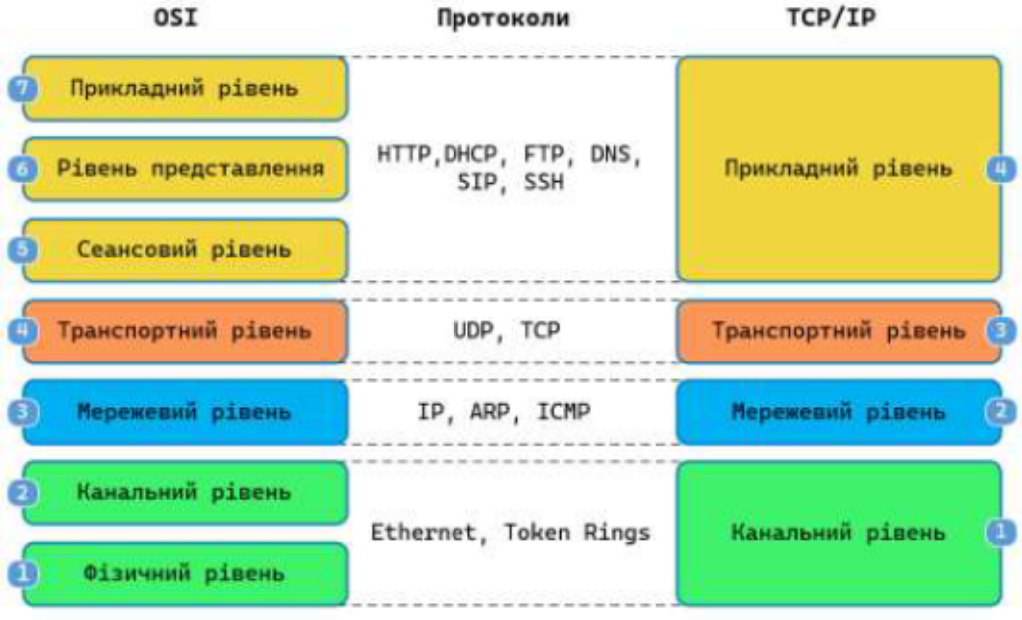


Рисунок 1 – Мережеві моделі OSI та TCP/IP

На кожному із рівнів мережевої моделі інформація представляється у вигляді фрагментів даних протоколу – PDU (Protocol Data Unit).

Процес перетворення даних із вищого рівня у нижчий називається – інкапсуляція, в цьому випадку кожен процес інкапсуляції супроводжується додаванням додаткових даних (заголовків), що потрібні для роботи нижчого протоколу. І навпаки процес перетворення із нижчого рівня у вищий називається декапсуляція, під час цього надлишкова інформація видаляється, оскільки вона вже не потрібна.

1.2 Протоколи транспортного рівня

Оскільки розроблюваний протокол передачі телеметричних даних є високорівневим, тобто прикладного рівня, його основою буде протокол транспортного рівня. Тому доцільно розглянути принцип роботи двох найбільш популярних транспортних протоколів – UDP та TCP.

1.2.1 Протокол TCP

TCP – це протокол, що надає можливості для транспортування даних, тобто їх передачу по мережі. Його головними особливостями є: надійність, цілісність та підтвердження при передачі даних.

TCP передає дані у вигляді потоку даних, який у свою чергу поділяється сегменти. Цей протокол орієнтований на встановлення з'єднання між абонентами перед початком передачі даних та на завершення такого з'єднання після закінчення передачі [4].

Протокол TCP повинен встановити з'єднання перед будь-якою передачею даних. Встановлення з'єднання складається із трьох повідомлень (трьох кратне рукостискання). Цей процес має такий вигляд (рис. 2) [4].

1. Сторона, що хоче встановити з'єднання відправляє другому абоненту повідомлення із бітом синхронізації (SYN), про встановлення з'єднання. У цьому повідомленні вказуєтеся номер порту для з'єднання та номер послідовності, що потрібна для нумерації сегментів під час передачі.
2. Другий абонент у свою чергу відповідає повідомленням із підтвердженням (ACK) встановлення зв'язку та номером своєї послідовності.
3. Сторона, що ініціювала створення з'єднання підтверджує згоду від другого абонента і з'єднання встановлюється.

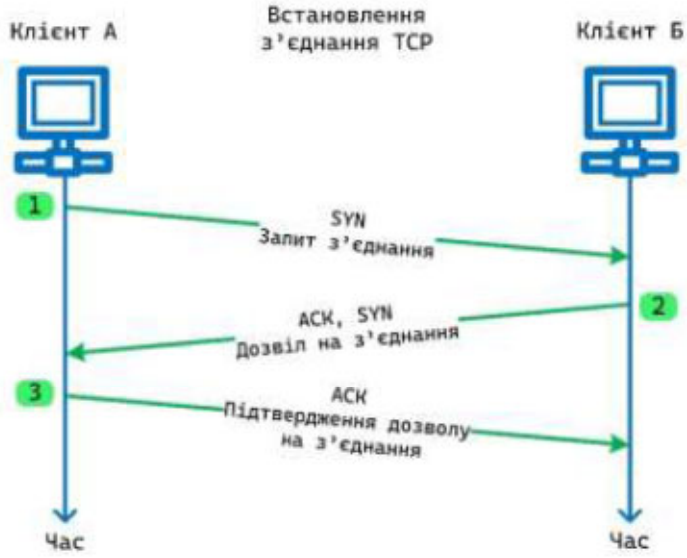


Рисунок 2 – Трьох кратне рукостискання у протоколі TCP

Закінчення передачі даних, повинно завершувати встановлене з'єднання. Цей процес має такий вигляд (рис. 3) [4]:

1. Сторона, що передає потік даних встановлює в останньому сегменті біт (FIN), що означає завершення з'єднання.
2. Друга сторона відповідає першій стороні бітом підтвердження (ACK) про завершення з'єднання. В цей час програмі, що використовує це з'єднання повідомляєтеся про його завершення.
3. Коли програма завершила з'єднання першій стороні відправляєтеся повідомлення про завершення з'єднання бітом (FIN).

4. Перша сторона відправляє другій стороні бітом підтвердження (ACK) у повідомленні, про те, що з'єднання завершилось.

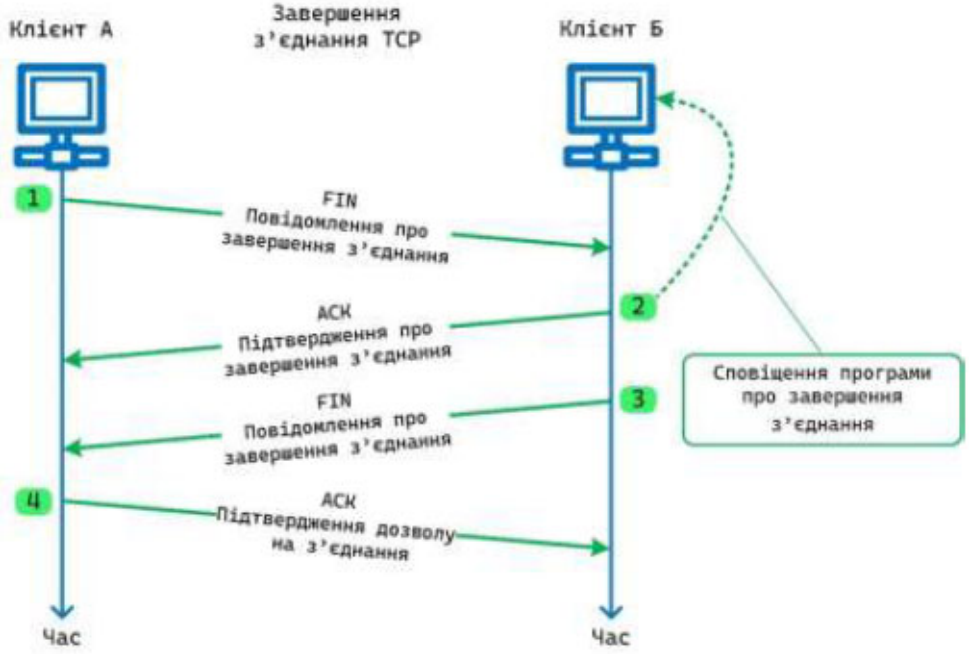


Рисунок 3 – Чотирьох етапний процес завершення з'єднання у протоколі TCP

Протокол TCP гарантує надійну доставлення даних та їх порядок при передачі через мережу. Розглянемо основні властивості протоколу за рахунок чого досягаються вищезгадані характеристики [4, 5].

Гарантія доставлення повідомлень:

- TCP потребує повідомлення із підтвердженням (біт ACK) від приймаючої сторони про отримання кожного сегменту чи групи сегментів які були відправленні. Це реалізація підтвердження про отримання.
- Якщо із будь-яких причин повідомлення про отримання сегменту не було отримано на стороні відправника – відправник через деякий встановлений час спробує знову надіслати дані. Так реалізується повторна відправка.

Керування потоком даних. Для оптимізації передачі даних протокол TCP використовує так зване «вікно» це кількість сегментів, що може бути прийнята приймаючою стороною без підтвердження. За рахунок цього вдається зменшити кількість передавань повідомлень про отримання сегментів.

Порядок слідування сегментів при передачі. При передачі по мережі, приймаюча сторона може отримати декілька однакових сегментів або отримати їх у не правильному порядку. Протокол TCP нумерує кожну порцію даних (байтів) і при передачі включає у сегмент номер першого байту даних котрі в ньому знаходяться. На приймаючій стороні достатньо порівняти сегменти із тими, що є для захисту від дублювання або виставити їх у правильному порядку за номером. Так реалізується механізм захисту від дублювання та порушення слідування даних.

Отже даний протокол має високу надійність при передачі даних, кінцевому абоненту гарантується доставка повідомлень та їх слідування. Проте він має надлишкові витрати на створення з'єднання між абонентами кожний раз коли потрібно щось передати.

1.2.2 Протокол UDP

Протокол UDP – це комунікаційний протокол для передачі даних, що працює на транспортному рівні. Його головними рисами є мінімальна надлишковість при передачі даних, за рахунок відсутності гарантій доставки інформації.

Дані що передаються за допомогою протоколу UDP поділяються на повідомлення і називаються: датаграми або дейтаграми (datagram). На відміну від TCP, протокол UDP не потребує попереднього встановлення з'єднання між хостами у мережі для передачі даних [6].

Протокол UDP має «просту» будову та складається із невеликої кількості елементів:

- Портів відправника та отримувача.
- Довжини датаграми
- Дані (корисне навантаження)
- Контрольна сума

Перелічимо основні особливості протоколу UDP [6, 7]:

- Немає контролю передачі повідомлень. Гарантія доставлення повідомлень та перевірка їх дублікатів відсутня.

- Відсутній контроль упорядкованості даних.
- Відправник не знає про те чи було доставлено повідомлення.
- Передбачається, що усі відсутні механізмі контролю, якщо вони потрібні будуть реалізовані на вищому рівні – прикладному.
- Висока швидкість передачі (порівняно із TCP), оскільки немає додаткових перевірок, встановлення та відстежування з'єднання передачі даних.
- На відміну від TCP який орієнтований на модель «кожен із кожним». Протокол UDP має можливість відправляти дані одночасно багатьом приймачам.

І хоча UPD не має ніяких додаткових механізмів для контролю надійності передачі даних, йому знаходиться застосування. За рахунок менших надлишкових витрат протокол UDP використовують для трафіку у якому втрата декількох повідомлень не так суттєва і більш важливим є час затримки при передачі інформації, наприклад для голосового чи відео потоку у реальному часі.

1.3 Програмний мережевий інтерфейс – сокет

Мережевий сокет – це така програмна абстракція для доступу обміну даними між різними програмними процесами, що мають з'єднання через мережу. Детальну будову і властивості мережевого сокету визначає інтерфейс прикладного програмування – API (зазвичай це визначається типом використовуваної ОС) [8, 9].

Сокет забезпечує:

- Двосторонній зв'язок між програмами, що працюють у мережі.
- Сокет реалізує інкапсуляцію протоколів мережевого і транспортного рівнів, зазвичай це IP, TCP, UDP.
- Сокет містить у собі IP адрес та порт, ця комбінація називається адресом сокету.

Мережеві сокети поділяються на типи [8, 9]:

- Потокові сокети – відправка потоку байтів із гарантованою доставкою. Потоковий сокет забезпечує передачу потоку байтів із гарантією доставки, перевірки правильності їх порядку слідування захист від дублювання даних. Зазвичай потокові сокети використовують протокол TCP.
- Датаграмні сокети – окремі повідомлення без гарантії доставки. Надійність та порядок слідування при передачі даних не гарантується. Зазвичай для такого сокету використовується протокол UDP.
- Сирі сокети (Raw socket) – такі сокети дозволяють прийом та відправлення даних без використання якогось із протоколів. В цьому сокеті на відміну від інших немає автоматичної інкапсуляції із протоколів транспортного рівня, тому також передаються заголовки протоколів для подальшої обробки.

Отже загалом сокет це спосіб взаємодії різних програм між собою через мережу. У своїй суті мережевий сокет є сукупність мережевої адреси і порту.

1.4 Види та класифікація прикладних протоколів передачі даних

Для протоколу керування дистанційною технікою та передачі телеметричних даних, потрібно передавати аудіо та відео дані у потоці та передавати показники датчиків. Беручи до уваги вище сказане, доцільно буде розглянути деякі основні прикладні протоколи, що використовуються для передачі подібних даних.

За призначенням протоколи можна класифікувати [10, 11]:

- Передача потокових відео/аудіо даних – такі протоколи передають набір байтів, що являють собою відео чи звукові сигнали. В цих протоколах зазвичай є елементи керування потоком, його корекція. Вони відзначаються низькою затримкою передачі, що зумовлено контентом, що вони передають.
- Передача деяких показників (значень із датчиків) – ці протоколи передають дані із датчиків у вигляді чисел чи строкового представлення, можуть

обслуговувати велику кількість одночасно працюючих клієнтів у мережі. Характеризуються доволі широким діапазоном можливих затримок, в залежності від пріоритету важливості даних, що передаються.

– Дистанційне керування – призначення цих протоколів передавати команди керування на віддалені об'єкти. Такі протоколи характеризуються можливістю аутентифікації користувача та надійністю при передачі даних. Зазвичай вони повинні мати мінімально можливу затримку передачі.
– Універсальні протоколи – ці протоколи маючи гнучку структуру можуть виконувати широкий спектр завдань.

Протоколи класифікуються за типом даних, які вони передають [11]:

– Текстові – передають корисні дані у вигляді набору символів. Всі дані, що не являються текстом, потрібно конвертувати у строковий формат. Із переваг такого протоколу можливість легко налагоджувати, оскільки такий протокол легко читати людині. Текстовий протокол дозволяє не залежати від кінцевої реалізації конкретних типів даних, що надає більшу сумісність. Але такий тип протоколів не такий швидкий як бінарний оскільки потребує додаткового часу кожний раз коли конвертується тип у строкове представлення та пакети, що передаються мають надлишкову інформацію.
– Бінарні – такі протоколи передають інформацію у вигляді набору байт, що являють собою представлення об'єктів, що треба передати. Бінарні протоколи швидше текстових, оскільки не потребують конвертації типів у строкове представлення та мають набагато менше надлишкової інформації. Проте недоліком є більш складна їх реалізації та потреба сумісності типів на кінцевих приладах.

1.5 Аналіз та класифікація існуючих типів протоколів для передачі телеметричних даних та дистанційного керування

1.5.1 Протоколи дистанційно керованої техніки

Сучасні протоколи для дистанційно керованої техніки поділяються на дві групи [12]:

– RX – внутрішній зв'язок. Ті які призначені для передачі даних від приймача до контролера техніки.

– TX – зовнішній зв'язок. Для передачі між пультом керування та приймачем техніки.

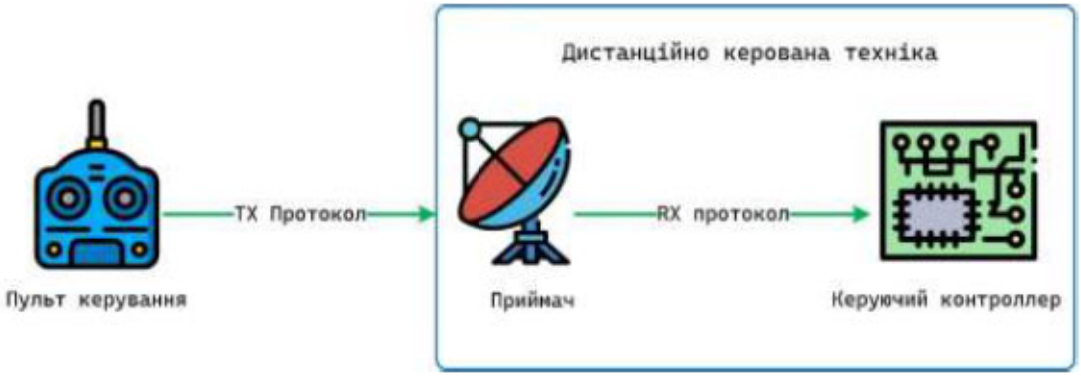


Рисунок 4 – Взаємодія TX та RX типів протоколів для віддалено керованої техніки

1.5.2 Внутрішні протоколи дистанційної техніки

Зазвичай внутрішні протоколи (RX) протоколи це дротова лінія зв'язку, що фізично розташована у дистанційно керованій техніці між приймачем сигналу та керуючий контролером.

Перелічимо основні види RX протоколів [12, 13]:

– PWM – керування за допомогою широтно-імпульсної модуляції (ШІМ). Це один із найрозповсюджених протоколів керування. Для керування потрібен по одному проводу для кожного виконавчого механізму (наприклад двигун). Перевагою даного способу є відносно легка реалізація даного протоколу та його сумісність із обладнанням. Основним недоліком є велика кількість з'єднань до кожного актуатора.

– PPM – управління через фазово-імпульсну модуляцію (ФІМ). Являє собою передачу набору сигналів ШІМ по одному й тому самому дротові, але із різною модуляцією. Перевагою даного способу є використання значно меншої кількості дротів. Оскільки по одному проводу можливо передавати до 8 керуючих сигналів.
– Послідовні протоколи, які передають послідовні серії керуючих сигналів. Такі протоколи повністю цифрові. Яскравим прикладом таких протоколів є UART, I2C, SPI, CAN та інші.

1.5.3 Зовнішні протоколи керування БПА

Зовнішні протоколи (ТХ) потрібні для керування дистанційною технікою, для її зв'язку із наземною станцією чи іншими безпілотними апаратами [13].

Дистанційне керування БПА може бути:

– Ручним – керування апаратом, повністю покладене на оператора.
– Допоміжним – оператор задає основний маршрут чи список дій та за необхідності проводить корекції під час виконання.
– Повністю автономне – втручання оператора потрібно лише за наявності непередбачуваних чи аварійних ситуацій.

Однім із видів керування дистанційною технікою у сучасному світі є аналогові радіоканали:

– На частотах 27 МГц та 35 МГц – для старі види радіозв'язку. Із переваг велика дальність, проте велика чутливість до взаємних завад.
– Частоти: 433/868/915 МГЦ – також один із застарілих видів зв'язку, проте все ще популярний.
– На частотах 2.4 ГГц та 5.8 ГГц – сучасні цифрові протоколи зв'язку.

Перелічимо одні із основних зовнішніх (ТХ) протоколів для керування дистанційною технікою. Аналогові протоколи дистанційного керування [13, 14]:

– Широтно-імпульсна модуляція (PWM).
– Фазово-імпульсна модуляція (PPM).
– Імпульсно-кодові модуляція (PCM).

Аналогові протоколи легкі в реалізації, мають прості апаратні декодери, мають дуже високу швидкодію та низьку затримку. Із недоліків мають вплив інтерференцій та обмежену кількість каналів керування.

Цифрові протоколи дистанційного керування:

- SBUS – підтримує до 18 ШІМ каналів керування закодованих в один канал (радіочастоту).
- DSM2, DSMX – один із найпопулярніших протоколів. Підтримує до 98 каналів. Працює на частоті 2.4 – 2483 ГГц. Має основний та резервний канал, для уникнення взаємних завад та можливість корекції помилок.
- Satellite – цей протокол є розширенням протоколу DSM. Головною особливістю є покращення дальності керування за рахунок використання додаткових антен.

Цифрові протоколи із підтримкою дуплексної передачі – команд керування та телеметричних даних [12, 13, 14]:

- Wi-Fi – цей протокол разом із транспортними протоколами TCP/UDP використовують для віддаленого керування технікою. Він потребує порівняно значної кількості енергії при передачі, має не великий радіус дії (до 150 метрів), проте доволі легко реалізується на сучасному обладнані. Головною перевагою є те, що в ролі терміналу керування можливо використовувати мобільні телефони та ноутбуки, оскільки вони вже мають підтримку цього протоколу.
- Bluetooth – має гарну підтримку різноманітним обладнанням. Із головних недоліків не можливість передачі великої кількості трафіку та обмежена дальність дії (10-20 м)
- IBUS – підтримує передачу у двох напрямках (дуплексний). Дозволяє передавати як керуючі команди так і телеметричні дані із БПА. Підтримує до 16 телеметричних показників. Передає дані у бінарному вигляді.

- MSP – текстовий протокол. На прикладному рівні використовує послідовний протокол UART. Є дуплексним, тобто дозволяє передавати і телеметричні дані. Має можливість віддаленої конфігурації БПА, налаштування плану автономного пересування.
- FrSky Telemetry – сумісний із послідовним інтерфейсом (UART). Радіус дії приблизно 1 км (обумовлено макс. потужністю сигналу). Бінарний протокол із відкритою специфікацією. Підтримує обмежену кількість визначеного набору датчиків.
- DJI Tello – протокол, що працює над протоколом Wi-Fi використовуючи в якості транспортного протоколу UDP. Підтримує режими підключення як: «один із одним» та «одна станція управління – багато БПА». Протокол використовує текстове представлення даних. Може передавати: команди керування, телеметрію та потокове відео.

1.5.4 Передача потокових медіа даних

Для передачі аудіо чи відео сигналу найбільше розповсюдження мають аналогові протоколи, оскільки вони мають найменш можливу затримку. Проте із збільшенням дальності зв'язку збільшуються завади у аналогових системах зв'язку.

Перерахуємо основні особливості передачі відео від БПА до терміналу керування [15]:

- Сучасні протоколи передачі відео працюють на частоті 5.8 ГГц та іноді на 2.4 ГГц.
- Аналогові формати відео – найбільш популярні у FPV керуванні. В якості відео формату використовують ТВ формати NTSC та PAL.
- Цифрові формати передачі даних – MJPEG, MPEG потоки (H264/H265). Зазвичай цифрові формати працюють добре, проте мають доволі високу затримку при передачі (іноді до 1с). Але із збільшенням дальності передачі вони мають перевагу за рахунок кодування та відновлення втрат при передачі.

– Основні види сучасних роздільних здатностей для відео-потоку – це 720p, 1080p, 4k, 5k та 8k. Зазвичай до оператора передається відео гіршої якості (із обмежень можливостей передачі), більш якісне записуєтеся на вбудовану у БПА флеш пам'ять.

1.6 Внутрішня будова сучасних протоколів телеметрії та управління БПА

1.6.1 Протокол ТЛІ Tello

Даний протокол є розробкою виробника ТЛІ. Він працює поверх Wi-Fi мережі та використовує UDP дейтаграми в якості транспорту. Стандартний порт для його роботи 8889 [16].

Розглянемо детальніше структуру Tello пакету у табл. 1.1:

Таблиця 1.1 – Структура пакету протоколу ТЛІ Tello

Позиція (байт)	Опис даних	Додаткова інформація
0	Заголовок	Це завжди 0xCC
1-2	Загальний розмір пакету	13-бітне значення
3	Контрольна сума заголовку CRC-8	Від заголовку до розміру пакету
4	Тип пакету	Біти типу: F;T;TYP;SUB
5-6	Ідентифікатор повідомлення	
7-8	Номер послідовності	
9...	Корисне навантаження (дані)	Опціонально
Кінець	Контрольна сума усього пакету CRC16	Від заголовку до кінця корисного навантаження

Опишемо значення деяких елементів у структурі даних пакету Tello [16]:

– Тип пакету – це може бути запит для отримання інформації, декілька типів даних чи тип встановлення конфігурації.

– Ідентифікатор повідомлення – конкретне призначення корисного навантаження, наприклад: налаштування Wi-Fi чи відео, команда почати рух, записати дані в журнал, відкалібрувати датчики, тощо.

Дані можуть передаватися у обох напрямках. Дані діляться на фрагменти, а ті кожен у свою чергу на 8 чанків (шматків). Кожний фрагмент, тобто 8 чанків даних передаються і потребують підтвердження, що були доставлені, або будуть відправлені знов.

1.6.2 Протокол MAVLink

MAVLink – це сучасний відкритий протокол для зв'язку із БПА. Він підтримує з'єднання типу «Публікація-підписка» та «Точка-точка». Цей протокол має невелику надлишковість оскільки розмір мінімального повідомлення є 8 байтів (службові дані) та 14 байтів для MAVLink 2 версії [17].

Опишемо основні характеристики протоколу MAVLink [17]:

- Підтримка керування до 255 апаратів та іншого обладнання.
- Працює на широкому спектрі обладнання: мікропроцессори ATMega, ARM7, dsPic, STM32 та ОС Windows, Linux, MacOS, Android і iOS.
- Висока надійність протоколу, він може відкидавати помилкові пакети та має гарний алгоритм контрольної суми для виявлення пошкоджень.
- Забезпечує безпеку у моделі «підписок» на повідомлення. Оскільки дозволяє провести аутентифікацію. Проте він не забезпечує шифрування трафіку.

Розглянемо детальніше структуру MAVLink 1 пакету у табл. 1.2:

Таблиця 1.2 – Структура пакету протоколу MAVLink 1

Позиція (байт)	Опис даних	Додаткова інформація
0	Заголовок	Це завжди 0xFE
1	Довжина навантаження	
2	Номер послідовності	
3	Ідентифікатор системи	Ідентифікатор дистанційної техніки у мережі
4	Ідентифікатор компоненту	Ідентифікатор компоненту у техніці
5	Ідентифікатор повідомлення	
6…	Корисне навантаження (дані)	
Кінець	Контрольна сума CRC-16	Контрольна сума пакету без заголовку

MAVLink не описує якими апаратними чи програмними засобами буде відправлено повідомлення, це може бути послідовні протоколи чи TCP/UPD повідомлення.

1.6.3 Протокол UAVCan

UAVCan – це відкритий протокол для керування та телеметрії БПА побудований на основі промислового протоколу CAN [18].

Перелічимо основні характеристики протоколу UAVCan [18]:

- Усі вузли в мережі мають однакові права, тобто немає єдиної точки відказу.
- Підтримка резервних вузлів.
- Із за простої логіки, низькі вимоги до обчислювальна ресурсів.
- Може працювати у жорсткому режимі реального часу, тобто низька затримка і висока пропускна здатність.

UAVCan представляє собою децентралізовану мережу, де кожен вузол має власний ідентифікатор. Вузли можуть обмінюватися двома типами повідомлень: широкомовні повідомлення (підписка) та службові повідомлення (запити).

Основний механізм спілкування між вузлами у мережі UAVCan є спосіб використання широкомовних розсилок. Оскільки кадр UAVCan інкапсулюється у CAN опишемо лише кадр пакету широкомовного повідомлення у табл. 1.3:

Таблиця 1.3 – Структура кадру широкомовного повідомлення протоколу UAVCan

Позиція (байт)	Опис даних	Додаткова інформація
0	Ідентифікатор джерела	Номер вузла-ініціатора
1-2	Тип повідомлення	
3	Пріоритет	

Створення запитів, тобто запуск служб у мережі UAVCan виконується через службові повідомлення. Опишемо лише кадр службового повідомлення у табл. 1.4:

Таблиця 4 – Структура кадру службового повідомлення протоколу UAVCan

Позиція (байт)	Опис даних	Додаткова інформація
0	Ідентифікатор джерела	Номер вузла-ініціатора
1	Ідентифікатор призначення	Номер кінцевого вузла
2	Тип служби	
3	Пріоритет	

1.6.4 Протокол UranusLink

UranusLink це пакетно-орієнтований протокол, для управління дистанційно керованою технікою. Протокол був розроблений із урахуванням втрати даних чи отримання неправильних даних [19].

Опишемо структуру пакету службового повідомлення UranusLink – табл. 1.5:

Таблиця 1.5 – Структура кадру службового повідомлення протоколу UranusLink

Позиція (байт)	Опис даних	Додаткова інформація
0	Заголовок	Це завжди 0xFD
1	Номер послідовності	
2	Ідентифікатор повідомлення	
3	Довжина даних	
4	Корисні дані	
5	Контрольна сума	

Опишемо деякі відмінності протоколу UranusLink [19]:

- Номер послідовності це діапазон від 0x0000 до 0xFCFE (оскільки це заголовок). Послідовність збільшується на 2 після відправлення кожного наступного пакету.
- З'єднання від БПА до станції керування може передавати: усі потокові дані (аудіо, відео) та дані с датчиків.
- З'єднання від станції керування до БПА може передавати команди керування та команди конфігурації.
- UranusLink має на 33% менше надлишкових витрат ніж MAVLink.

ВИСНОВОК

У першому розділі було розглянуто основну теорію протоколів передачі даних. В особливості призначення та опис мережевого протоколу зв'язку та мережеві моделі, що використовуються в якості еталону при розробці протоколів – OSI та TCP/IP. Були описано будову та принцип роботи двох найбільш популярних протоколів, що використовуються на транспортному рівні передачі даних – TCP та UDP.

Із огляду на подальшу розробку, було описано будову та призначення сокету, що виконує роль мережевого інтерфейсу для програмного забезпечення.

Було проведено класифікацію прикладних протоколів передачі даних, що використовуються у дистанційно керованій техніці, за призначенням та типом даних, що передаються.

Протоколи, що використовуються на дистанційній техніці були поділені на дві групи – внутрішнього та зовнішнього зв'язку. У ході проведення аналізу над сучасними телеметричними протоколами були виявлені їх основні особливості при роботі із різним видом трафіку. Також було розглянуто внутрішню будову популярних протоколів телеметрії та керування БПА.

Завдяки аналітичній роботі, що була проведена над існуючими протоколам передачі даних, зокрема тих що використовуються для роботи із БПА можливо розпочати опис та розробку майбутнього протоколу у наступному розділі.

2 РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ

2.1 Методи та принципи розробки прикладних протоколів передачі даних

2.1.1 Клієнт-серверна взаємодія

У сучасному світі розробки протоколів існує концепція клієнт-серверної архітектури, яка полягає у розмежуванні зон відповідальності, тобто у розділенні функцій клієнта та сервера. Наприклад, ми поділяємо нашу систему так, що клієнт (припустимо, це мобільний застосунок) реалізує лише функціональну взаємодію із сервером. При цьому сервер повинен реалізовувати у собі логіку зберігання даних та складні операції із ними чи суміжними системами.

Для початку потрібно розібратися із основними поняттями: клієнт та сервер [20]. Клієнт це віддалений пристрій, що відправляє запити на віддалений сервер отримуючи у відповідь якусь інформацію, послуги чи запуск інших програм. Сервер це теж віддалений пристрій, який прослуховує мережу на відомому порті та у разі виникнення запитів у ньому, відповідає клієнту відповідно до додаткової інформації, що містилась у запиті. У загальному випадку, що клієнт, що сервер являють собою спеціальне програмне забезпечення, призначення якого передавати та отримувати дані через мережу, використовуючи якісь правила обміну (протокол).

Особливості клієнт-серверної взаємодії: [20, 21]

- Реалізація клієнту та серверу потребують різної кількості апаратних та програмних ресурсів, що актуально для мікроконтролерів, які широко використовуються у БПА.
- Велика можливість масштабування передачі: збільшення клієнтських приладів чи перехід на більш потужний сервер або їх кластер.
- Клієнтська або серверна програмна реалізація безпосередньо взаємодіє з протоколом транспортного рівня для встановлення зв’язку та надсилання і отримання даних.

– Існує вірогідність відмови сервера, як єдиної точки відмови, про те це можна компенсувати наявністю реалізації серверу та клієнта на кожному приладі.

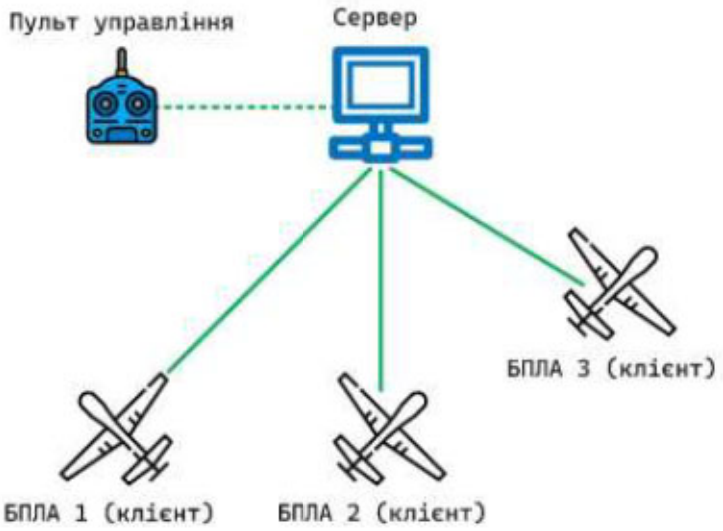


Рисунок 2.1 – Приклад клієнт-серверної архітектури для БПА

Взаємодія між сервером та клієнтом може відбуватися декількома способами розглянемо їх нижче.

Спосіб передачі даних типу запит-відповідь – при такій взаємодії кожен клієнт відкриває пряме з'єднання з кожним сервером, створюючи запит, сервер повинен відповісти, вклавши у відповідь корисні дані, що можуть бути різного формату.

Запити поділяються на два типи: ті які зберігають стан запиту (Stateful) та ті запити що не зберігають стан (Stateless) [22].

– Запит без зберігання стану – це запит при якому сервер не зберігає інформацію (контекст) про сесію з клієнтом. Він повинен у кожному новому запиті отримувати всю інформацію що необхідна для обробки. Такі види запитів є популярним рішенням і використовується наприклад у протоколах HTTP та IP.
– Запит із зберіганням стану – такий запит при якому сервер зберігає контекстну інформацію про сесію із клієнтом. Даний підхід дозволяє зменшити кількість переданої інформації через мережу, ціною ускладення архітектури сервера. Прикладом використання такого типу запитів є протокол FTP.

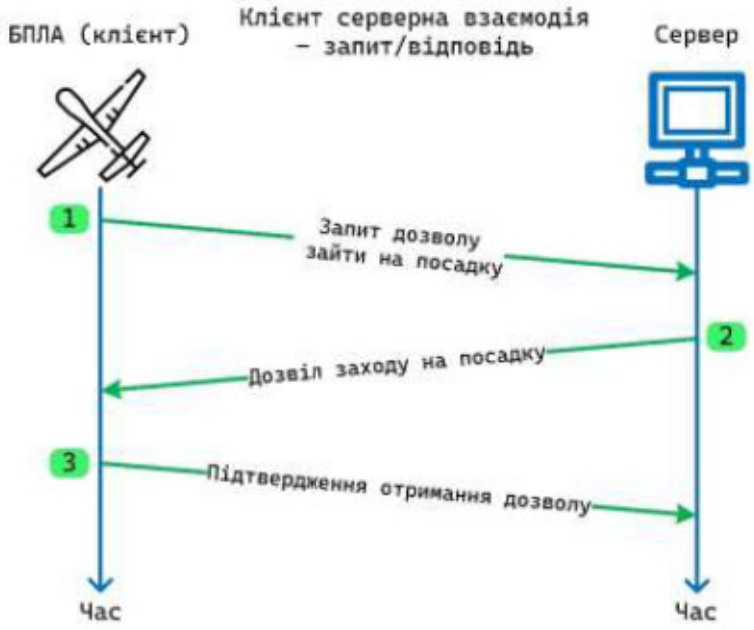


Рисунок 2.2 – Приклад клієнт-серверної взаємодії запит-відповідь

Іншим підходом до передачі даних є потокова передача. При цьому сервер передає нові дані по мірі їх надходження через відкритий сокет, таким чином надаються дані реального часу. Таким способом добре передаються відео та аудіо потоки оскільки такий вид трафіку дуже критичний до часу затримки і немає коли відкривати запити та очікувати на підтвердження їх отримання – коли воно прийде дані вже будуть не актуальні.

Однією з головних відмінностей потокової передачі, є те, що вона мають набагато меншу гнучкість щодо типу вмісту порівняно із взаємодією типу запит-відповідь. Оскільки початковий запит містить усю інформацію, необхідну для обробки цього запиту.

2.1.2 Сеанс зв'язку

Сеанс зв’язку – це обмін даними через мережу, який відбувається в інтерактивному режимі між двома пристроями, що беруть участь у цьому сеансі зв’язку. Сеанс встановлюється на певний час, а потім завершується [23]. Установлений сеанс зв’язку може включати більше одного повідомлення в кожному напрямку. Сеанс зазвичай має режим із збереженням стану, що означає, що принаймні одна зі сторін, що спілкуються, зберігає інформацію про поточний

сеанс, на відміну від зв'язку без збереження стану, де зв'язок складається з незалежних запитів та відповіді.

Для встановлення з'єднання сеансу зв'язку зазвичай використовується спеціальний процес «рукостискання», який використовується для синхронізації версій та типів протоколів для кожної сторони. Також процес з'єднання налаштовує додаткові параметри, що відрізняються відповідно до типу протоколу.

Опишемо приклад процесу встановлення з'єднання [23]:

1. Спочатку клієнт відправляє пакет на сервер який вказує, що він хоче встановити з'єднання.
2. Потім сервер відповідає повідомленням, вказуючи на те, що він бажає та готовий продовжити з'єднання із клієнтом.
3. Нарешті, клієнт відправляє назад пакет, що вказує, що з'єднання слід вважати встановленим та працюючим.

З'єднання може бути завершено різними способами, але зазвичай це відбувається у двох випадках коли одна із сторін вирішила завершити з'єднання чи коли одна із сторін різко зникла (із за різних причин).

Приклад процедури завершення з'єднання виглядає так:

1. Відправка пакету першою стороною про завершення з'єднання однією із сторін.
2. Відправка пакету другою стороною про підтвердження роз'єднання.
3. Відправка першою стороною пакет підтвердження про завершення з'єднання. Звільнення лінії зв'язку (порту чи сокету).

2.1.3. Бітова структура пакету

Мережевий пакет — це базова одиниця даних, яку описує протокол. Загалом пакет має три частини: заголовок пакету, корисне навантаження та службові дані.

Опишемо загальні бітові поля, що утворюють структуру пакету [24]:

– Заголовок – це свого роду «мітка», яка надає інформацію про вміст пакета, його походження та призначення. Зазвичай тут описується тип протоколу, що б відрізняти даний пакет від пакетів інших протоколів.

- Блок довжини даних – це довжина усього пакету чи блоку корисних даних, виміряна в байтах. Цей блок не використовується якщо наперед відомий розмір пакету.
- Службові бітові поля – різні прапорці для передання службової інформації про те як потрібно обробляти пакет.
- Контрольна сума корисних даних чи заголовку – дозволяє визначати пошкодження пакету при його передачі.
- Корисне навантаження – містить інформацію, що потрібно передати

2.1.4 Формат даних

Хоча протоколи передачі даних і передають дані на нижчих рівнях у двійковому вигляді, проте мають різне представлення даних на своєму рівні абстракції. Тобто вигляд даних із якими власне працює протокол, поділяється на бінарне та текстове представлення [25].

Двійкові структури представляють дані у вигляді бінарних кодів, саме так як вони зберігаються в комп'ютері. Прикладами бінарних протоколів є RTP, TCP, IP. Опишемо основні особливості при використанні двійкових структур для передачі даних:

- Двійкове представлення важко читати людиною без використання додаткових програм. Тобто важче проводити налагоджування.
- Підготовка таких даних до передачі відбувається дуже швидко, оскільки не потрібно перекодовувати дані.
- Можливість більш простої організації передачі, оскільки не потрібно додатково владувати дані у якийсь інший строковий формат.
- Більш компактний формат передачі, оскільки дані передаються у тому вигляді в якому вони є насправді.
- Складно підтримувати сумісність оскільки, типи даних можуть відрізнятися.
- Дані передаються без надлишкової інформації, про те що ці дані означають, тому потрібна окрема програма для їх декодування.

Текстові формати мають представлення даних у вигляді масивів символів. Прикладами текстових протоколів є HTTP, SIP. Опишемо основні особливості при використанні текстових форматів для передачі даних:

- Строкове представлення може бути прочитане та зрозуміле для людини. Тобто передачу таких дані легше налагоджувати.
- Мають місце додаткових накладні витрати на перекодування даних.
- Прикладом текстових форматів для даних є JSON та XML.
- Бінарні дані можна конвертувати у строкове представлення за допомогою додаткового кодування такого як Base64.
- Повідомлення збільшуються у розмірі із за текстового представлення.
- Використання строк дає можливість однаково декодувати різні типи даних на різних архітекторах. Тобто це дає кросплатформність.
- Дані передаються у парах ім'я-значення, що дозволяє провести декодування без використання окремої програми.

2.1.5 Гарантія доставки повідомлень

При передачі по мережі через низку різних причин, повідомлення можуть не прийти до отримувача. Кожен протокол сам визначає послідовність дій у таких випадках, але є декілька популярних варіацій гарантії доставки повідомлень [26]:

- Повідомлення відправляється без підтвердження отримання. Має найменші накладні витрати, оскільки цей режим не створює додаткових пакетів підтвердження для надісланих або доставлених повідомлень. Залежно від надійності мережі деякі повідомлення можуть бути втрачені. Такий тип передачі використовується для телеметрії, показників які швидко замінюються та/або де повідомлення надсилаються з високою швидкістю.
- Повідомлення відправляється та потребує підтвердження кожен раз коли отримується, до тих пір повідомлення бути надсилатися повторно. Спричиняє невеликі накладні витрати через додаткові пакети підтвердження. Такий тип передачі пакетів використовується для надсилання важливих даних, наприклад команд чи транзакцій.

Опишемо послідовність передачі повідомлень із гарантією відправки:

1. Відправник надсилає сегмент даних.
2. Якщо отримувач отримав сегмент даних він надсилає повідомлення про отримання даних.
3. Якщо відправник не отримав підтвердження про отримання, він буде повторно пробувати відправити повідомлення, до тих пір поки не отримає підтвердження або повідомить програмі про помилку при передачі.

Також протокол передачі даних, може містити додаткову інформацію у кожному пакеті – номер послідовності, що б мати змогу контролювати черговість повідомлень та їх правильний збір на прийомній стороні.

2.1.6 Фрагментація пакетів – MTU

Максимальна одиниця передачі (MTU) – він визначає найбільший розмір пакету, який можна передати у мережевому з'єднанні. Розмір MTU визначає кількість даних, які можна передати в байтах по мережі. Це обмеження визначає протокол канального рівня [27].

Пакети які перевищують MTU, розбиваються на менші частини, щоб вони могли бути переданими. Цей процес називається фрагментацією. Фрагментовані пакети збираються в одне ціле, коли вони досягають місця призначення.

Фрагментація пакетів має свої недоліки:

- Зменшення продуктивності мережі та кінцевого обладнання, оскільки зростає навантаження на процесори та канал мережі із за більшої ніж потрібно кількості кадрів.
- Деякі мережеві екрани можуть блокувати фрагментацію пакетів.
- Якщо хоча б один фрагмент пакету під час передачі не дійде, знадобиться відправляти знову увесь пакет.
- Фрагментація не підтримується у IPv6.

Тому рекомендується на перевищувати без потреби розмір MTU для пакетів прикладних протоколів. Загалом максимальне значення для Ethernet має 1500 байт.

2.1.7 Виявлення помилок при передачі

Іноді при передачі по мережі виникають бітові помилки у вмісті пакетів. Це відбувається, наприклад, через електричні перешкоди або тепловий шум. Хоча помилки трапляються не часто, особливо на оптичних лініях зв'язку, потрібен певний механізм для виявлення цих помилок, щоб можна було вжити заходів для їх виправлення.

Основна ідея будь-якої схеми виявлення помилок полягає в додаванні надлишкової інформації до пакету, яку можна використовувати для визначення наявності помилок. Одним із механізмів виявлення помилок є використання циклічного надлишкового коду (CRC) [28]. Це такий алгоритм що дозволяє знайти контрольну суму за допомогою якого можна визначити наявність чи відсутність помилок при передачі даних, тобто перевірити їх цілісність.

Популярним рішенням є додавання поля контрольної суми у пакет. Опишемо процес дій при передачі пошкодженого пакету даних:

- Відправник надсилає пакет даних із контрольною сумою.
- При проходженні через мережу пакет пошкоджується та він отримує декілька бітових помилок.
- При отриманні такого пакету отримувач звіряє відповідність контрольної суми, вона показує, що пакет був пошкоджений. Отримувач надсилає повідомлення про повторну передачу пакету.

2.1.8 Шифрування трафіку

Шифрування використовується для захисту даних, що передаються між двома пристроями. Шифрування здійснюється за допомогою алгоритмів шифрування. Ці алгоритми виконують усі криптографічні операції з даними, використовуючи ключ шифрування. Сенс протоколу шифрування полягає у виконанні певної функції над даними, під час якого вони змінюються так, що не від них немає користі, тільки якщо їх розшифрувати [29].

Шифрування буває симетричне та асиметричне:

- Симетричне шифрування є простою формою шифрування. Під час нього використовує один ключ для шифрування даних. Ключ створюється та надається як відправнику, так і одержувачу повідомлення. Інформацію в повідомленні зашифровано за допомогою симетричного ключа, тобто єдина особа, яка може прочитати ці дані, — це той, хто володіє ключем шифрування. Коли повідомлення досягає одержувача, він може використовувати симетричний ключ для розшифровки даних.
- Асиметричне шифрування є більш складним видом шифрування. При цьому створюється пара ключів, яка складається з відкритого та закритого ключів. Відкритий ключ доступний для перегляду будь-кому, тоді як закритий ключ відомий лише автору пари ключів. Щоб асиметрично зашифрувати дані, автор пари ключів шифрує повідомлення своїм особистим ключем, надсилає зашифроване повідомлення одержувачу, а одержувач може використовувати відкритий ключ, який зазвичай знаходиться в сховищі відкритих ключів, щоб розшифрувати повідомлення. Розшифровуючи повідомлення за допомогою відкритого ключа, одержувач даних може визначити, що повідомлення надійшло від того, від кого, на його думку, воно надійшло, і що дані в повідомленні не були змінені. Якщо дані в повідомленні були змінені, розшифровка за допомогою відкритого ключа не дасть читабельного повідомлення, оскільки дані були б зашифровані для іншого значення.

Одними із розповсюджених протоколів шифрування є TLS/SSL – широко використовуються у веб технологіях, використовується асиметрична пара ключів на етапі початкового з’єднання «рукостискання».

2.2 Розробка специфікації протоколу

Протокол телеметрії для дистанційно керованої техніки являє собою – прикладний протокол передачі даних для обміну даними між оператором керування та дистанційно керованою технікою. Цей протокол може використовуватися для віддаленого керування, моніторингу чи налаштування різної техніки.

Однією із головних особливостей є легкість його реалізації та використання для розробника. Опишемо основні особливості протоколу телеметрії, що розробляється:

- Протокол може передавати різні типи даних: команди керування, дані телеметрії, аудіо і відео потоки та спеціальні службові дані.
- В залежності від типу даних, що передаються, протокол має різні механізми гарантії їх доставки та перевірки цілісності. Наприклад для команд керування дані повинні надходити гарантовано і швидко, а для відео потоку, гарантія не важлива, важливіше буде затримка передачі.
- Опціонально протокол має можливість шифрувати трафік.
- Протокол забезпечує контейнеризацію та передачу основних типів даних – чисел, строк, двійкових структур.
- Протокол може забезпечувати пріоритетність передавання пакетів. Пакети, вищий пріоритет будуть оброблятися швидше, ніж ті в яких пріоритет менше.

Протокол працює поверх транспортного рівня тому він займає місце 5-го, 6-го та 7-го рівнів у мережевій моделі OSI або 4-й прикладний рівень у моделі TCP/IP. Оскільки протокол базується на транспортному рівні, він може бути переданий по будь-якому середовищу передачі, але при розробці пріоритет надавався використанню протоколу при бездротовому з'єднанні такому як Wi-Fi, оскільки зараз – це є популярним рішенням для підключення дистанційно керованої техніки у сучасному світі.

2.3 Вибір інструментів для розробки

2.3.1 Мережевий аналізатор

Аналізатор мережевого трафіку – це така програма, що записує трафік у мережі для подальшого аналізу. Ця програма зазвичай може фільтрувати трафік за програмою, користувачем або IP адресом [30]. Аналіз мережевого трафіку дозволяє проводити моніторинг мережі, розгортати структури пакетів, це дуже важливо для контролю діяльності різних протоколів та виявлення похибок. Тобто це інструмент для налагоджування протоколу, що спросить розробку власного протоколу телеметрії.

Функції які виконує мережевий аналізатор:

- Під'єднання до інтерфейсу мережевої карти та «прослуховування» трафіку, із можливістю захоплення Ethernet фреймів.
- Можливість підключення аналізатору у розрив каналу зв'язку, що дозволить аналізувати увесь прохідний трафік.
- Перенаправлення трафіку чи створення його копії із подальшим аналізом для різних цілей.
- Фільтрування трафіку за різними критеріями, що дозволяє швидко аналізувати прохідну інформацію людиною.

На ринку представлено багато різних програм аналізаторів трафіку, перелічимо найбільш популярні рішення, що використовуються [30]:

- TCP dump – має консольний інтерфейс, проте незважаючи на це зі своєю роботою він справляється добре, причому використовуючи для цього мінімум системних ресурсів. Але складно розібратися у величезній кількості «сухих» таблиць з даними без графічного інтерфейсу. У деяких середовищах або на ПК, що працюють, мінімалізм може виявитися єдиним прийнятним варіантом. Працює у ОС Windows та Linux.
- Аналізатор трафіку Kismet, ще один приклад консольної програми. Надає розширені функціональні можливості для роботи із бездротовими мережами. Він здатний проводити аналіз трафіку прихованих бездротових мереж, які не

транслюють свій ідентифікатор SSID. Kismet доступний для середовищ Linux, Windows та macOS.

– Wireshark — це мережевий аналізатор, який надає детальну інформацію та може аналізувати сотні сучасних протоколів. Він може захоплювати та розпаковувати файли, забезпечує підтримку дешифрування для різних протоколів, Він може читати та записувати різні формати файлів захоплення. Отримані мережеві дані можна переглядати за допомогою графічного інтерфейсу користувача або консольної утиліти. Він працює у популярних ОС Windows, MacOs та Linux.
– EtherApe виділяється на тлі інших рішень — це орієнтація на графічне відображення. Мережевий трафік EtherApe відображається за допомогою просунутого інтерфейсу, де кожна вершина графа є окремим хостом, розміри вершин і ребер вказують на розмір мережевого трафіку, а кольором різні протоколи. Цей аналізатор має гарне візуальне сприйняття статистичної інформації. Доступний у ОС Linux та MacOS.

Усі представлені аналізатори трафіку мають гарні характеристики, різні слабкі та сильні сторони. У ході аналізу наявних програм було обрано Wireshark у зв'язку із його зручним інтерфейсом користувача який наочно показує структуру пакетів протоколів та їх взаємодію у ієрархії між собою, до того ж він має безкоштовну модель розповсюдження, що добре підходить для освітніх цілей дослідження.

2.3.2 Мова програмування

Для розробки протоколу прикладного рівня потрібна мова програмування, що підтримує роботу із мережею, тобто програмними мережевими інтерфейсами – сокетами. Оскільки більшість сучасних мов програмування відповідають таким вимогам, то немає суттєвої різниці, яку мову програмування обрати. Звідси виходить, що логічніше вибирати мову програмування за критеріями зручності для розробника. Для розробки даного протоколу телеметрії було обрано мову програмування C#, опишемо ряд особливостей, характерні саме для цієї мови та розробки мережевого протоколу [31]:

- Мова C# орієнтується на об'єктно-орієнтовну модель програмування, що дозволяє із легкістю розширювати кодову базу, вносити зміни та оперувати абстрактними сутностями.
- Код на C# транслюється у байт-код, що виконується на віртуальній машині (CLR), звичайно це додає надлишкових витрат, проте має свої переваги у вигляді сумісності із багатьма платформами та автоматичного керування виділенням пам'яті.
- C# має строгу статичну типізацію, що на відміну від динамічної потребує трохи складнішої реалізації від розробника, але натомість надає можливість знайти проблеми ще на етапі компіляції та зменшити можливі надлишкові витрати на перетворення типів.
- Різноманітність підтримуваних технологій для C#, оскільки середовище виконання CLR та базова бібліотека класів – це основа для цілого стеку технологій, які розробники можуть задіяти під час створення різних додатків. Наприклад, для баз даних у цьому стеку є декілька технологій для графічноного інтерфейсу, для розробки для мобільних пристроїв. Окрім цього мова C# також використовується у багатьох сучасних середовищах розробки та версіях бібліотек.
- Мова програмування C# наразі є однією із популярною у галузі для прикладного програмування, тобто цей вибір є актуальним у сучасному світі розробників, що також бачиться і в подальшому майбутньому.

2.3.3 Середовище розробки

Інтегроване середовище розробки (IDE) - це програмне забезпечення для створення програм, що поєднує загальні інструменти розробника в єдиний графічний інтерфейс користувача (GUI). Перечислимо основні функції IDE [32]:

- Редагування коду – середовище повинно вміти створювати нові файли програмного коду, підтримувати їх зручне редагування та збереження.

- Підсвічування синтаксису – IDE розуміє синтаксис мови програмування та надає зручні візуальні підказки. Ключові слова, слова, які мають особливе значення, виділені різними кольорами та можуть мати різне оформлення.
- Підтримка автоматичного заповнення – IDE розуміючи мову програмування, може передбачити, що ви збираєтеся ввести далі розробник, за допомогою алгоритму чи нейронної мережі. Це зберігає час витрачений на зайві натискання клавіш, щоб програміст міг зосередитися на архітектурі свого коду.
- Налагодження – IDE надає інструменти для налагодження, які дозволяють програмістам досліджувати різні змінні та перевіряти свій код виконуючи операції крок за кроком.
- Створення виконуваних файлів – IDE бере на себе процес компіляції, збирання, компонування програмного коду, надаючи на виході виконуваний файл який одразу готовий до запуску.

Виходячи із обраної мови програмування (у підрозділі 2.3.2) необхідно обрати підходяшу для неї середу розробки. Оскільки їх існує велика кількість, обиратимемо серед найбільш типових та популярних у даний час:

- Project Rider – ця IDE створення як продовження ідей популярного розширення Visual Studio для розробників .NET. Це потужна та легка IDE, яка може похвалитися багатьма інструментами для рефакторингу коду та покращеннями продуктивності.
- MS Visual Studio – це IDE та інструмент розробки програмного забезпечення для створення програм .NET за допомогою C#.
- Eclipse – IDE, що легко може розширяти свою функціональність за допомогою плагінів, доступних на ринку Eclipse. Також поставляється з інструментами аналізу коду та його налагодження.
- MonoDevelop — це редактор коду, розроблений Xamarin. IDE може похвалитися широким набором функцій, які спрощують розробникам створення складних проектів або рішень.

Беручи до уваги вищезгадані особливості IDE, було обрано MS Visual Studio, оскільки: вона має безкоштовну ліцензію, має підтримку сучасних стандартів мови програмування C# оскільки, що IDE, що C# розробляє одна й та сама компанія. Окрім всього MS Visual Studio одні зі найкращих інструментів налагоджування програм, що позитивно відзначиться при розробці прикладного протоколу телеметрії.

2.4 Програмна реалізація протоколу телеметрії

2.4.1 Вибір транспортного протоколу

Будь-який протокол прикладного рівня базується на протоколі транспортного рівня, цей протокол бере на себе роль передачі повідомлень від одного приладу до іншого, не залежно від типу пристрою. На даний час є два типи транспортних протоколів, що підтримуються на рівні ОС та не мають проблем сумісності із обладнанням – TCP та UDP [4]. Є також похідні протоколи від них, але вони на тому для спрощення аналізу вони розглядатися не будуть. Ці два протоколи мають фундаментальні відмінності у підході до передачі даних, приведемо їх нижче:

- TCP створює з'єднання для кожної передачі та має додаткові перевірки для гарантії доставки повідомлень, такі як: підтвердження, нумерація сегментів, контроль похибок. Також присутній механізм керування потоком передачі. Увесь цей додатковий функціонал корисний, проте має свою ціну – надлишкові витрати продуктивності обладнання та мережі, у тому разі якщо додаткові перевірки не потрібні.
- UDP має спрощений механізм передачі – не потрібно створювати з'єднання для передачі даних, немає зайвих полів у пакетах (дейтаграмах). За рахунок цього досягається висока швидкість відклику, але усі додаткові механізми, якщо необхідно потрібно реалізовувати на вищих рівнях.

Виходячи із вимог (підрозділ 2.2) які були поставлені перед створенням протоколу телеметрії було обрано транспортний протокол UDP, оскільки на відміну від TCP немає зайвих перевірок, менша необхідна кількість пакетів без організації з'єднання, менший розмір пакетів оскільки дейтаграми UDP мають

мінімальну кількість полів. За необхідності усі ці механізми можна реалізувати на прикладному рівні саме там де вони потрібні, тобто маючи вибір в залежності від типу даних, що передаються.

2.4.2 Сеанс зв'язку. Ініціалізація та завершення з'єднання

Сеанс зв'язку – це такий процес який включає в себе створення з'єднання із віддаленим мережевим хостом, передачу корисної інформації та завершення з'єднання. Оскільки раніше обраний протокол транспортного рівня UDP (підрозділ 2.4.1) не забезпечує організацію сеансу зв'язку, його потрібно реалізувати у протоколі телеметрії.

Хоча в сесії протоколу телеметрії можна виділити клієнтську та серверну сторони, протокол насправді повністю симетричний. Після встановлення з'єднання обидва хости грають роль рівноцінних кінцевих точок, що обмінюються різними типами даних. Серверна частина відповідає за прослуховування каналу та службові повідомлення. Клієнтська частина відповідає за пересилання безпосередньо корисної інформації.

Опишемо процес встановлення зв'язку:

- Перший хост спробує відправити пакету віддаленому хосту.
- Спроб встановлення з'єднання може бути декілька, щоб компенсувати можливі втрати пакетів.
- Якщо віддалений хост існує він отримає повідомлення, якщо він підтримує протокол телеметрії він відповідає відповідним повідомленням та додає у спеціальне поле номер версії протоколу, що він підтримує. Таке повідомлення відправляється у відповідь першому хосту.
- Якщо перший хост отримав повідомлення, він звіряє версію протоколу, якщо все гаразд то відповідає повідомленням на згоду, якщо ні, то відповідає повідомленням про завершення з'єднання.

Встановлене з'єднання може час від часу перевірятися службовими повідомленнями на те що воно досі існує чи перевіряти стан каналу зв'язку. Під час встановленого з'єднання можуть передаватися різні типи пакетів протоколу.

Опишемо процес завершення зв'язку:

- Перший вузол, що хоче розірвати з'єднання відправляє другому вузлу повідомлення про завершення з'єднання та завершує сеанс зв'язку.
- Другий вузол отримує повідомлення та завершує сеанс зв'язку.
- Якщо другий вузол не отримав повідомлення про завершення з'єднання, через деякий час він зрозуміє за допомогою службових повідомлень, що іншого хосту немає та завершить з'єднання.

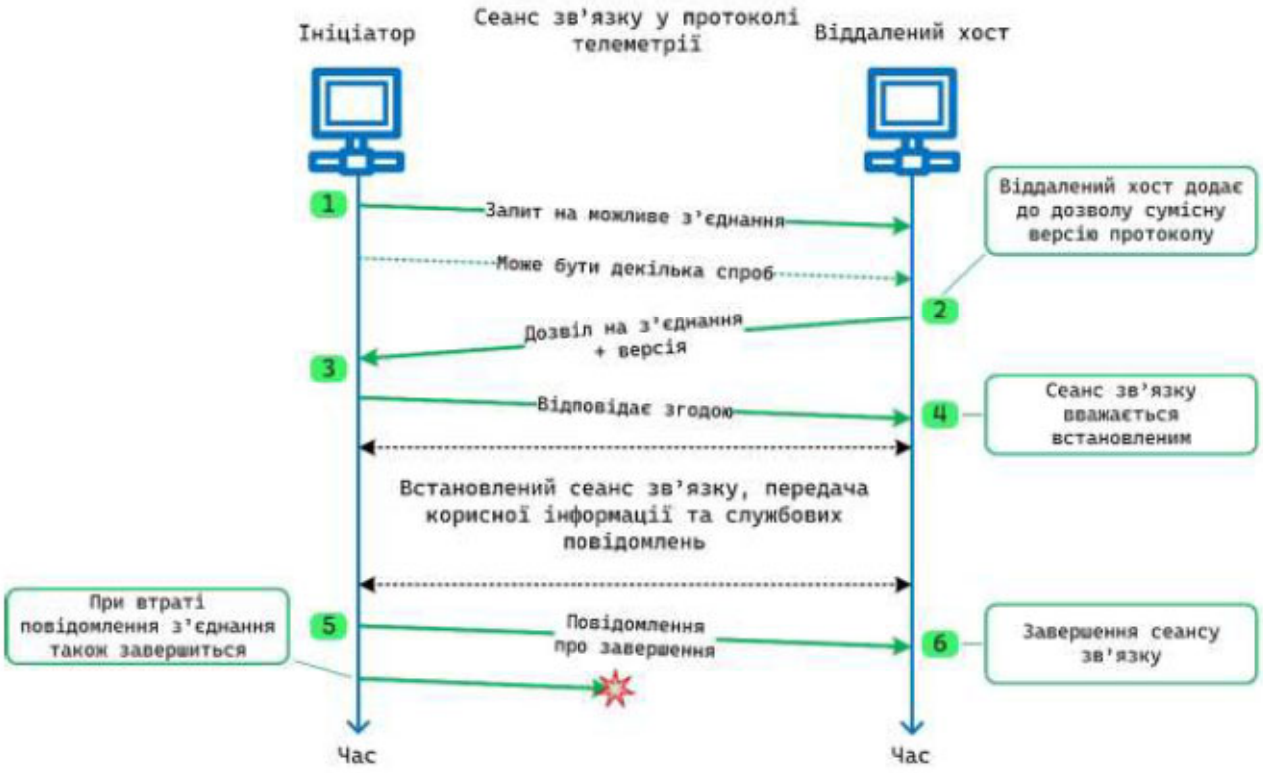


Рисунок 2.3 – Діаграма послідовності сеансу зв'язку

2.4.3 Вибір формату даних

На перший погляд може здатися очевидний вибір на користь бінарного проколу оскільки він більш швидкий, потребує менших накладних витрат та не потрібен якийсь текстовий формат для строкових даних. Але існує багато проблем, які виникають у мережі, коли обладнання та програмне забезпечення різних

виробників повинні існувати та працювати разом. Однією із таких проблем є те як операційні системи та апаратні архітектури представляють числові та інші типи даних у своїй пам'яті. В такому випадку якщо використовувати двійкове представлення даних, протокол потребуватиме перевірки сумісності типів даних на кожному обладнанні. Якщо використовувати текстовий формат даних, він буде інкапсулювати у собі представлення даних та відтворюватися на кожному обладнанні правильно, але він має великі надлишкові витрати.

Тому протокол телеметрії має гібридний формат, поєднуючи гарні сторони обох підходів. Для тих даних де типів не багато і є потреба у їх продуктивній передачі буде використовуватися двійкове представлення, а для інших даних використовується представлення у текстовому форматі.

Можна класифікувати повідомлення у протоколі за типом представлення даних:

- Двійковий формат – дані керування, команди налаштувань, запити, потокові дані аудіо/відео форматів.
- Текстовий формат – довільні дані телеметричних сенсорів або комплексні команди налаштувань.

Для текстового представлення даних доцільно використовувати формат JSON [33], оскільки на відміну від інших (наприклад XML) він має зручний вигляд, легко читати людиною та мінімальний опис виду «властивість – значення. Такий мінімальний формат гарно підходить для протоколу телеметрії, що б бути сумісним із обладнанням, що має невеликі апаратні можливості.

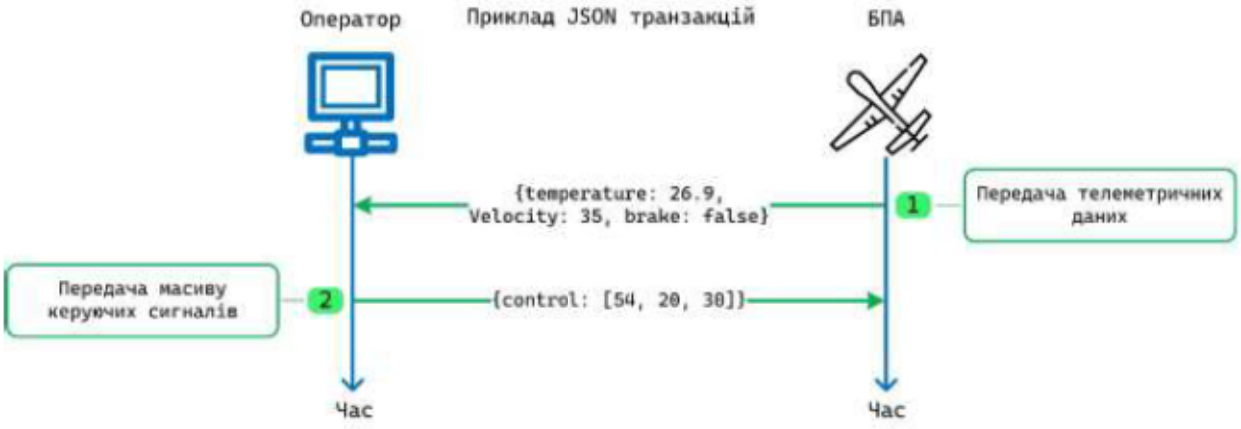


Рисунок 2.4 – Структура корисного навантаження у форматі JSON

2.4.4 Структура пакету

Структуру пакету протоколу телеметрії можна поділити на три частини:

1. Заголовок – йде перед будь яким пакетом.
2. Тіло пакету – має корисні дані та додаткову інформацію для їх обробки.
3. Контрольна сума – вона вставляється у кінці кожного повідомлення і виявляє похибки у тілі пакету при передачі.

У протоколі телеметрії є декілька видів пакетів:

– Тип «Керування» – пакети команд та значень для керування.
– Тип «Запит-відповідь» – пакети запитів та відповідей.
– Тип «Потокові дані» – пакети потокових даних аудіо та відео.
– Тип «Дані» – пакети телеметричних, конфігурації та довільних даних.

Опишемо структуру заголовку, що слідує перед кожним пакетом:

1. Маркер, що позначає, що це протокол телеметрії. Цей маркер має довжину три байти та такі шістнадцятирічні значення – 2D, FF, 2D.
2. Байт, що вказує на наявність шифрування, якщо він відмінний від нуля, то пакет зашифрований.
3. Байт, що вказує на тип повідомлення яке має у собі пакет. Даний тип даних може вказувати до 255 типів повідомлень.
4. Байт контрольної суми – дозволяє виявити похибки у заголовку, що виникли під час передачі.

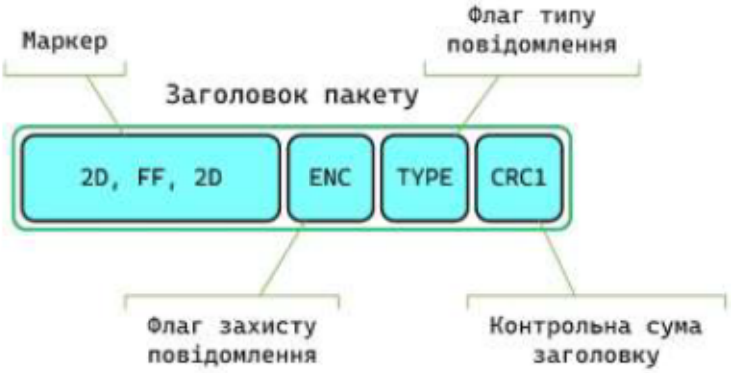


Рисунок 2.5 – Структура заголовку пакету протоколу телеметрії

Усі подальші описи структур пакетів, будуть без опису заголовку, що б уникнути дублювання. Опишемо структуру пакету команд керування:

1. 12 каналів ШІМ 16 бітної розрядності. Такої кількості каналів вистачить для більшості моторів чи сервоприводів, що встановлені на БПА.
2. Контрольна сума.

Опишемо структуру пакету запитів-відповідей:

1. Байт, що відповідає коду відповіді чи запиту.
2. Контрольна сума.

Опишемо структуру пакету даних:

1. Номер послідовності – дозволяє визначити сегмент при передачі даних, що є більшими максимальний розмір пакету.
2. Корисне навантаження у вигляді текстового формату JSON.
3. Контрольна сума.

Опишемо структуру пакету потокових даних:

1. Номер послідовності – дозволяє визначити сегмент при передачі даних, що є більшими максимальний розмір пакету.
2. Корисне навантаження у вигляді бінарному вигляді.
3. Контрольна сума.

Хоча розмір пакету протоколу обмежується розміром максимальної дейтаграми, що здатен перенести протокол UDP, але для практичного використання максимальний розмір усього пакету встановлений на 508 байтів, це обумовлено максимальним розміром MTU із вирахуванням максимально можливих розмірів заголовку IP. Даний максимальний розмір пакету гарантує проходження пакету через майже будь-яке мережеве обладнання.

Для передачі ШІМ сигналів використовуються числа 16 бітної розрядності (2 байти), цього цілком досить для передачі команди керування адже зазвичай контролери мають керуючий двигунами ШІМ 10/12/16 бітної роздільної здатності. Використання такого типу даних дозволяє встановити 65 535 рівнів регулювання двигунів. дозволяє зменшити надлишковість об’єму інформації, що в свою чергу зменшує також відклик системи керування і все це без втрати якості керування.

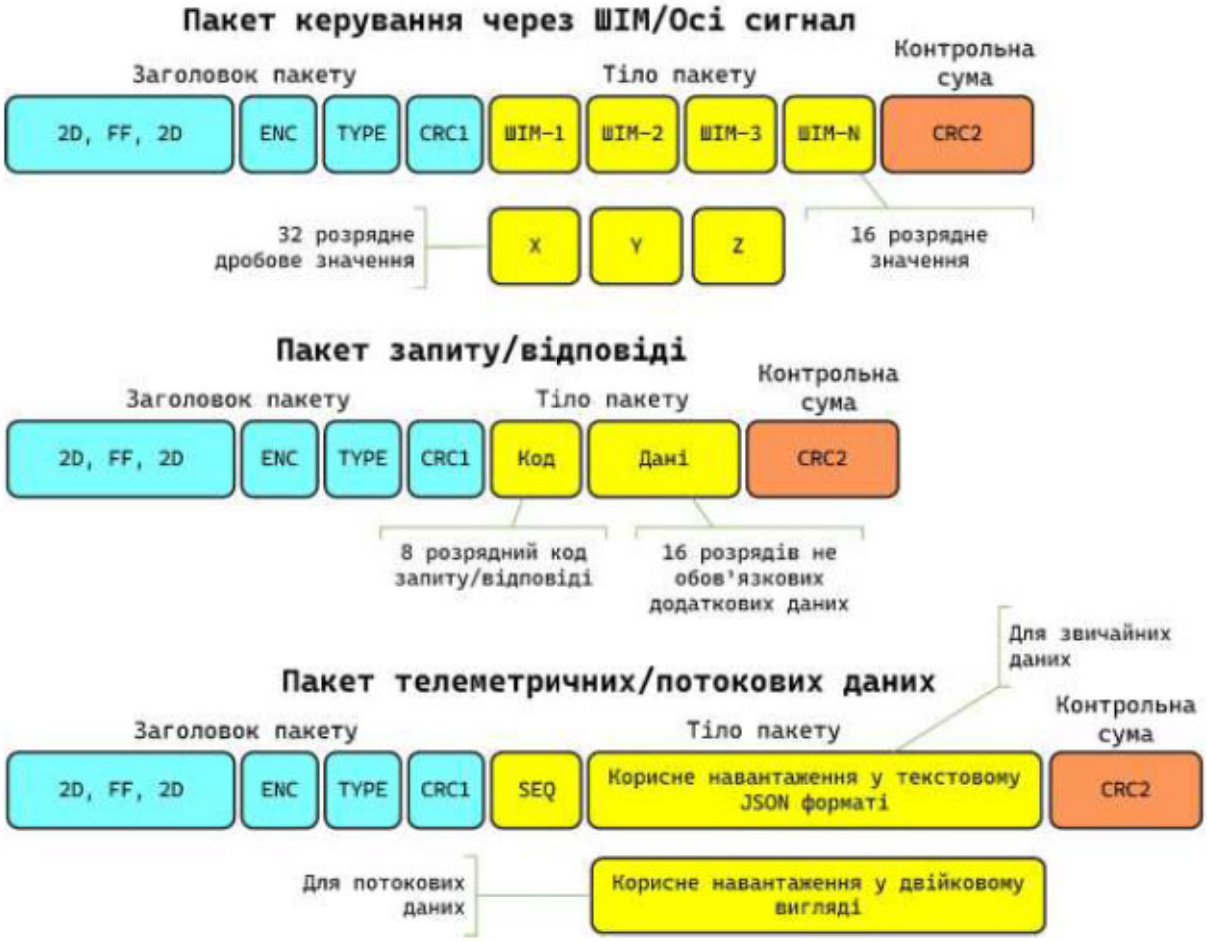


Рисунок 2.6 –Структури типів повідомлень протоколу телеметрії

2.4.5 Семантика повідомлень, команд та запитів

Семантика повідомлень описує смислове значення кожного повідомлення, що використовується у протоколі телеметрії для дистанційно керованої техніки. Розглянемо наявні типи повідомлень із огляду на їх смислове значення:

– Тип «Дані» – дозволяє передавати довільні дані у текстовому форматі JSON. Призначення цього типу передавати різноманітні телеметричні дані із датчиків та іншого обладнання, що встановлене на БПА, оскільки JSON може переносити майже будь-які дані, які можливо конвертувати у строкове представлення. Також цей тип використовується для комплексного налаштування БПА, оскільки дозволяє передати однією транзакцією відразу всі налаштування будь-якого роду.

- Тип «Керування» – дозволяють швидко і з гарантованою доставкою передати значення керування рухом від оператора до дистанційної техніки. Наявно два типи: ШІМ, що дозволяє передавати до 12 каналів керування по 16 біт розрядів та значення осей керування, це надає змогу передати дробові значення трьох осей що мають 32 бітну розрядність. За необхідності цей тип можливо розширити за рахунок нових підтипів, що мають інше представлення даних.
- Тип «Запити-відповіді» – надають можливість швидко передати коротке повідомлення. Отримавши це повідомлення в залежності від його змісту, отримувач також може відправити відповідь. Цей тип використовується для службового трафіку типу: перевірка стану зв'язку, доступність хостів, встановлення та роз'єднання сеансу зв'язку. Окрім цього запити можуть використовуватися для передачі команд управління різним обладнанням на БПА. Для цього потрібно відправити запит із ідентифікатором обладнання та його новим значенням.
- Тип «Потокові дані» – такий тип забезпечує передавання потоків аудіо та відео. Дані передаються у двійковому вигляді прямо із обладнання, що генерує потік даних без додаткових перетворень окрім фрагментації якщо вона потрібна. Гарантія доставки відсутня, є лише контроль послідовності, якщо використовується фрагментація для розбиття пакетів на менші ніж максимальний розмір пакету сегменти. Такий підхід дозволяє швидко передавати потокові дані з камери чи мікрофону встановленому на БПА.

Опишемо вид представлення даних, їх корисне навантаження та приклади практичного використання повідомлень, команд та запитів у протоколі телеметрії (табл. 2.1). Приклади виду корисного навантаження не є точним відображенням реалізації у програмній реалізації та є спрощеними для зручного сприйняття.

Таблиця 2.1 – Приклади корисного навантаження різних типів повідомлень

Тип повідомлення	Макс. значення	Представлення даних	Приклад корисного навантаження
«Керування» ШІМ	12 ШІМ каналів по 65 535 рівні	Двійкове, 16 розрядів * 12	ШІМ-1 = 30; ШІМ-2 = 500; ШІМ-3 = 780; ШІМ-4 = 945; …
«Керування» Осі	3 осі X,Y,Z. Дробове значення від 0 до 1	Двійкове, 32 розряди * 3	X = 0.35, Y = 0, Z = 0.27
«Запит» Команда управління	Ідентифікатор обладнання та його значення до 65 535 рівнів	Двійкове, 8 + 16 розрядів	ID 14 = 450 (Обладнання №14 встановити своє значення на 450)
«Запит» Доступність хосту	Код запиту. Значення до 255	Двійкове, 8 розрядів	Код = 20 (Отримувач повинен відповісти кодом 60)
«Відповідь» Добре	Код відповіді Значення до 255	Двійкове, 8 розрядів	Код = 61 (Це може бути підтвердження повідомлення)
«Дані» Телеметрія	Рядок символів до 31 МБ	Текстове JSON, частинами по 498 байт	Датчик 1=2.37, Датчик 2 = Холодно, Датчик 3 = 980.3 … (Цей тип може містити різноманітні дані)
«Потокові дані» Відео	Масив байт до 31 МБ	Двійкове, частинами по 498 байт	Кадри із потоку зображень відео-формату

2.4.6 Забезпечення гарантії доставки повідомлень

Протокол телеметрії розробляється із забезпеченням мінімальної кількості надлишкових даних та зайвих повідомлень, проте для деяких даних критично мати забезпечення гарантії доставки повідомлень. Таким типом повідомлень є тип «Дані» що передають інформацію у текстовому форматі JSON.

Функція гарантії доставки є необов'язковою оскільки для деяких телеметричних даних, краще передати новий пакет із оновленими показниками, ніж в разі помилки передавати старий. Для типу повідомлень «Дані» та «Потокові дані» обов'язковим є контроль послідовності сегментів, що б мати можливість передавати дані, що більші ніж максимальний розмір пакету даних у протоколі телеметрії.

Тобто у протоколі телеметрії є два підходи до гарантії доставки повідомлень:

1. Гарантії доставки немає взагалі, відправник не очікує ніякого підтвердження про отримання повідомлення, якщо виникне помилка при передачі, то повторного відправлення не буде. Такий підхід використовується для більшості типу трафіку у протоколі.
2. Наявна гарантія доставки повідомлень, відправник повідомлення очкує підтвердження повідомлення, якщо його немає то відправляє повторно повідомлення. Лише у разі якщо було отримано підтвердження відправник відправляє наступний пакет даних. Це дає надійний захист від втрати повідомлення у каналі зв'язку, але додає затримки при передачі, оскільки потребує додаткових повідомлень-підтверджень. Такий підхід використовується обов'язково для типу «Керування» та може бути ввімкнений для типів «Запит» і «Дані».

Окремо слід зазначити контроль послідовності сегментів при передачі. Якщо увесь об'єм даних, що передаються перевищує максимальний розмір пакету, то вони фрагментуються на дрібніші пакети – сегменти, що б не перевищувати вищезгаданий ліміт. У такому вигляді пакети-сегменти передаються по мережі, але може виникнути ситуація коли один пакет-сегмент прийшов раніше ніж інші, або взагалі не прийшов. При об'єднанні таких сегментів у єдине повідомлення буде отримано дані із похибкою. Тому усі пакети-сегменти нумеруються послідовним номером. Сторона, що приймає повідомлення може за допомогою нумерації зібрати пакети-сегменти у правильному порядку чи зрозуміти що їх не вистачає і відкинути дані чи запросити новий пакет. Таким чином забезпечується цілістність даних. Цей підхід використовується у типах повідомлень «Дані» та «Потокові

дані», адже лише вони можуть мати розмір більший за максимальний розмір пакету.

Для більш наочного сприйняття приведемо наявність гарантій доставки та контролю послідовності для різних типів даних у таблиці 2.2 нижче.

Таблиця 2.2 – Можливості забезпечення гарантії доставки у повідомленнях

Тип даних, що передаються	Гарантія доставки	Контроль послідовності
Телеметричні дані	Немає	Є
Довільні дані	Опціонально	Є
Потік аудіо даних	Немає	Немає
Потік відео даних	Немає	Немає
Дані конфігурації	Є	Є
Запити	Опціонально	Немає
Відповіді	Немає	Немає
Дані керування	Є	Немає

2.4.7 Виявлення помилок при передачі

Пакетні помилки є поширеними помилками при передачі в багатьох каналах зв'язку, включаючи кабельне з'єднання, бездротовий радіозв'язок та оптичні лінії. Для успішної передачі по протоколу телеметрії необхідно, щоб цільова система отримувала ті самі дані, що їх надсилає вихідна система. Хоча протокол UDP може мати перевірку помилок за допомогою контрольної суми, за замовчуванням вона виключена. До того ж використання власної системи дозволить забезпечити перевірку помилок незалежно від протоколів нижчих рівнів.

Є декілька розповсюджених способів виявлення помилок при передачі:

– Біт парності – це контрольний біт, який додається до блоку даних з метою виявлення помилок. Він використовується для перевірки цілісності даних. Значення біту парності встановлюється або 0, або 1, в залежності від парності суми всіх одиниць у повідомленні. Проте перевірка на парність підходить лише для виявлення однобітової помилки.

– Контрольна сума — метод виявлення помилок, який перевіряє цілісність переданих даних. Значення контрольної суми складається з послідовності цифр і літер. Щоб обчислити контрольну суму, потрібно пропустити дані через хеш-функцію. Часто використовувані алгоритми: MD5, SHA-1, SHA-256 і SHA-512.
– Циклічна перевірка надлишковості (CRC) — це код виявлення помилок, який дозволяє виявити випадкові зміни переданих даних. У місці призначення вхідні дані діляться на контрольне значення (дільник). Якщо залишок дорівнює нулю, то блок даних вважається дійсним і прийнятим . Залишок, відмінний від нуля, вказує на те, що дані пошкоджені та повинні бути відхилені. Перевірка CRC у передавача та генератор CRC у приймача повинні працювати однаково.

Вибираючи між вищезгаданими методами, відразу відкидаєтеся біт парності, оскільки ця перевірка дозволяє лише визначити помилку у одному байті, тоді коли у розповсюдженому типі середовища для керування БПА – радіоканалу, виникають помилки, що впливають відразу на декілька біт.

Для вибору між контрольною сумою і циклічним надлишковим кодом потрібно привести їх різницю між собою:

– CRC може виявляти двозначні помилки, контрольна сума лише однобітові.
– Контрольна сума потребує менше ресурсів для розрахунку.
– Циклічний код надлишковості є більш надійним видом виявлення помилок, завдяки математиці, що покладена в його основу.
– CRC є більш новою концепцією виявлення помилок, ніж контрольна сума.

З огляду на все вище сказане та беручи приклад із інших прикладних протоколів доцільно обрати циклічний код надлишковості для виявлення помилок у протоколі телеметрії.

Розглянемо процес контролю помилок за допомогою CRC:

– На стороні, що передає потрібно додавати бітовий (в залежності від розрядності контрольної суми) контрольний код до корисних даних, та надіслати пакет на приймальну сторону.
– Під час отримання пакету приймальна сторона перевіряє правильність отриманих даних на основі отриманих даних і контрольного коду.
– Контрольний код повинен гарантувати, що дані можуть ділиться на певне число, яке вибирається як передавальною, так і приймальною стороною. Приймальна сторона ділить отриманий новий пакет на вибраний дільник. Залишок від ділення повинен дорівнювати нулю. Якщо є залишок, то це значить що є помилка під час передачі пакету.

Для виявлення помилок у протоколі телеметрії використовується дві контрольні суми одна CRC-8 для захисту заголовку пакету та друга CRC-16 для тіла пакету. Використання двох контрольних сум дозволяє збільшити шанс виявлення помилки при передачі. Розглянемо їх розміщення у пакеті:

– CRC-8 розміщується в кінці заголовку та дозволяє виявити в ньому помилки. В разі помилки скидається відразу усе повідомлення, не витрачаючи час на обробку іншої частини пакету.
– CRC-16 розміщується в кінці пакету, відразу за тілом повідомлення. Дозволяє виявити помилки у корисному навантаженні. Розміщення CRC у кінці пакету дозволяє по мірі оброблення даних обчислювати контрольну суму та не потребуючи додаткової буферизації порівняти отримане значення із CRC в кінці.

2.4.8 Криптографічний захист даних

Якщо у прикладному протоколі не передбачено використання шифрування він уразливий для будь-якого виду атак, до яких уразливий його транспортний рівень, тобто протокол UDP. Криптографія може бути різною, але для гарантованого результату, кращим варіантом буде вибрати готове рішення. Таким рішенням виступає алгоритм Rijndae [34]. Який вже має реалізацію у стандартній бібліотеці C#. Потрібно лише додати його до протоколу телеметрії.

Опишемо його головні особливості алгоритму шифрування Rijndael:

- Rijndael являє собою алгоритмічну назву розширеного стандарту шифрування (AES). У сучасному світі це є один із самих розповсюджених варіантів шифрування, що використовується у сфері телекомунікацій.
- У платформі .NET цей алгоритм має довжину ключів 128, 192 чи 256 біт; значення за замовчуванням – 128 біт (AES сумісний).
- Алгоритм Rijndael представляє собою симетричне шифрування, що використовує єдиний секретний ключ. Дані, зашифровані за допомогою цього ключа, можуть бути розшифровані тільки за допомогою того самого ключа.
- Симетричні алгоритми завжди використовують один і той же ключ для шифрування та розшифрування. Симетричні алгоритми працюють швидко в обох напрямках.
- Надійність шифрування пропорційна довжині ключа. Чим більша довжина ключа, тим менша ймовірність успішного злому методом перебору, оскільки потрібно буде перевіряти набагато більше можливих значень ключа.

Шифрування у протоколі телеметрії є додатковою можливістю, для цього потрібно виставити у заголовку поле «Encode» значенням true. Опишемо процес створення зашифрованого каналу між БПА та оператором, використовуючи протокол телеметрії:

- Секретний ключ повинен бути попередньо згенерований на обох сторонах. Довжина ключу 256 біт.
- Сторони шифрують свої повідомлення відомим їм ключем та обмінюються зашифрованим трафіком.
- Задля безпеки не повинно бути доступу до програмного забезпечення де використовується протокол, оскільки тоді є можливість розбору виконуючих файлів та компрометація секретного ключа.

ВИСНОВОК

Даний розділ описував створення протоколу телеметрії для дистанційно керованої техніки. Для чого було проаналізовано та виведено методи і принципи розробки прикладних протоколів: сеанс-зв'язку, детально розібрано структура пакетів у байтах та їх призначення, представлення даних у протоколах, фрагментація пакетів від максимального розміру, що може бути переданий по мережі.

Перед початком програмної реалізації протоколу телеметрії, була описана його специфікація – призначення протоколу, вимоги та основні задачі які повинен вирішувати. Після поставленого завдання було обрано інструменти для програмної реалізації – середовище розробника, мережевий аналізатор трафіку та мова програмування.

Описавши необхідні умови та вимоги із вибраними інструментами для створення протоколу телеметрії, була розпочата розробка протоколу: вибір транспортного протоколу, організація сеансу зв'язку, його встановлення та роз'єднання. Враховуючи вимоги до протоколу було обрано гібридний формат даних: для користувацьких даних та телеметрії текстовий JSON, а для всього іншого двійковий вигляд. Також були описані типи пакетів, їх структурні особливості, шифрування даних та спосіб виявлення помилок при передачі.

Отже була розроблена та створена програмна реалізація із детальним описом використовуваних методів, це дозволяє перейти до створення моделі тестування та провести дослідження роботи протоколу телеметрії у наступному розділі.

3 РОЗРОБКА ПРОГРАМНОЇ МОДЕЛІ НАЛАГОДЖУВАННЯ ПРОТОКОЛУ ТЕЛЕМЕТРІЇ

3.1 Опис моделі налагоджування

У зв'язку із потребою тестування роботи протоколу виникла необхідність розробки та створення моделі тестування. Така програмна модель дозволяє провести тестування без потенційних ризиків та витрат, які можуть виникнути реальному житті.

Оскільки був створений протокол передачі даних на прикладному рівні необхідно протестувати усі вищі над прикладним рівні: сеансовий рівень – координація зв'язку між двома хостами у мережі, транспортний рівень – контроль за передачею, підрахунок черговості отримання повідомлень та рівня представлення, де виконуються внутрішні перетворення із форматів даних протоколу передачі у формат даних, що використовується безпосередньо прикладним забезпеченням і виконує корисну роботу.

Рисунок 3.1 – Необхідні рівні тестування протоколу телеметрії

Програмна модель для практичного налагоджування протоколу повинна відповідати таким умовам:

- Виводити в зручному для сприйняття вигляді розгорнуту інформацію про внутрішню будову пакетів, що передаються у протоколі телеметрії.
- Тестування створення сеансу зв'язку: встановлення з'єднання, перевірка стану з'єднання, завершення з'єднання.

- Тестування передачі різних типів даних у протоколі – потокових, команд керування та службового трафіку у вигляді запитів відповідей.
- Перевірка криптографічного захисту при передачі інформації, контроль шифрування та дешифрування пакетів.
- Тестування відповідності даних, що були на виході передавача таким даним, що є на вході отримувача. Тобто контроль перетворення внутрішніх форматів даних у формат прикладного забезпечення.
- Перевірка гарантії доставки повідомлень та забезпечення їх цілісності при передачі через мережу.

Програма для налагоджування протоколу дозволить провести тестування мережевої роботи механізм протоколу, відпрацювати його базовий функціонал у умовах, що імітують реальні – генерацію змінних, їх псевдо випадковість часу передачі, передачу характерного для БПА трафіку. Крім того можна використовувати віртуальні аналоги обладнання наприклад, замість відеокамери пересилання серії зображень, сама модель тестування за рахунок використання платформи розробки Unity може працювати на широкому асортименті обладнання, що дозволить легко переробити модель тестування під мінливість ситуацій, тобто полегшуйтеся усе те, що зможе зменшити часові та коштовні витрати на налагоджування протоколу.

3.2 Програмна реалізація моделі налагоджування

3.2.1 Створення графічного інтерфейсу користувача

У програмі налагоджування та тестування протоколу телеметрії користувацький інтерфейс (UI) необхідний для зручного сприйняття людиною, адже сам по собі протокол оперує структурами даних у вигляді текстових форматів що мають своє форматування або ж взагалі дані у вигляді байтів. До того часові затримки між відправкою та прийняттям пакетів даних можуть складати одиниці мс, що є очевидно дуже малим часом для усвідомлення розробником. Враховуючі ці вищезгадані основні проблеми є доцільним для налагоджування протоколу телеметрії створити зручний інтерфейс.

Отже у ході розробки був створений інтерфейс для налагоджування протоколу у якому відображаються такі характеристики та типи даних:

- Список усіх відправлених та прийнятих пакетів даних.
- Розгляд окремого пакету даних із розгортанням його внутрішньої будови та представленням типів даних у зручній формі.
- На панелі детального розгляду (рис. 3.3) можна побачити час прийняття чи відправки пакету, ідентифікатор протоколу, тип пакету даних та його корисне навантаження.

Для створення інтерфейсу було обрано широкоформатну так звану альбомну орієнтацію коли ширина екрану більша за висоту, це дозволить зручно сприймати інформацію та натискати на кнопки керування.

Одним із способів зберегти розташування об’єктів UI в області є зміна компонування таким чином, щоб місця їх розташування були прив’язані до відповідних кутів на екрані. Як тільки об’єкти будуть прив'язані до своїх кутів, то при подальших змінах розподільчої здатності екрану та співвідношень сторін вони зберігатимуть свою позицію відносно цих кутів. Таке рішення дозволяє досягти мультиплатформенності при розробці застосунків.

Побудова користувацького інтерфейсу відбувається за допомогою вбудованого інструменту UI в Unity [35]. Вона має ієрархічну будову коли один компонент вкладається у інший (рис. 3.2), наприклад список прийнятих пакетів включає у себе динамічне тіло відображення, повзунок для перетягування списку та безпосередньо самі об’єкти пакетів даних.

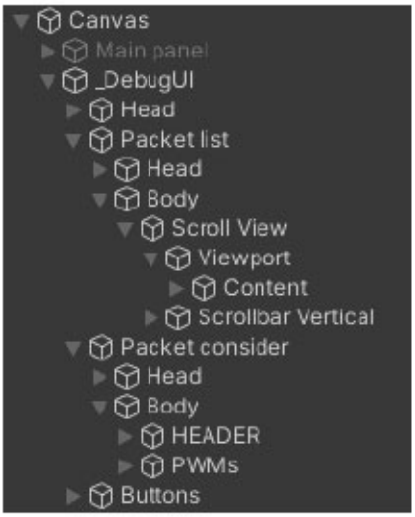


Рисунок приклад 3.2 – Ієрархія UI компонентів панелі налагоджування

Компонент Canvas повинен бути батьківським для усіх подальших графічних об’єктів. Canvas – об'єкт із вбудованим у нього компонентом Canvas, всі елементи інтерфейсу якого є його нащадками. Canvas є полотном для малювання графічного інтерфейсу. Далі було створено графічні об’єкти типу Panel, він містить компонент Image який дозволяє відображати текстури тобто двох вимірні зображення. Крім цього було також використано компонент TextMeshPro, його призначення робота із текстом. Він підтримує усі основні види форматування та кодування текстових символів окрім, цього має підтримку графічних символів типу емоджі. Використовуючи ці три основних компоненти було побудовано графічний інтерфейс для налагоджування протоколу телеметрії, отримана графічна система може працювати на багатьох сучасних платформах та підтримує велику кількість розподільчих здатностей дисплеїв, що позитивно відображається на процесі розробки усього проєкту.

Рисунок 3.2 – Меню GUI для налагоджування протоколу телеметрії

3.2.2 Реалізація віртуального БПА

Концепція тестування протоколу телеметрії у моделі налагоджування полягає у віддаленому керуванні віртуальною моделлю БПА обмін даних з яким буде відбуватися за допомогою попередньо розробленого протоколу передачі даних.

Опишемо віртуальну модель БПА та її основні характеристики за рахунок яких вона може бути використана для якісного тестування протоколу телеметрії:

– Підтримує можливість рухатися по віртуальному середовищу.
– Має віртуальні датчики які можуть генерувати показники.
– Додаткове обладнання яким можна керувати.

Для пересування моделі БПА було створено двох вимірну віртуальну мапу, що являє собою симуляцію навколишнього середовища. Мапа має випадковим чином згенеровані зони із показниками різних величин: вологості, температури, швидкості вітру, атмосферного тиску і т.п. для подальшої можливості їх виміру.

Мапа для тестування генерується випадковим чином за допомогою додаткового набору компонентів «2D Map Generator Tool». Це дозволяє швидко створювати віртуальне середовище із необхідними параметрами. Далі для кожної клітинки задаються параметри для зчитування віртуальними датчиками БПА такі як температура, вологість та інші дані для телеметрії. Це дозволяє провести тестування передачі телеметричних даних у протоколі телеметрії одразу після внесення корегувань розробником без необхідності реальних тестувань, економлячи час та зусилля для налагоджування.

На рисунку нижче (рис. 3.4) показана створена мапа для тестувань, у якій видно машинку для віддаленого керування (імітація БПА). Клітинки на мапі мають різний колір: синій це вода, що являє собою непрохідну зону, різні відтінки зеленого це відображення рівню вологості на кожній клітинці, що дозволяє відразу бачити кореляцію із власне візуальним положенням БПА на мапі та показниками, що передаються без необхідності ручного звірення величин.

Рисунок 3.4 – Віртуальна мапа із зонами показників вологості

3.2.3 Інтеграція протоколу телеметрії у модель тестування

Зазвичай для керування БПА використовуються спеціальні пульти чи джойстики, проте оскільки вид терміналу не впливає на принцип роботи протоколу передачі даних то для спрощення розробки та робіт із налагодження було обрано програмний вид терміналу.

Опишемо основні переваги які надає використання програмного терміналу та моделі:

- Можливість запускати тестування одразу після додавання нового функціоналу у середовищі розробника, що у рази пришвидшує розробку.
- Не потрібно брати із собою обладнання, ризикувати їм та шукати місце для його запуску.
- Немає обмежень що до часу роботи дистанційно керованої техніки, у реальних умовах це обмежується ємністю акумулятора.
- Можливість швидко добавляти різні сценарії у модель тестування.

Для налагоджування протоколу потрібен термінал керування який являє собою програмне забезпечення та віртуальну модель тестування, вони обидва можуть бути з'єднанні через протокол телеметрії.

Опишемо типовий процес роботи із терміналом для налагодження:

- Уведення в терміналі керування мережевої IP адреси та порту призначення, що відповідає хосту до якого потрібно підключитися, тобто моделі налагоджування.
- Спроба з'єднання, на даному етапі перевіряється доступність адреси та відповідність протоколів передачі даних. Якщо все добре то з'єднання буде встановлене і на екрані з'явиться характерне повідомлення.
- В подальшому можна керувати через віртуальний джойстик дистанційною технікою чи подавати різні сигнали для виконавчих механізмів. Усі ці команди подаються через мережу протоколом телеметрії та приймаються стороною на якій встановлена віртуальна модель де одразу будуть видні зміни в поведінці техніки в залежності від надісланої команди керування.

– Віртуальна дистанційна техніка у моделі тестування постійно передає певний потік даних, в залежності від налаштувань це можуть бути як показники датчиків так і аудіо чи відео потоки або усе разом. Ці дані також передаються через мережу за допомогою протоколу телеметрії.
– Для завершення тестування достатньо розірвати з'єднання відповідною кнопкою. Якщо під 'єднатися знов техніка буде у тому ж стані як на момент роз'єднання зв'язку, але є можливість примусово привести її до початкового стану.

Робота камери являє собою зняття знімків мапи із положенням БПА та відправку цих кадрів на термінал. За рахунок цього можна протестувати передачу безперервних потокових типів даних таких як відео чи аудіо сигнал.

Віртуальний термінал керування (рис. 3.5) являє собою застосунок у якому є графічний інтерфейс на якому зверху зображені дані телеметрії – показники датчиків, справа панель де відображаються надіслані ШІМ сигнали керування, у самому низі знаходяться два джойстики керування та кнопка гальмування.

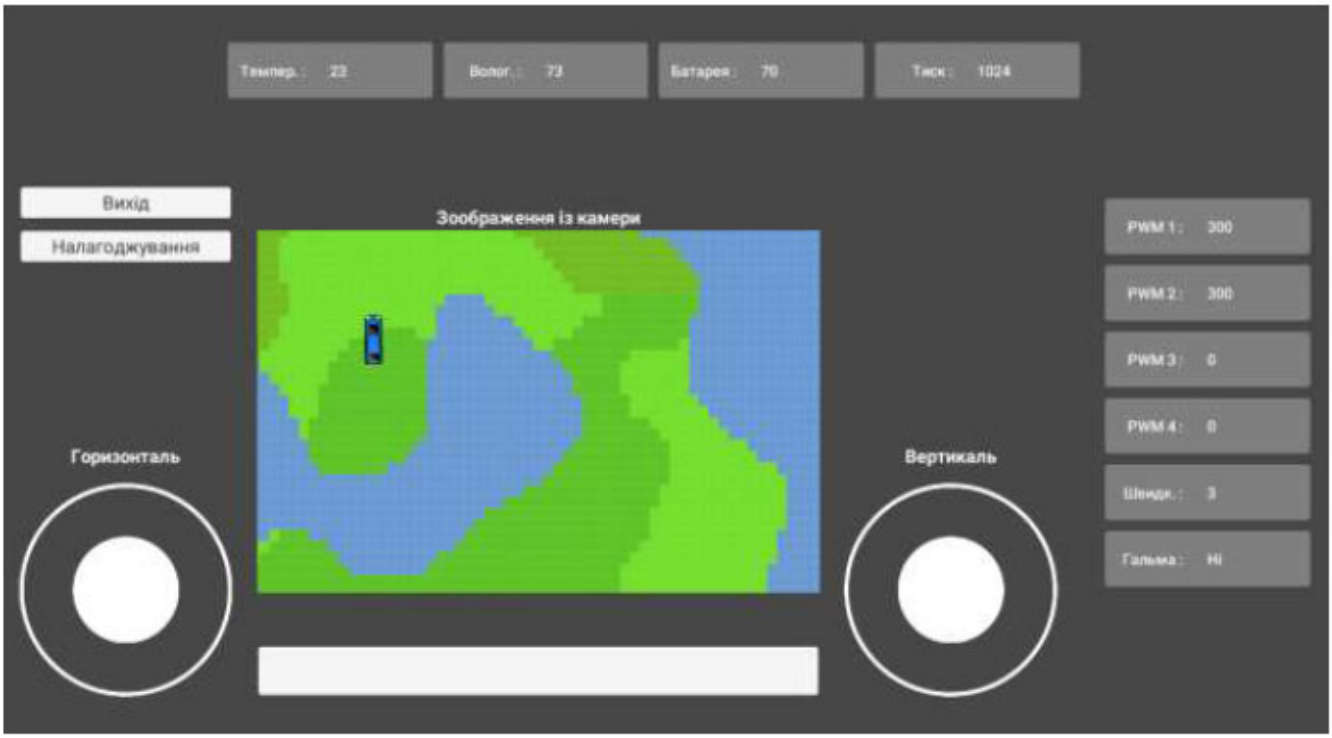


Рисунок 3.5 – Віртуальний термінал керування БПА

3.3 Дослідження роботи протоколу телеметрії

3.3.1 Дослідження сеансу зв'язку за допомогою мережевого аналізатору

Для підтвердження теоретичних відомостей та відображення працездатності протоколу доцільно провести дослідження із перехопленням пакетів мережевим програмним аналізатором Wireshark.

Для успішного перехоплення необхідно описати процес сеансу зв'язку, як приклад встановлення з'єднання:

- Відправка від термінала керування запиту на встановлення з'єднання, запит включає в себе безпосередньо сам код запиту та додаткову інформацію у яку вкладається версія протоколу.
- Дистанційно керована техніка отримавши такий запит повинна відповісти пакетом типу «Відповідь» із вкладеним у нього додатковою інформацією версія протоколу + якесь задане число. При цьому з'єднання на техніці вважається за встановлене.
- Якщо термінал керування отримав пакет із відповіддю та правильною сумою версії протоколу, то з'єднання вважається встановленим.
- Далі два хости обмінюються трафіком у який також входять службові дані для перевірки та контролю з'єднання, в разі якщо один із хостів перестане відповідати, з'єднання буде вважатися розірваним.

Для дослідження перехопленого пакету був запущений Wireshark, у режимі прослуховування локального трафіку (loopback), оскільки тестування проводилося на одному і тому самому комп'ютері то мережеві адреси хостів відрізнялися лише номерами портів. Далі було запущено два застосунки один це віртуальний термінал керування БПА та модель для налагоджування, після запуску вони почали процес встановлення з'єднання через розроблений протокол телеметрії. Один із пакетів сеансу зв'язку був перехоплений за допомогою мережевого аналізатору (рис. 3.6)

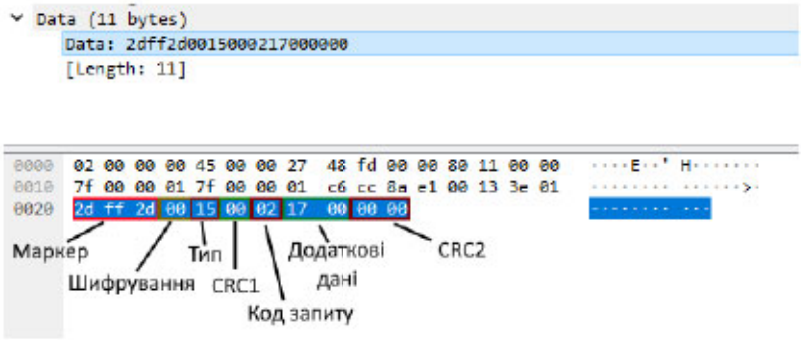


Рисунок 3.6 – Перехоплений пакет запиту встановлення з'єднання

Оглядаючи вищезгаданий рисунок (рис. 3.6) можна побачити:

- Маркер 2D, FF, 2D, що дозволяє відрізнити протокол від інших протоколів, що працюють поверх UDP.
- Флаг, що відповідає на наявність шифрування.
- Тип пакету у вигляді числа 15 – це означає тип «Запит».
- Код запиту у вигляді числа 2 – це відповідає «Запит на встановлення з'єднання»
- Додаткові дані у яких міститься номер версії протоколу.
- Також присутні контрольні суми для заголовку та корисного навантаження відповідно.

3.3.2 Побудова діаграм станів передачі протоколу

Графічні діаграми дозволяють зручно відобразити в наочному форматі процес передавання тих чи інших даних. Опишемо діаграми із зазначенням головних деталей для основних типових даних, що передаються під час роботи із дистанційно керованою технікою.

У протоколі телеметрії команди керування представлені у вигляді двох видів: за допомогою осей та ШІМ сигналів. Оскільки принцип їх передачі суттєво не відрізняється розглядати будемо на найбільш поширених ШІМ сигналах (рис. 3.7).

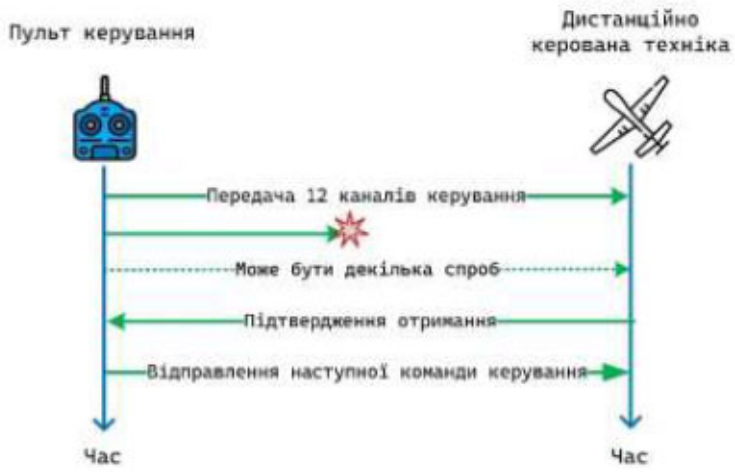


Рисунок 3.7 – Діаграма станів передачі команд керування

Опишемо головні особливості при передачі команд керування:

– Передача керуючих сигналів завжди потребує підтвердження, оскільки це є критичним по важливості для роботи дистанційно керованої техніки. Виключенням із цього правила є лише наявність більш актуальних сигналів керування, в такому випадку, застарілі дані вже не будуть потребувати підтвердження.
– Сигнали керування передаються одним мережевим пакетом, вони не роздільні, це уникнути затримок при синхронізації багатьох частин даних, крім того сигнали керування достатньо малі за об’ємом, що б реалізувати таку поведінку.

Тип повідомлень запит-відповідь (рис. 3.8), являє собою обмін короткими повідомленнями між двома хостами. За допомогою цього типу реалізований службовий трафік для контролю з’єднання, окрім цього можливо додавання довільних запитів таких як встановлення конфігурації різного типу обладнання на дистанційно керованій техніці. Цей тип повідомлень може використовується для сповіщення про події від дистанційної техніки до терміналу керування чи навпаки у зворотному напрямку. Загалом це є одним із самих універсальних типів повідомлень для невеликого об’єму інформації.

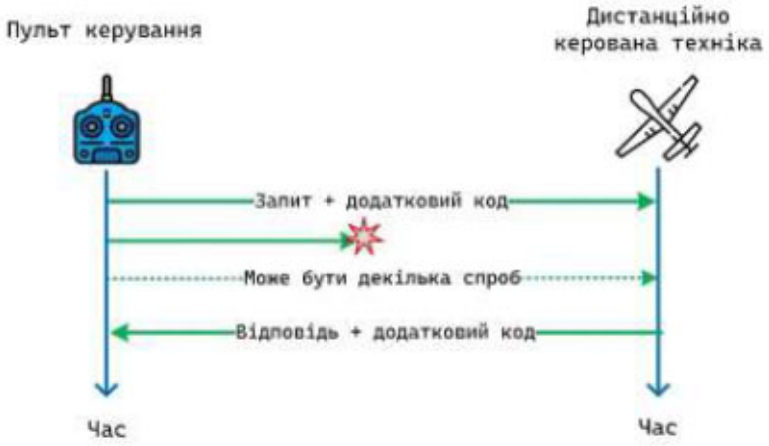


Рисунок 3.8 – Діаграма станів передачі запитів-відповідей

Опишемо головні особливості при передачі запитів та відповідей:

– Запит дозволяє швидко відправити короткі за об'ємом дані, для розширеного функціоналу є можливість відправити додаткові дані (два байти).
– Запити не завжди потребують «відповіді» у відповідь – це є не обов'язковою можливістю, що для деяких випадків буває корисно.

Передача даних аудіо чи відео формату відбувається за допомогою потокового типу даних. Такий тип дозволяє приймати послідовність кадрів відеосигналу чи імпульсно-кодованих комбінацій для передачі звуку. На діаграмі стану передачі представлений вид передачі відео (рис. 3.9).

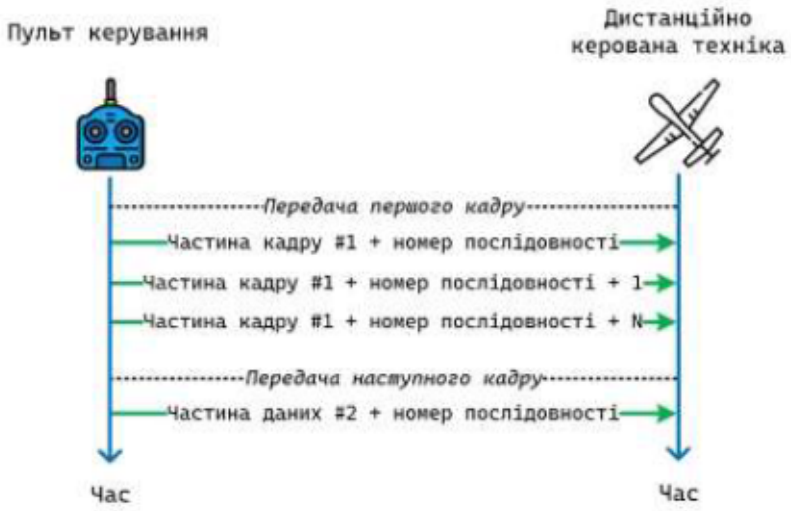


Рисунок 3.9 – Діаграма станів передачі потокових даних

Опишемо головні особливості при передачі потокових типів даних:

– Потокові дані передаються у двійковому форматі, вони поділяються на частини якщо вони перевищують максимальний розмір корисного навантаження.
– Такі дані не потребують підтвердження на прийомній стороні оскільки, через те що вони дуже швидко поступають передати нові дані є більш актуальним ніж намагатися передати старі.

Оскільки не можливо заздалегідь передбачити усі можливі формати даних, що можуть передаватися за допомогою протоколу телеметрії був створений формат для передачі даних довільного формату та розміру (рис. 3.10).

Рисунок 3.10 – Діаграма станів передачі телеметричних даних

Опишемо головні особливості при передачі довільного типу даних:

– Цей формат передає текстову строку, що є JSON форматом, для представлення різних типів даних.
– При перевищенні розміру такі дані поділяються на частини.
– Гарантія отримання пакету тут є опцією, в залежності від того, що потрібно розробнику, адже в деяких ситуаціях це корисно як наприклад передача конфігурації якогось складного обладнання, а в іншому випадку дані можливо краще передати оновлені дані.
– Загалом даний тип схожий на потокові дані із тим винятком що тут можлива гарантія доставки та тип представлення даних є текстовим.

3.3.3 Тестування роботи протоколу телеметрії

За допомогою створеної у попередніх підрозділах моделі налагоджування можна провести тестування основних функцій протоколу телеметрії. Для цього буде запущено два застосунки, один віртуальний термінал керування для дистанційної техніки та інший застосунок для моделі налагоджування. Обидва застосунки підключаються один до одного за допомогою протоколу телеметрії, у подальшому при розгляді різноманітних сценаріїв передачі даних буде матися на увазі, що з'єднання вже встановлене та гарно працює.

Для початку розглянемо найбільш поширений тип даних, що передаються при роботі із дистанційно керованою технікою – команди для керування. Оскільки модель налагоджування має двох вимірну карту, то і пересуватися можна лише по двох осях тому для зручності був обраний тип керування типу «Осі» із яких використовуються лише дві координати X та Z. Хоча як приклад можна також використовувати 4-ри ШІМ сигнали для керування моторами по 1-му на кожне мотор-колесо.

Опишемо процес тестування сигналів керування за допомого моделі налагоджування:

- За допомогою відхилення віртуальних джойстиків створюється сигнали керування. Лівий джойстик являє собою горизонтальне керування, а правий відповідно вертикальне.
- Діапазон відхилень по осям має значення від -1 до +1 таке нормалізоване значення дозволяє легко перетворювати сигнали керування у будь-який зручний формат на прийомній стороні.
- На прийомній стороні БПА, у меню детального розгляду пакету (рис. 3.11), видно що це пакет керування. Потрібно повернути колеса на 43% вправо та зусилля руху вперед повинно бути 28%.

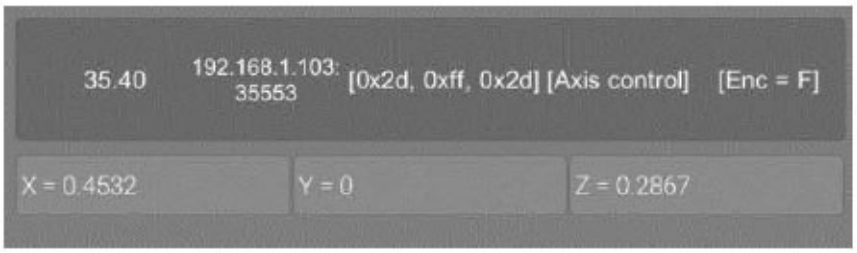


Рисунок 3.11 – Отриманий у ході тестування пакет передачі команд керування

Другим розповсюдженим типом даних для дистанційної техніки є передавання телеметричних даних, які можуть мати довільний характер. Працездатність даного типу даних є одним із головних у роботі протоколу телеметрії.

Опишемо процес тестування довільних телеметричних даних за допомого моделі налагоджування:

- Віртуальний БПА через деякі проміжки часу збирає дані у середовищі налагоджування протоколу. Ці дані імітують показники реальних сенсорів на дистанційно керованій техніці.
- Показники із датчиків та інша додаткова інформація конвертується у текстовий формат.
- На рисунку нижче (рис. 3.12), можна бачити приклад прийнятого пакету телеметричних даних у форматі JSON. Тут є номер послідовності 1, що означає першу частину повідомлення, значення швидкості, вологості температури тощо. Крім того видно масив заданих швидкостей ШІМ сигналів, що користувацьким налаштуванням.

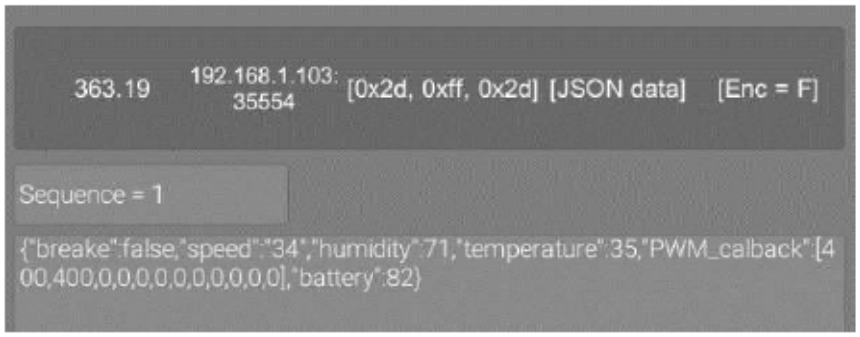


Рисунок 3.12 – Тестування передачі телеметричних даних із датчиків

Потокові відео та аудіо дані також є типовими для більшості дистанційно керованої техніки. Тому доцільно розглянути під час тестування у моделі налагоджування передачу відеозображення. Оскільки віртуальна модель не має реальної камери то для перевірки було обрано передачу кадрів із віртуальної камери що робить захоплення екрану у моделі налагоджування та передає таким чином кадри.

– У моделі налагоджування працює віртуальна камера передає знімки екрану через однакові проміжки часу, черговість таких кадрів своїм чином формує відео.
– Кадр може ділитися на дрібніші частини, якщо він перевищує розмір максимального пакету даних.
– На рисунку нижче (рис. 3.13) у розгляді детальної структури пакету, можна побачити номер послідовності, що означає, що це третя частина кадру та частину бітової послідовності, що являє собою значення пікселів у JPEG форматі.

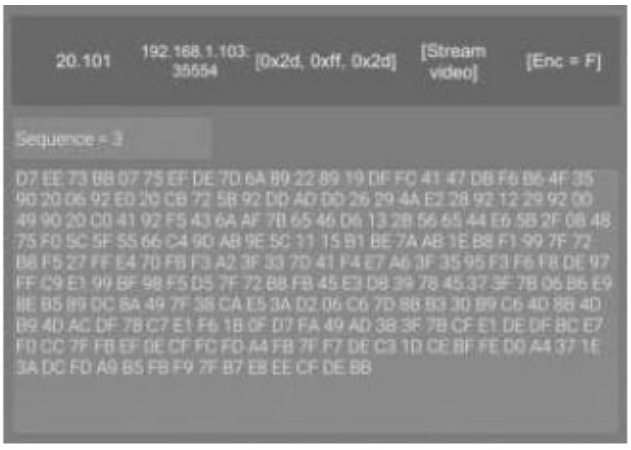


Рисунок 3.13 – Тестування потокових аудіо-відео даних

ВИСНОВОК

Даний розділ описував створення віртуальної моделі для подальшого тестування протоколу телеметрії. Розроблення моделі було розбито на три етапи: розроблення графічного інтерфейсу користувача, створення віртуальної дистанційно керованої техніки та інтеграція протоколу телеметрії у розроблений проект. Це дозволило протестувати основні типи даних, а саме: телеметричні та довільні дані у текстовому форматі JSON, потокові дані призначення яких аудіо та відео формати, запити відповіді, що можуть передавати як службові повідомлення для контролю з'єднання так і корисні запити типу встановлення конфігурації обладнання, а також було проведено тестування даних керування двох видів – за допомогою 3-х осей та 12-ти ШІМ каналів.

Крім того за допомогою програмної моделі налагоджування та мережевого аналізатору були опрацьовані на практиці основі механізми роботи протоколу: встановлення та контроль сеансу зв'язку, гарантія доставки та перевірка помилок при передачі.

Отже у даному розділі було встановлено працездатність створеного протоколу телеметрії для керування різного роду дистанційною технікою.

ЗАГАЛЬНІ ВИСНОВКИ

Метою даної кваліфікаційної роботи є розробка протоколу телеметрії для дистанційно-керованої техніки та його програмної реалізація.

У першому розділі було розглянуто основні теоретичні відомості про мережеві протоколи передачі даних. Описані ключові поняття, що необхідні для розробки такі як програмний мережевий інтерфейс та типи сучасних телеметричних протоколів, що використовуються на БПА. Проведений аналіз існуючих протоколів передачі даних та супутньої теорії, дозволив розпочати подальшу розробку протоколу телеметрії.

У другому розділі було розроблено специфікацію майбутнього протоколу телеметрії, вивівши основні вимоги та користуючись отриманою у ході попереднього аналізу теорією було розпочато програмну реалізацію протоколу телеметрії за допомогою обраних інструментів: програмного середовища Unity та мережевого аналізатору. У ході розробки було обрано транспортний протокол, формат представлення даних для корисних даних, описані характеристики та структура пакетів протоколу телеметрії.

У третьому розділі було створено застосунки: термінал керування та віртуальну модель із середовищем та БПА, що являє собою віртуальну машинку із датчиками, що збирають показники які випадковим чином генеруються у віртуальному середовищі, крім цього наявна імітація камери що відправляє знімки екрану на термінал керування. Створені застосунки моделі та терміналу керуванні дозволили провести тестування коректності передачі типів даних – запитів, відповідей, потокових та текстових форматів, крім цього протестовані команди керування за допомого ШІМ сигналів та нормалізованих значень із осей.

Отже у ході даної кваліфікаційної роботи було виконано поставлені перед нею завдання, а саме розроблення протоколу телеметрії для керування дистанційної техніки та програмна реалізація цього протоколу у вигляді двох додатків – моделі для налагоджування та терміналу керування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Мережевий протокол [Електронний ресурс] – Режим доступу до ресурсу: 1. https://uk.wikipedia.org/wiki/Мережевий_протокол.
2. Микитишин А. Г. Комп'ютерні мережі / А. Г. Микитишин, М. М. Митник, В. В. Стухляк. – Львів: Магнолія 2006, 2013. – 256 с.
3. Craig H. TCP/IP Network Administration / Hunt Craig., 2002. – 746 с.
4. TCP/IP Overview [Електронний ресурс] – Режим доступу до ресурсу: https://www.cisco.com/c/en/us/support/docs/ip/routing-information-protocol-rip/13769-5.html.
5. Transmission Control Protocol [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
6. Take Total Control Of Your Networking With .NET And UDP [Електронний ресурс] – Режим доступу до ресурсу: https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/february/udp-delivers-take-total-control-of-your-networking-with-net-and-udp.
7. User Datagram Protocol [Електронний ресурс] – Режим доступу до ресурсу:. https://en.wikipedia.org/wiki/User_Datagram_Protocol.
8. Beej's Guide to Network Programming [Електронний ресурс] – Режим доступу до ресурсу: https://beej.us/guide/bgnet/.
9. Networking Basics [Електронний ресурс] – Режим доступу до ресурсу: https://docs.oracle.com/javase/tutorial/networking/overview/networking.html.
10. OSI Layer 7 - Application Layer [Електронний ресурс] – Режим доступу до ресурсу: https://osi-model.com/application-layer/.
11. Ryan L. Designing and Implementing an Application Layer Network Protocol Using UNIX Sockets and TCP / Ryan Lattrel, 2012.
12. The Beginners' Guide to RC Protocols [Електронний ресурс] – Режим доступу до ресурсу: https://www.rcgroups.com/forums/showthread.php?2301242-The-Beginners-Guide-to-RC-Protocols.
13. RC Protocols Explained [Електронний ресурс] – Режим доступу до ресурсу: https://oscarliang.com/rc-protocols/.

14. Aranzazu C. S. 14. Unmanned Aerial Vehicle Networking Protocols / Aranzazu Catalina Suescun, 2016.
15. Dai C. Communication among UAVs / Dai Chaoyou, 2010.
16. Low-Level Protocol ТЛ Tello [Електронний ресурс] – Режим доступу до ресурсу: https://tellopilots.com/wiki/protocol/.
17. MAVLink Developer Guide [Електронний ресурс] – Режим доступу до ресурсу: https://mavlink.io/en/.
18. UAVCan Specification [Електронний ресурс] – Режим доступу до ресурсу: https://legacy.uavcan.org/Specification/.
19. Kriz V. UranusLink - Communication Protocol for UAV with Small Overhead and Encryption Ability / Kriz Vlastimil, 2015
20. Developer Mozilla. Клієнт-сервер [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/ru/docs/Learn/Server-side/First_steps/Client-Server_overview.
21. Технологія "клієнт/сервер" ч. І. Основні ідеї та положення / В. Д. Вовк, Б.М. Голуб, А.В. Дубовик – Львів, 1999.
22. Stateful vs Stateless: Full Difference [Електронний ресурс] – Режим доступу до ресурсу: https://www.interviewbit.com/blog/stateful-vs-stateless/.
23. Струбицький Р.П. Розробка протоколу сеансового рівня для високошвидкісних регіонально-розподілених мереж / Струбицький Р.П.. – 2015.
24. What is a packet? | Network packet definition [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudflare.com/learning/network-layer/what-is-a-packet/.
25. Rescorla E. Writing Protocol Models RFC 4101 - IETF / Rescorla E., 2005. – 22 с.
26. Guaranteed Delivery of Data [Електронний ресурс] – Режим доступу до ресурсу: https://community.rti.com/static/documentation/connext-dds/5.2.0/doc/manuals/connext_dds/html_files/RTI_ConnextDDS_CoreLibraries_UsersManual/Content/UsersManual/GuaranteedDelivery.htm.

27. What is MTU (maximum transmission unit)? [Електронний ресурс] – Режим доступу до ресурсу: https://www.cloudflare.com/learning/network-layer/what-is-mtu/.
28. What is Cyclic Redundancy Check (CRC)? [Електронний ресурс] – Режим доступу до ресурсу: https://info.support.huawei.com/info-finder/encyclopedia/en/CRC.html.
29. Encryption: What Is It and How Does Encryption Work? [Електронний ресурс] – Режим доступу до ресурсу: https://vpnoverview.com/internet-safety/secure-browsing/what-is-encryption/.
30. 11 Best Network Traffic Analyzers For Windows, Mac & Linux In 2022 [Електронний ресурс] – Режим доступу до ресурсу: https://www.softwaretestinghelp.com/top-network-traffic-analyzers/.
31. A tour of the C# language [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/.
32. The 12 best IDEs for programming [Електронний ресурс] – Режим доступу до ресурсу: https://www.techrepublic.com/article/best-ide-software/.
33. JSON - Introduction [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/js/js_json_intro.asp.
34. Rijndael Class [Електронний ресурс] – Режим доступу до ресурсу: https://learn.microsoft.com/en-us/dotnet/api/system.security.cryptography.rijndael?view=net-7.0.
35. Unity UI: Unity User Interface [Електронний ресурс] – Режим доступу до ресурсу: https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html.

ДОДАТОК А

Переклад першого розділу

1 THEORY AND ANALYSIS OF REMOTE-CONTROLLED EQUIPMENT PROTOCOLS

1.1 General information about data transfer protocols

1.1.1 Network communication protocol

A network communication protocol is a description of rules or agreements for information exchange. Such agreements allow different devices or programs to exchange messages with a previously known structure [1]. So, this is a necessary set of rules that forms a data format.

Depending on the task, the protocol may define a number of certain requirements or properties. Let's list the most typical requirements for network communication protocols [1, 2]:

- The size of the data packet being transmitted.
- Minimum and maximum transmission speed and its coordination.
- Ability to correct errors during transmission.
- The process of establishing a connection, the so-called handshake.
- Subscriber authorization.
- Routing – routing traffic according to criteria.
- Data flow control.

Between sender and receiver, a message consists of discrete blocks, where each individual data packet has its own header, body information, and routing path. All this must be synchronized and adjusted to the smallest detail.

There are a large number of communication protocols, some of them implemented in hardware, others in software. They are used in all digital and analog communications. What they all have in common is that they allow different devices to exchange messages, that is, protocols perform the role of standardization in the field of telecommunications.

1.1.2 OSI and TCP/IP network models

Modern communication protocols do not work in isolation from each other and depend on other protocols, this dependence is shown in multi-layer models. Each higher level in the model depends on the levels below. The most famous and popular models today are two network models: OSI and TCP/IP [3].

The OSI network model is a conceptual framework that declares network or telecommunication systems in the form of seven layers, each of which performs its own function.

Let's list the levels of the network model with an indication of the functions of each [2, 3]:

8. Physical layer - all physical representations of the network are located here: electrical cables, radio frequency channels, that is, all physical connections and signal exchange that make up the communication channel.
9. Channel layer - this layer provides the ability to transmit data between two end nodes and the processing of corrections of errors associated with transmission.
10. Network layer - the network layer deals with the routing of transmitted data packets, it paves the best path according to the given criteria.
11. Transport layer - at this level, the coordination of data transfer between different end systems takes place, for example: setting the speed, destination and size of the data to be sent.
12. Session level - used to establish a communication session between different network devices.
13. Presentation Layer – This layer is concerned with converting session-level data format to application-level data format and vice versa. An example would be encryption and decryption of a transmission stream.
14. Application level - interacts directly with the end user.

Since the OSI model is an idealized network model, in practice a simplified TCP/IP model is used, which has four levels. TCP/IP does not follow the OSI model directly.

TCP/IP either combines several OSI layers into one, or does not use some layers at all. Let's list the four levels of TCP/IP [3]:

5. Channel Layer – This layer defines how data is sent, handles the physical process of sending and receiving data, and is responsible for transferring data between applications or devices on a network.
6. Network Layer – Controls the sending and moving of packets over the network to ensure that they reach their destination.
7. Transport layer – establishes and supports the connection and exchange of data between two devices. This is the layer where data is fragmented into packets and numbered to create a sequence of packets.
8. Application level – processes data at the level of programs that use TCP/IP to communicate with each other. This is the layer that users typically interact with, such as e-mail systems or an Internet browser. It combines the session, presentation, and application layers of the OSI model.

The relationship between the network layers of the OSI and TCP/IP models and the corresponding protocols can be seen in image 1.

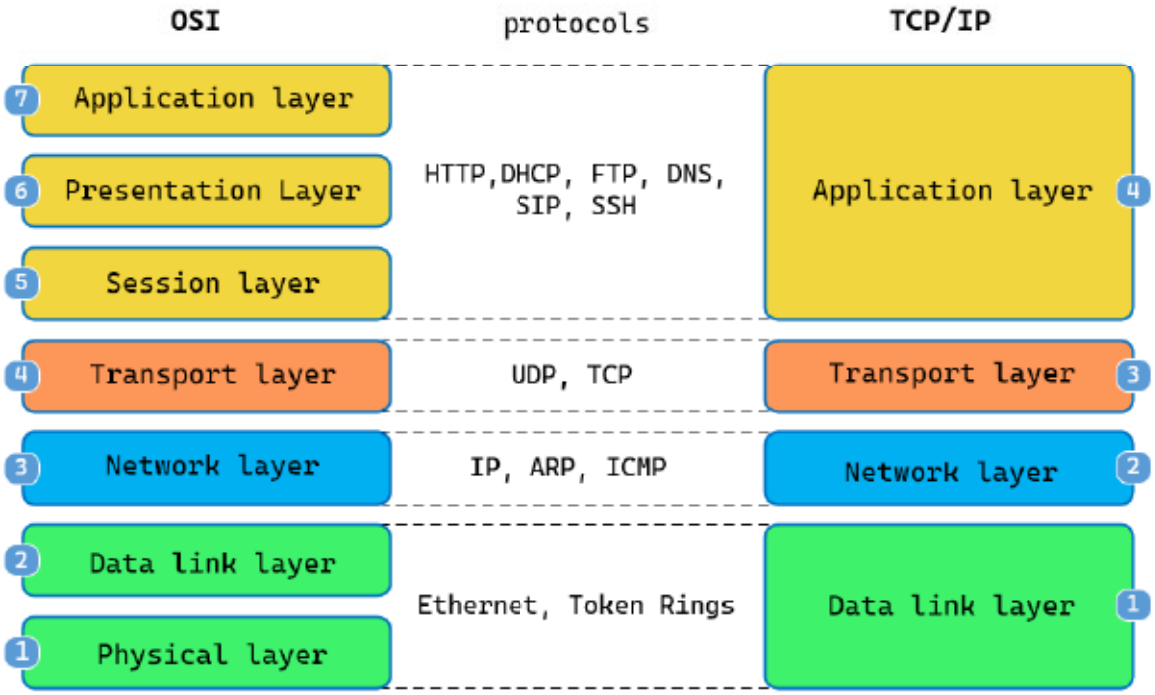


Image 1 – OSI and TCP/IP network models

At each level of the network model, information is presented in the form of protocol data fragments – PDU (Protocol Data Unit).

The process of converting data from a higher level to a lower one is called encapsulation, in this case each encapsulation process is accompanied by the addition of

additional data (headers) required for the operation of the lower protocol. Conversely, the process of converting from a lower level to a higher one is called decapsulation, during which redundant information is removed because it is no longer needed.

1.2 Transport layer protocols

Since the developed telemetry data transfer protocol is high-level, that is, application level, its basis will be the transport level protocol. Therefore, it is advisable to consider the principle of operation of the two most popular transport protocols - UDP and TCP.

1.2.1 TCP protocol

TCP – it is a protocol that provides opportunities for data transport, that is, their transmission over the network. Its main features are: reliability, integrity and confirmation during data transmission.

TCP transmits data in the form of a data stream, which in turn is divided into segments. This protocol is focused on establishing a connection between subscribers before the start of data transmission and on completing such a connection after the end of the transmission [4].

TCP must establish a connection before any data can be transferred. Establishing a connection consists of three messages (three times handshake). This process looks like this (Image 2) [4].

4. The party that wants to establish a connection sends a message with a synchronization bit (SYN) to the second subscriber about establishing a connection. This message specifies the port number for the connection and the sequence number required for segment numbering during transmission.
5. The second subscriber, in turn, responds with a message confirming the establishment of communication (ACK) and its sequence number.
6. The party that initiated the connection confirms the consent of the second subscriber and the connection is established.

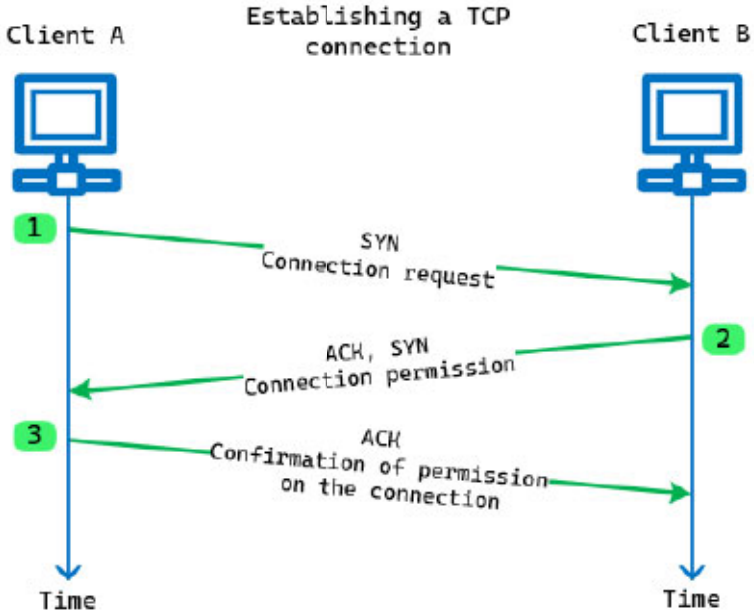


Image 2 – Three-way handshake in the TCP protocol

The end of the data transfer should end the established connection. This process looks like this (Image 3) [4]:

5. The party transmitting the data stream sets the bit (FIN) in the last segment, which means the end of the connection.
6. The second party responds to the first party with an acknowledgment (ACK) bit to indicate that the connection is complete. At this time, the program using this connection is notified of its termination.
7. When the application has completed the connection, a connection completion bit (FIN) message is sent to the first party.
8. The first party sends an acknowledgment (ACK) bit to the second party in the message that the connection is complete.

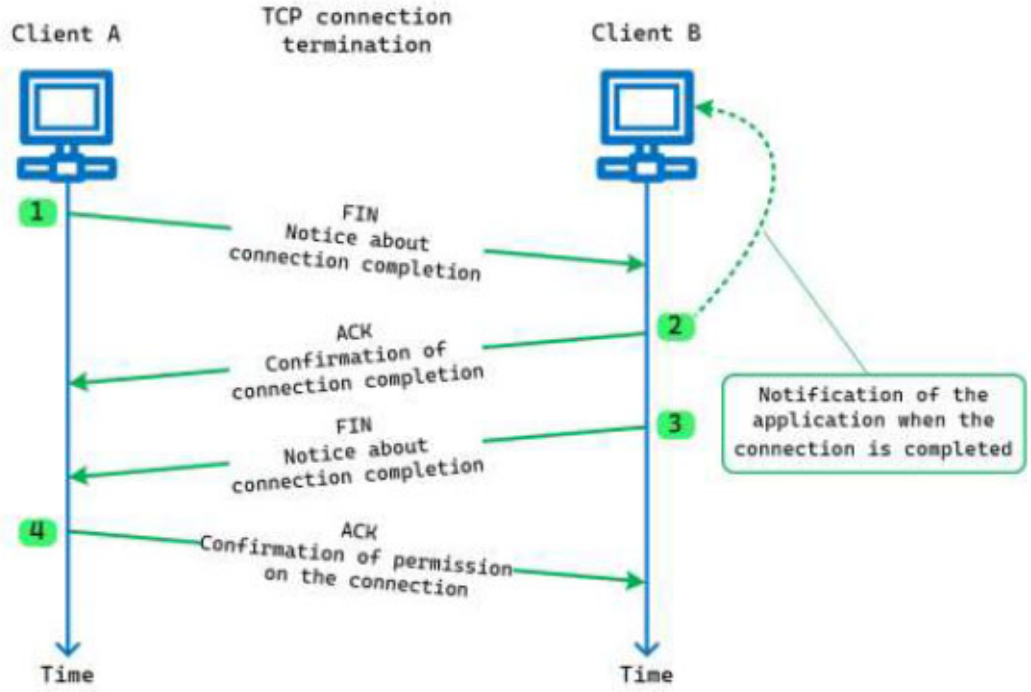


Image 3 - The four-step process of terminating a connection in the TCP protocol

The TCP protocol guarantees the reliable delivery of data and their order during transmission over the network. Let's consider the main properties of the protocol due to which the aforementioned characteristics are achieved [4, 5].

Message delivery guarantee:

- TCP requires an acknowledgment message (ACK bit) from the receiving party to receive each segment or group of segments that were sent. This is an implementation of acknowledgment of receipt.
- If, for any reason, the message about receiving the segment was not received on the sender's side, the sender will try to send the data again after a set time. This is how resending is implemented.

Data flow control. To optimize data transmission, the TCP protocol uses the so-called "window", which is the number of segments that can be accepted by the receiving party without confirmation. Due to this, it is possible to reduce the number of transmissions of messages about receiving segments.

The sequence of segments during transmission. When transmitting over a network, the receiving party may receive several identical segments or receive them in the wrong order. The TCP protocol numbers each portion of data (bytes) and during transmission

includes the number of the first byte of data contained in the segment. On the receiving side, it is sufficient to compare the segments with those for anti-duplication or to put them in the correct order by number. This is how the mechanism of protection against duplication and violation of data tracking is implemented.

Therefore, this protocol has high reliability in data transmission, the end subscriber is guaranteed the delivery of messages and their tracking. However, it has the overhead of establishing a connection between subscribers every time something needs to be transferred.

1.2.2 UDP protocol

The UDP protocol is a communication protocol for data transfer that operates at the transport layer. Its main features are minimal redundancy in data transmission due to the lack of guarantees of information delivery.

Data transmitted using the UDP protocol are divided into messages and are called datagrams or datagrams. Unlike TCP, the UDP protocol does not require prior establishment of a connection between hosts in the network for data transmission [6].

The UDP protocol has a "simple" structure and consists of a small number of elements:

- Sender and recipient ports.
- Datagram lengths
- Data (payload)
- Checksum

Let's list the main features of the UDP protocol [6, 7]:

- There is no control over the transmission of messages. There is no guarantee of delivery of messages and verification of their duplicates.
- There is no data order control.
- The sender does not know whether the message was delivered.
- It is assumed that all missing control mechanisms, if needed, will be implemented at a higher level – applied.

- High transmission speed (compared to TCP) because there are no additional checks, establishment and monitoring of the data connection.
- In contrast to TCP, which is oriented to the "one-to-one" model. The UDP protocol has the ability to send data simultaneously to many receivers.

And although UPD does not have any additional mechanisms to control the reliability of data transmission, it is used. Due to lower overheads, the UDP protocol is used for traffic in which the loss of several messages is not so significant and the delay time when transmitting information is more important, for example, for a voice or video stream in real time.

1.3 A software network interface – a socket

A network socket is a software abstraction for accessing data exchange between different software processes that have a network connection. The detailed structure and properties of the network socket are determined by the application programming interface – API (usually this is determined by the type of OS used) [8, 9].

Socket provides:

- Two-way communication between programs running on the network.
- The socket implements the encapsulation of network and transport layer protocols, usually IP, TCP, UDP.
- A socket contains an IP address and a port, this combination is called a socket address.

Network sockets are divided into types [8, 9]:

- Streaming sockets – sending a stream of bytes with guaranteed delivery. A stream socket provides the transmission of a stream of bytes with a guarantee of delivery, checking the correctness of their sequence, protection against duplication of data. Typically, streaming sockets use the TCP protocol.
- Datagram sockets – individual messages without delivery guarantee. Reliability and order of follow-up during data transmission is not guaranteed. UDP protocol is usually used for such a socket.

- Raw sockets – such sockets allow receiving and sending data without using any of the protocols. In this socket, unlike others, there is no automatic encapsulation from transport layer protocols, so protocol headers are also transmitted for further processing.

So, in general, a socket is a way of interaction of various programs with each other through the network. In essence, a network socket is a combination of a network address and a port.

1.4 Types and classification of applied data transmission protocols

For remote control and telemetry protocol, you need to stream audio and video data and transmit sensor readings. Taking into account the above, it will be appropriate to consider some of the main application protocols used to transfer such data.

By purpose, protocols can be classified [10, 11]:

- Transmission of streaming video/audio data – such protocols transmit a set of bytes representing video or audio signals. These protocols usually have flow control elements, its correction. They are characterized by low transmission latency due to the content they transmit.
- Transmission of some indicators (values from sensors) - these protocols transmit data from sensors in the form of numbers or a time representation, can serve a large number of simultaneously working clients in the network. They are characterized by a fairly wide range of possible delays, depending on the priority of the importance of the transmitted data.
- Remote control – the purpose of these protocols is to transmit control commands to remote objects. Such protocols are characterized by the possibility of user authentication and reliability in data transmission. Typically, they should have the lowest possible transmission delay.
- Universal protocols – these protocols, having a flexible structure, can perform a wide range of tasks.

Protocols are classified by the type of data they transfer [11]:

- Text – transmit useful data in the form of a set of symbols. All non-text data must be converted to string format. Among the advantages of such a protocol is the possibility of easy debugging, since such a protocol is easy to read by a person. The text protocol allows you to be independent of the final implementation of specific data types, which provides greater compatibility. But this type of protocol is not as fast as binary because it requires additional time each time the type is converted to a time representation and the transmitted packets have redundant information.
- Binary - such protocols transmit information in the form of a set of bytes, representing the objects to be transmitted. Binary protocols are faster than text protocols because they do not need to convert types to string representation and have much less redundant information. However, the disadvantage is their more complex implementation and the need for type compatibility on end devices.

1.5 Analysis and classification of existing types of protocols for transmission of telemetry data and remote control

1.5.1 Protocols of remotely controlled equipment

Modern protocols for remotely controlled equipment are divided into two groups [12]:

- RX – internal communication. Those designed for data transmission from the receiver to the equipment controller.
- TX – external communication. For transmission between the control panel and the equipment receiver.

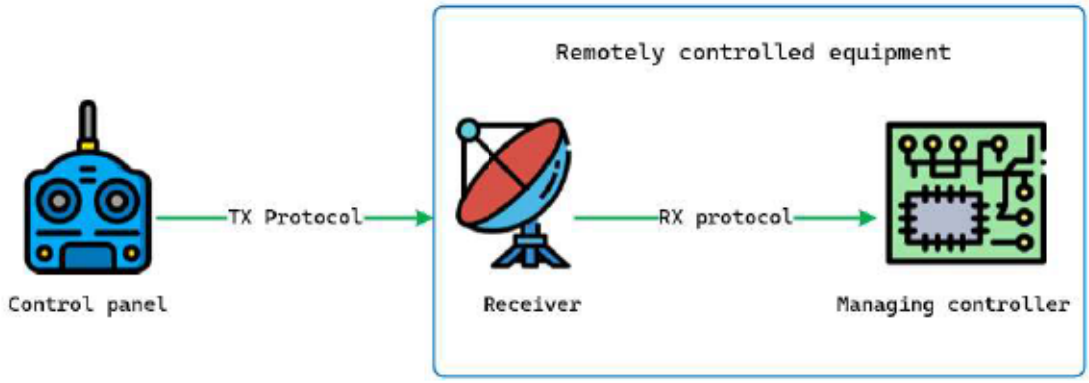


Image 4 – Interplay of TX and RX protocol types for remotely controlled equipment

1.5.2 Internal protocols of remote technology

Typically, the internal protocols (RX) are a wired communication line that is physically located in the remote controlled equipment between the signal receiver and the control controller.

Let's list the main types of RX protocols [12, 13]:

- PWM – control using pulse width modulation. It is one of the most common control protocols. Control requires one wire for each actuator (for example motor). The advantage of this method is the relatively easy implementation of this protocol and its compatibility with the equipment. The main disadvantage is the large number of connections to each actuator.
- PPM – control through phase-pulse modulation (PIM). Represents the transmission of a set of PWM signals over the same wire, but with different modulation. The advantage of this method is the use of significantly fewer wires. Because one wire can transmit up to 8 control signals.
- Sequential protocols that transmit sequential series of control signals. Such protocols are completely digital. A vivid example of such protocols are UART, I2C, SPI, CAN and others.

1.5.3 External BPA control protocols

External protocols (TX) are needed to control remote equipment, to communicate with a ground station or other unmanned vehicles [13].

Remote control of BPA can be:

- Manual – the control of the device is fully entrusted to the operator.
- Auxiliary - the operator sets the main route or list of actions and, if necessary, makes corrections during execution.
- Fully autonomous – operator intervention is required only in the presence of unforeseen or emergency situations.

One of the types of control of remote equipment in the modern world is analog radio channels:

- At frequencies of 27 MHz and 35 MHz – for old types of radio communication. Among the advantages is a long range, but high sensitivity to mutual interference.
- Frequencies: 433/868/915 MHz - also one of the outdated types of communication, but still popular.
- At frequencies of 2.4 GHz and 5.8 GHz - modern digital communication protocols.

Let's list some of the main external (TX) protocols for controlling remote equipment. Analog remote control protocols [13, 14]:

- Pulse width modulation (PWM).
- Phase-pulse modulation (PPM).
- Pulse code modulation (PCM).

Analog protocols are easy to implement, have simple hardware decoders, have very high speed and low latency. Disadvantages include interference and a limited number of control channels.

Digital remote control protocols:

- SBUS – supports up to 18 PWM control channels coded into one channel (radio frequency).
- DSM2, DSMX is one of the most popular protocols. Supports up to 98 channels. Works at a frequency of 2.4 - 2483 GHz. It has a main and a backup channel, to avoid mutual interference and the possibility of error correction.
- Satellite – this protocol is an extension of the DSM protocol. The main feature is the improvement of the control range due to the use of additional antennas.

Digital protocols with support for duplex transmission of control commands and telemetry data [12, 13, 14]:

- Wi-Fi – this protocol together with TCP/UDP transport protocols is used for remote control of equipment. It requires a relatively large amount of energy during transmission, has a small radius of action (up to 150 meters), but is quite easily

implemented on modern equipment. The main advantage is that it is possible to use mobile phones and laptops as a control terminal, since they already support this protocol.

- Bluetooth - has good support for various equipment. The main disadvantages are not the ability to transmit a large amount of traffic and the limited range of action (10-20 m)
- IBUS – supports transmission in two directions (duplex). Allows transmission of both control commands and telemetry data from the BPA. Supports up to 16 telemetry indicators. Transmits data in binary form.
- MSP – text protocol. At the application level, it uses the UART serial protocol. It is duplex, that is, it allows transmitting telemetry data as well. It has the possibility of remote configuration of the BPA, setting of the plan of autonomous movement.
- FrSky Telemetry – compatible with a serial interface (UART). The range is approximately 1 km (depending on the maximum signal strength). A binary protocol with an open specification. Supports a limited number of defined sensor sets.
- DJI Tello is a protocol that works on the Wi-Fi protocol using UDP as a transport protocol. Supports connection modes such as "one-to-one" and "one control station - many BPAs". The protocol uses a text representation of the data. Can transmit: control commands, telemetry and streaming video.

1.5.4 Transmission of streaming media data

For the transmission of audio or video signals, analog protocols are the most common, as they have the lowest possible delay. However, as the communication range increases, interference in analog communication systems increases.

Let's list the main features of video transmission from the BPA to the control terminal [15]:

- Modern video transmission protocols operate at a frequency of 5.8 GHz and sometimes at 2.4 GHz.

- Analog video formats are the most popular in FPV control. NTSC and PAL TV formats are used as video formats.
- Digital data transfer formats – MJPEG, MPEG streams (H264/H265). Digital formats usually work well, but have a fairly high transmission delay (sometimes up to 1s). But with increased transmission range, they have an advantage due to coding and transmission loss recovery.
- The main types of current resolutions for streaming video are 720p, 1080p, 4k, 5k and 8k. Usually, the operator receives poorer quality video (due to transmission limitations), better quality is recorded on the flash memory built into the BPA.

1.6 The internal structure of modern protocols of telemetry and control of BPA

1.6.1 TJI Tello protocol

This protocol is a development of the manufacturer TJI. It works on top of a Wi-Fi network and uses UDP datagrams as transport. The standard port for its operation is 8889 [16].

Let's take a closer look at the structure of the Tello package in the table. 1.1:

Table 1.1 – TJI Tello protocol stack structure

Position (Byte)	Data description	Additional Information
0	Title	It always is 0xCC
1-2	Total package size	13- bit value
3	Header checksum CRC-8	From header to packet size
4	Package type	Bits of type: F;T;TYP;SUB
5-6	Message identifier	
7-8	Sequence number	
9...	Payload (data)	Optional
End	The checksum of the entire package CRC16	From the header to the end of the payload

Let's describe the meaning of some elements in the data structure of the Tello package [16]:

- Packet type – this can be a request for information, multiple data types, or a configuration setup type.
- Message ID – specific purpose of the payload, for example: Wi-Fi or video settings, command to start movement, log data, calibrate sensors, etc.

Data can be transferred in both directions. The data is divided into fragments, and each fragment, in turn, is divided into 8 chunks (pieces). Each fragment, i.e. 8 chunks of data are transmitted and need confirmation that they have been delivered or will be sent again.

1.6.2 MAVLink protocol

MAVLink is a modern open protocol for communication with BPA. It supports Publish-Subscribe and Point-to-Point connections. This protocol has a slight redundancy as the minimum message size is 8 bytes (service data) and 14 bytes for MAVLink version 2 [17].

We will describe the main characteristics of the MAVLink protocol [17]:

- Support for managing up to 255 devices and other equipment.
- Works on a wide range of hardware: ATMega, ARM7, dsPic, STM32 microprocessors and Windows, Linux, MacOS, Android and iOS.
- High reliability of the protocol, it can discard erroneous packets and has a good checksum algorithm to detect corruption.
- Ensures security in the message "subscription" model. Because it allows authentication. However, it does not provide traffic encryption.

Let's consider in more detail the structure of the MAVLink 1 package in the table. 1.2:

Table 1.2 – MAVLink 1 protocol packet structure

Position (byte)	Data description	Additional Information
0	Title	It always is 0xFE
1	Load length	
2	Sequence number	
3	System ID	The identifier of the remote equipment in the network
4	Component identifier	The identifier of the component in the technique
5	Message identifier	
6…	Payload (data)	
End	Checksum CRC-16	Packet checksum without header

MAVLink does not describe by which hardware or software the message will be sent, it can be serial protocols or TCP/UPD messages.

1.6.3 UAVCan protocol

UAVCan is an open protocol for UAV control and telemetry built on the industrial CAN protocol [18].

Let's list the main characteristics of the UAVCan protocol [18]:

- All nodes in the network have the same rights, that is, there is no single point of failure.
- Support for backup nodes.
- Due to simple logic, low requirements for computing resources.
- Can work in hard real-time, i.e. low latency and high throughput.

UAVCan is a decentralized network where each node has its own ID. Nodes can exchange two types of messages: broadcast messages (subscription) and service messages (requests).

The main mechanism of communication between nodes in the UAVCan network is the method of using broadcast messages. Since the UAVCan frame is encapsulated in CAN, we describe only the frame of the broadcast message packet in the table. 1.3:

Table 1.3 – UAVCan protocol broadcast message frame structure

Position (byte)	Data description	Additional Information
0	Source ID	Number of the initiator node
1-2	Message type	
3	Priority	

Creating requests, i.e. starting services in the UAVCan network, is done through service messages. We will describe only the frame of the official message in the table. 1.4:

Table 4 – Structure of the service message frame of the UAVCan protocol

Position (byte)	Data description	Additional Information
0	Source ID	Number of the initiator node
1	Destination ID	End node number
2	Service type	
3	Priority	

1.6.4 UranusLink protocol

UranusLink is a packet-oriented protocol for controlling remotely controlled equipment. The protocol was designed considering data loss or incorrect data acquisition [19].

We describe the structure of the UranusLink service message package - table. 1.5:

Table 1.5 – UranusLink protocol service message frame structure

Position (byte)	Data description	Additional Information
0	Title	It is always 0xFD
1	Sequence number	
2	Message identifier	
3	Data length	
4	Useful data	
5	Checksum	

Let's describe some differences of the UranusLink protocol [19]:

- The sequence number is a range from 0x0000 to 0xFCFE (since it is a header). The sequence is incremented by 2 after each subsequent packet is sent.
- The connection from the BPA to the control station can transmit: all streaming data (audio, video) and data from sensors.
- The connection from the control station to the UPS can transmit control commands and configuration commands.
- UranusLink has 33% less overhead than MAVLink.

CONCLUSION

In the first chapter, the basic theory of data transfer protocols was considered. In particular, the purpose and description of the network communication protocol and network models used as a standard in the development of protocols - OSI and TCP/IP. The structure and principle of operation of the two most popular protocols used at the transport level of data transmission – TCP and UDP – were described.

With a view to further development, the structure and purpose of the socket, which acts as a network interface for software, has been described.

The classification of applied data transmission protocols used in remotely controlled equipment was carried out according to the purpose and type of data transmitted.

The protocols used on remote equipment were divided into two groups - internal and external communication. During the analysis of modern telemetry protocols, their main features were revealed when working with different types of traffic. The internal structure of popular telemetry and control protocols of BPA was also considered.

Thanks to the analytical work that was carried out on the existing data transfer protocols, in particular those used for working with BPA, it is possible to start the description and development of the future protocol in the next section.

ДОДАТОК Б

Лістинг програмного коду протоколу телеметрії

Файл ControlAxisPacket.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
internal class ControlAxisPacket : Packet
{
    public float[] axis;
    public ControlAxisPacket() => axis = new float[Settings.axisCount];
    public ControlAxisPacket(bool encryption, float[] axis) :
base(encryption, (byte)PacketType.Controls.Axis)
    {
        if (axis == null || axis.Length != Settings.axisCount)
        {
            Debug.Log("Axis is bad!");
            return;
        }
        this.axis = axis;
        CalculateCRC2();
    }
    public ControlAxisPacket(float[] axis) : this(false, axis) { }
    public override void FromBytes(byte[] bytes)
    {
        byte[] packetBytes = ParseBytes(bytes);
        for (int i = 0, offset = 0; i < Settings.axisCount; i++, offset +=
4)
            axis[i] = BitConverter.ToSingle(packetBytes, offset);
    }
    public override byte[] ToBytesBody()
    {
        List<byte> bytes = new List<byte>();
        for (int i = 0; i < axis.Length; i++)
            bytes.AddRange(BitConverter.GetBytes(axis[i]));
        return bytes.ToArray();
    }
    public override string ToString()
    {
        string axis_s = "";
        for (int i = 0; i < axis.Length; i++)
        {
            axis_s += $"{axis[i]} ";
            if (i % 30 == 0 && i != 0)
                axis_s += "\n";
        }
        string additional =
        $"Axis [{axis_s}]";
        return base.ToString(additional);
    }
```

```
}
```

Файл ControlPWMPacket.cs

```
using Mono.Cecil.Cil;
using System;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;
internal class ControlPWMPacket : Packet
{
    public ushort[] PWMs;
    public ControlPWMPacket() => PWMs = new ushort[Settings.PWMsCount];
    public ControlPWMPacket(bool encryption, ushort[] PWMs) : base
(encryption, (byte)PacketType.Controls.PWM)
    {
        if (PWMs == null || PWMs.Length != Settings.PWMsCount)
        {
            Debug.Log("PWM channels is bad!");
            return;
        }
        this.PWMs = PWMs;
        CalculateCRC2();
    }
    public ControlPWMPacket(ushort[] PWMs) : this(false, PWMs) { }
    public override void FromBytes(byte[] bytes)
    {
        byte[] packetBytes = ParseBytes(bytes);
        for (int i = 0, offset = 0; i < Settings.PWMsCount; i++, offset +=
2)
            PWMs[i] = BitConverter.ToUInt16(packetBytes, offset);
    }
    public override byte[] ToBytesBody()
    {
        List<byte> bytes = new List<byte>();
        for (int i = 0; i < PWMs.Length; i++)
            bytes.AddRange(BitConverter.GetBytes(PWMs[i]));
        return bytes.ToArray();
    }
    public override string ToString()
    {
        string pwms_s = "";
        for (int i = 0; i < PWMs.Length; i++)
        {
            pwms_s += $"{PWMs[i]} ";
            if (i % 30 == 0 && i !=0)
                pwms_s += "\n";
        }
        string additional =
        $"PWMs [{pwms_s}]";
        return base.ToString(additional);
    }
```

```
}
```

Файл DataPacket.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
using System.Text;
internal class DataPacket : Packet
{
    public ushort sequence;
    public string jsonData;
    public static int maxDataSize = Settings.maxPacketSize - 2 -
Header.headLengh - 2;
    public DataPacket() => jsonData = "";
    public DataPacket(bool encryption, byte type, ushort sequence, string
jsonData) : base(encryption, type)
    {
        if (jsonData == null || jsonData.Length == 0 || jsonData.Length >
maxDataSize)
        {
            Debug.Log("Bad JSON string");
            return;
        }
        this.sequence = sequence;
        this.jsonData = jsonData;
        CalculateCRC2();
    }
    public DataPacket(byte type, ushort sequence, string jsonData) :
this(false, type, sequence, jsonData) { }
    public override void FromBytes(byte[] bytes)
    {
        byte[] packetBytes = ParseBytes(bytes);
        sequence = BitConverter.ToUInt16(packetBytes, 0);
        jsonData = Encoding.UTF8.GetString(packetBytes, 2,
packetBytes.Length - 2);
    }
    public override byte[] ToBytesBody()
    {
        List<byte> bytes = new List<byte>();
        bytes.AddRange(BitConverter.GetBytes(sequence));
        bytes.AddRange(Encoding.UTF8.GetBytes(jsonData));
        return bytes.ToArray();
    }
    public override string ToString()
    {
        string additional =
        $"SQC[{sequence}] " +
        $"JSON DATA[{jsonData}]";
        return base.ToString(additional);
    }
}
```

Файл Header.cs

```
using System;
public class Header
{
    public byte[] marker { get => Settings.marker; }
    public bool encryption;
    public byte type;
    public byte CRC1;
    public static readonly ushort headLengh = 3 + 1 + 1 + 1;
    public Header() { }
    public Header(bool encryption, byte type)
    {
        this.encryption = encryption;
        this.type = type;
        CalculateCRC1();
    }
    public Header(byte type) : this(false, type) { }
    public Header(byte[] bytes)
    {
        FromBytes(bytes);
    }
    public void CalculateCRC1()
    {
        CRC1 = CRC.GetCRC1(ToBytes());
    }
    public byte[] ToBytes()
    {
        return new byte[] {
            marker[0], marker[1], marker[2],
            Convert.ToByte(encryption),
            type,
            CRC1
        };
    }
    public static Header GetHeader(byte[] bytes)
    {
        Header head = new Header();
        head.encryption = Convert.ToBoolean(bytes[3]);
        head.type = bytes[4];
        head.CRC1 = bytes[5];
        return head;
    }
    private void FromBytes(byte[] bytes)
    {
        encryption = Convert.ToBoolean(bytes[3]);
        type = bytes[4];
        CRC1 = bytes[5];
    }
    public override string ToString()
    {
```

```
        string str =
            $"Head[{marker[0]:X} {marker[1]:X} {marker[2]:X}]\t" +
            $"ENC[{encryption}]\t" +
            $"TYP[{type}]\t" +
            $"CRC1[{CRC1}]";
        return str;
    }
    public bool CheckCRC()
    {
        if (CRC1 == CRC.GetCRC1(ToBytes()))
            return true;
        else
            return false;
    }
}
```

Файл NetManager.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Net;
using System.Threading;
using UnityEngine;
public class NetManager : MonoBehaviour
{
    private void Start()
    {
        Networking net1 = new("PC 1", Settings.receivePort + 1);
        Networking net2 = new("PC 2");
        net1.Start();
        net2.Start();
        net1.Connect(IPAddress.Loopback, Settings.receivePort);
    }
    private void ProcessPWMControl(byte[] packetBytes)
    {
        ControlPWMPacket packet = new();
        packet.FromBytes(packetBytes);
        Debug.Log($"Control packet has been received, PWM 1 =
{packet.PWMs[0]}, PWM 2 {packet.PWMs[1]}");
    }
}
```

Файл Networking.cs

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Threading;
using System.Threading.Tasks;
using UnityEngine;
internal class Networking
```

```
{
    public delegate void Handler(byte[] packetBytes, IPEndPoint
remoteEndPoint);
    public delegate bool Checker();
    public enum ConnectionStatus
    {
        Connected,
        Disconnected,
        Connection
    }
    Dictionary<byte, Handler> serviceHandlers = new();
    Dictionary<byte, Handler> userHandlers = new ();
    IPEndPoint mainRemoteEndPoint;
    int receivePort;
    private ConnectionStatus status;
    public ConnectionStatus Status
    {
        private set
        {
            status = value;
            Debug.Log($"{name} status is {status}");
        }
        get { return status; }

    }
    public string name { get; private set; }
    public Networking(string name, int receivePort = Settings.receivePort)
    {
        this.name = name;
        this.receivePort = receivePort;
        SetServiceHandlers();
        status = ConnectionStatus.Disconnected;
    }
    public void Connect(IPAddress remoteIP, int remotePort =
Settings.receivePort)
    {
        if (status == ConnectionStatus.Disconnected)
        {
            mainRemoteEndPoint = new IPEndPoint(remoteIP, remotePort);
            RequestResponsePacket packet =
RequestResponsePacket.GetConnectionRequst();
            bool IsConnected()
            {
                if (status == ConnectionStatus.Connected)
                    return true;
                else return false;
            }
            status = ConnectionStatus.Connection;
            Send(packet, Settings.maxAttempts,
Settings.connectionAttemptTimeout, IsConnected);
        }
```

```
        else
        {
            Debug.Log($"Can't join in this status {status}");
        }
    }
    public void Connect(string IP, int remotePort = Settings.receivePort)
=> Connect(IPAddress.Parse(IP), remotePort);
    public void Disconnect()
    {
        if (status != ConnectionStatus.Disconnected)
        {
            mainRemoteEndPoint = null;
        }
    }
    public void Start()
    {
        Receive();
        Debug.Log($"{name} is started on
{IPAddress.Loopback}:{receivePort}");
    }
    public void Stop()
    {
        Disconnect();
    }
    public async void Send<T>(T packet, int attemptCount, int timeout,
Func<bool> checkEndAttempts) where T : Packet
    {
        UdpClient sender = new UdpClient();
        byte[] datagram = packet.ToBytes();
        for (int i = 0; i < attemptCount; i ++)
        {
            if (checkEndAttempts())
                break;
            try
            {
                DebugPanelController.PacketInfo = ($"HOST:{name} to
{mainRemoteEndPoint.Address}:{mainRemoteEndPoint.Port}\n{packet}");
                Debug.Log(packet);
                sender.Send(datagram, datagram.Length, mainRemoteEndPoint);
            }
            catch (Exception e)
            {
                Debug.Log($"Cant send packet! {e}");
            }
            await Task.Delay(timeout);
        }
    }
    public void Send<T>(T packet) where T: Packet
    {
        Send(packet, 1, 0, () => false);
    }
```

```
    public void Send<T>(T packet, int attemptCount, int timeout) where T :
Packet
    {
        Send(packet, attemptCount, timeout, () => false);
    }
    private async void Receive()
    {
        UdpClient receiver = new UdpClient(receivePort);
        IPEndPoint remoteEndPoint = null;
        await Task.Run(() =>
        {
            while (true)
            {
                try
                {
                    byte[] datagram = receiver.Receive(ref remoteEndPoint);
                    byte[] marker = { datagram[0], datagram[1], datagram[2]
};
                    if (CheckMarker(marker))
                    {
                        byte[] headerBytes = new byte[Header.headLengh];
                        Buffer.BlockCopy(datagram, 0, headerBytes, 0,
Header.headLengh);
                        Header header = new Header(headerBytes);
                        if (header.CRC1 == 0)
                        {
                            if (header.encryption)
                            {
                            }
                            ProcessReceivedPacket(datagram, header,
remoteEndPoint);
                        }
                        else
                        {
                            Debug.Log("Bad CRC-1");
                        }
                    }
                }
                catch (Exception e)
                {
                    Debug.Log(e);
                }
            }
        });
    }
    private bool CheckMarker(byte[] marker)
    {
        if (marker.Length == Settings.marker.Length &&
            marker[0] == Settings.marker[0] &&
            marker[1] == Settings.marker[1] &&
            marker[2] == Settings.marker[2])
```

```
        {
            return true;
        }
        else
        {
            Debug.Log("Unknow protocol marker");
            return false;
        }
    }

    private void ProcessReceivedPacket(byte[] packetBytes, Header header,
IPEndPoint remoteEndPoint)
    {
        byte type = header.type;
        if (type == PacketType.Response)
        {
            Debug.Log("RESP");
        }
        if (serviceHandlers.ContainsKey(type))
        {
            serviceHandlers[type].Invoke(packetBytes, remoteEndPoint);
        }
        else if (userHandlers.ContainsKey(type))
        {
            userHandlers[type].Invoke(packetBytes, remoteEndPoint);
        }
        else
        {
            Debug.Log($"No handlers for {type} type");
        }
    }
    public void SetHandlers(Dictionary<byte, Handler> handlers)
    {
        this.userHandlers.Clear();
        this.userHandlers = handlers;
    }
    public void SetHandler(Enum packetType, Handler handler)
    {
        this.userHandlers[Convert.ToByte(packetType)] = handler;
    }
    private void SetServiceHandlers()
    {
        serviceHandlers[PacketType.Request] = OnRequestHandler;
        serviceHandlers[PacketType.Response] = OnResponseHandler;
    }
    private void OnRequestHandler(byte[] packetBytes, IPEndPoint
remoteEndPoint)
    {
        RequestResponsePacket packet = new();
        packet.FromBytes(packetBytes);
        RequestsCode code = (RequestsCode)packet.code;
```

```
        switch(code)
        {
            case RequestsCode.Connection:
                OnConnectionRequestHandler(packet, remoteEndPoint);
                break;
            case RequestsCode.Ping:
                break;
        }
    }
    private void OnResponseHandler(byte[] packetBytes, IPEndPoint
remoteEndPoint)
    {
        RequestResponsePacket packet = new();
        packet.FromBytes(packetBytes);
        ResponsesCode code = (ResponsesCode)packet.code;
        switch (code)
        {
            case ResponsesCode.Connection:
                OnConnectionResponseHandler(packet, remoteEndPoint);
                break;
            case ResponsesCode.Pong:
                break;
            case ResponsesCode.OK:
                break;
        }
    }
    private void OnConnectionRequestHandler(RequestResponsePacket packet,
IPEndPoint remoteEndPoint)
    {
        ushort version = packet.data;
        if (version == Settings.protocolVersion)
        {
            RequestResponsePacket responsePacket = new
RequestResponsePacket(PacketType.Response,
                (byte)ResponsesCode.Connection, (ushort)(version +
Settings.magicConnectionAttempValue));
            mainRemoteEndPoint = new IPEndPoint(remoteEndPoint.Address,
remoteEndPoint.Port);
            status = ConnectionStatus.Connected;
            Send(responsePacket, Settings.maxAttempts,
Settings.connectionAttemptTimeout);
        }
        else
            Debug.Log($"Incorrect protocol version {version}");
    }
    private void OnConnectionResponseHandler(RequestResponsePacket packet,
IPEndPoint remoteEndPoint)
    {
        ushort version = packet.data;
        if (status != ConnectionStatus.Connected && mainRemoteEndPoint !=
remoteEndPoint)
```

```
        {
            if (version == Settings.protocolVersion +
Settings.magicConnectionAttempValue)
            {
                mainRemoteEndPoint = new IPEndPoint(remoteEndPoint.Address,
remoteEndPoint.Port);
                status = ConnectionStatus.Connected;
            }
            else
                Debug.Log($"Incorrect magic value in connection attempt
{version}");
        }
    }
}
```

Файл Packet.cs

```
using System;
using System.Collections.Generic;
abstract class Packet
{
    public Header header;
    public ushort CRC2;
    public static readonly int CRC2_lenght = 2;
    public Packet() { }
    public Packet(bool encryption, byte type)
    {
        header = new Header(encryption, type);
    }
    public Packet(byte type) : this(false, type) { }
    public void CalculateCRC2()
    {
        CRC2 = CRC.GetCRC2(ToBytesBody());
    }
    public byte[] ToBytes()
    {
        List<byte> bytes = new List<byte>();
        bytes.AddRange(header.ToBytes());
        bytes.AddRange(ToBytesBody());
        bytes.AddRange(BitConverter.GetBytes(CRC2));
        return bytes.ToArray();
    }
    public abstract byte[] ToBytesBody();
    public abstract void FromBytes(byte[] bytes);
    public byte[] ParseBytes(byte[] bytes)
    {
        byte[] headerBytes = new byte[Header.headLengh];
        Array.Copy(bytes,0,headerBytes,0,Header.headLengh);
        header = Header.GetHeader(headerBytes);
        int bodyBytesCount = bytes.Length - Header.headLengh -
Packet.CRC2_lenght;
        byte[] bodyBytes = new byte[bodyBytesCount];
```

```
            Array.Copy(bytes, Header.headLengh, bodyBytes, 0, bodyBytesCount);
            CRC2 = BitConverter.ToUInt16(bytes, bytes.Length - 2);
            return bodyBytes;
        }
        public override string ToString() => ToString("");
        public virtual string ToString(string additional)
        {
            if (additional == null || additional.Length == 0)
                additional = "No additional data";
            string str =
                $"\n\t\t\t-----PACKET STRUCTURE START-----\n" +
                header.ToString() +
                $"\n{additional}\t" +
                $"\tCRC2[{CRC2}]\n " +
                $"\t\t\t-----PACKET STRUCTURE END-----";
            return str;
        }
    }
```

Файл PacketType.cs

```
public enum RequestsCode
{
    Ping,
    Connection
}
public enum ResponsesCode
{
    Pong,
    OK,
    Connection
}
internal static class PacketType
{
    public const byte controlStartIndex = 1;
    public const byte streamStartIndex = 10;
    public static readonly byte Request = 20;
    public static readonly byte Response = 21;
    public const byte dataStartIndex = 22;
    public enum Controls
    {
        PWM = controlStartIndex,
        Axis
    }
    public enum Stream
    {
        Audio = streamStartIndex,
        Video
    }
    public enum Data
    {
        Telemetry = dataStartIndex,
```

```
        Settings
    }
}
```

Файл RequestResponcePacket.cs

```
using System;
using System.Collections.Generic;
using System.Security.Cryptography;
using Unity.VisualScripting;
internal class RequestResponsePacket : Packet
{
    public byte code;
    public ushort data;
    public RequestResponsePacket() { }
    public RequestResponsePacket(bool encryption, byte type, byte code,
ushort data) : base(encryption, type)
    {
        this.code = code;
        this.data = data;
        CalculateCRC2();
    }
    public RequestResponsePacket(byte type, byte code, ushort data) :
this(false, type, code, data) { }
    public override void FromBytes(byte[] bytes)
    {
        byte[] packetBytes = ParseBytes(bytes);
        code = packetBytes[0];
        data = BitConverter.ToUInt16(packetBytes, 1);
    }
    public override byte[] ToBytesBody()
    {
        List<byte> bytes = new List<byte>();
        bytes.Add(code);
        bytes.AddRange(BitConverter.GetBytes(data));
        return bytes.ToArray();
    }
    public override string ToString()
    {
        string additional =
        $"CODE[{code}] " +
        $"DATA[{data}]";
        return base.ToString(additional);
    }
    public static RequestResponsePacket GetConnectionRequst()
    {
        return new RequestResponsePacket(PacketType.Request,
(byte)RequestsCode.Connection, Settings.protocolVersion);
    }

}
```

Файл Settings.cs

```
static class Settings
{
    public static readonly byte[] marker = { 0x2d, 0xff, 0x2d };
    public static readonly byte protocolVersion = 1;
    public const int receivePort = 35553;
    public static readonly int PWMsCount = 12;
    public static readonly int axisCount = 3;
    public static readonly int maxPacketSize = 508;
    public static readonly int maxAttempts = 3;
    public static readonly int connectionAttemptTimeout = 1000;
    public static readonly ushort magicConnectionAttempValue = 22;
}
```

Файл StreamPacket.cs

```
using System;
using System.Collections.Generic;
using UnityEngine;
internal class StreamPacket : Packet
{
    public ushort sequence;
    public byte[] data;
    public static int maxDataSize = Settings.maxPacketSize - 2 -
Header.headLengh - 2;
    public StreamPacket() => data = new byte[1];
    public StreamPacket(bool encryption, byte type, ushort sequence, byte[]
data) : base(encryption, type)
    {
        if (data == null || data.Length == 0 || data.Length > maxDataSize)
        {
            Debug.Log("Bad JSON string");
            return;
        }
        this.sequence = sequence;
        this.data = data;
        CalculateCRC2();
    }
    public StreamPacket(byte type, ushort sequence, byte[] data) :
this(false, type, sequence, data) { }
    public override void FromBytes(byte[] bytes)
    {
        byte[] packetBytes = ParseBytes(bytes);
        sequence = BitConverter.ToUInt16(packetBytes, 0);

        data = new byte[packetBytes.Length - 2];
        Array.Copy(packetBytes, 2, data, 0, data.Length);
    }
    public override byte[] ToBytesBody()
    {
        List<byte> bytes = new List<byte>();
        bytes.AddRange(BitConverter.GetBytes(sequence));
        bytes.AddRange(data);
```

```
        return bytes.ToArray();
    }
    public override string ToString()
    {
        string bytes_s = "";
        for (int i = 0; i < data.Length; i++)
        {
            bytes_s += $"{data[i]:X} ";
            if (i % 30 == 0 && i != 0)
                bytes_s += "\n";
        }
        string additional =
        $"SQC[{sequence}] " +
        $"BIN DATA[{bytes_s}]";
        return base.ToString(additional);
    }
}
```

ДОДАТОК В

Міністерство освіти і науки України
Акціонерне товариство „Українська залізниця"
Транспортна академія України
Федерація залізничників України
Український державний університет залізничного транспорту

УкрДУЗТ – УЗ - ФЗУ
ІНФОРМАЦІЙНО-КЕРУЮЧІ СИСТЕМИ НА ЗАЛІЗНИЧНОМУ ТРАНСПОРТІ
2022 УКРАЇНА листопад

35-а міжнародна науково-практична конференція листопад 2022р. Україна

„Інформаційно-керуючі системи на залізничному транспорті"

Оргкомітет:
Голова:
Панченко С. В., д.т.н., ректор Українського державного університету залізничного транспорту

Члени оргкомітету: Бабаєв М. М., д.т.н. (Україна), Бунчуков О. А. (Україна), Бутько Т. В., д.т.н. (Україна), Гаврилюк В. І., д.ф-м.н. (Україна), Гончаренко В. І. (Україна), Доценко С. І ., д.т.н. (Україна), Жуковицький І. В., д.т.н. (Україна), Каргін А. О., д.т.н. (Україна), Климаш М. М., д.т.н. (Україна), Збігнев Лукасік, д.т.н. (Польща), Марек Мезитис, д.т.н. (Латвія), Мойсеєнко В. І ., д.т.н. (Україна), Приходько С. І., д.т.н. (Україна), Рубан І. В., д.т.н. (Україна), Самсонкін В. М., д.т.н. (Україна), Серков О. А., д.т.н. (Україна), Скалозуб В. В., д.т.н. (Україна), Терещенко Ю. М. (Україна), Трубчанінова К. А., д.т.н. (Україна), Тьєрі Хорсін (Франція), Шиш В. О., к.т.н. (Україна), Штомпель М. А., д.т.н. (Україна)

2022 р.
11 листопада

м. Харків,
Україна

ТЕЗИ СТЕНДОВИХ ДОПОВІДЕЙ ТА ВИСТУПІВ УЧАСНИКІВ КОНФЕРЕНЦІЇ

HIGHLIGHTS OF REPORTS AND PRESENTATIONS OF PARTICIPANTS TO THE CONFERENCE

СПИСОК АВТОРІВ

параметрів мережі під час навчання (на кожній ітерації) для того щоб повернутися до оптимальних параметрів у разі помилки;

- Інтерфейс створення топології мережі та налаштування параметрів її окремих структурних блоків.

- Програмна реалізація усіх необхідних для роботи та навчання мережі алгоритмів.

За основу для такого програмного забезпечення можна брати готовий фрейворк для глибокого машинного навчання або реалізувати усю логіку роботи мережі самостійно у рамках конкретного проєкту (якщо є необхідність). Наявність такого програмного забезпечення у відкритому доступі дозволить редагувати окремі структурні блоки, що забезпечує гнучкість.

Загальна ідея розробки такого програмного забезпечення полягає в тому, що графічний інтерфейс значно простіший для сприйняття і забезпечує кращі можливості з повторного використання, адже окремі етапи побудови та налаштування моделі мережі можна спростити (сховати за інтерфейс).

Таким чином, розроблене програмне забезпечення дозволяє зменшити час на побудову та вибір параметрів для навчання моделі мережі. Подальші дослідження спрямовані на вирішення задач машинного зору (класифікації, стискання графічних образів) в рамках даного програмного забезпечення.

Список використаних джерел

1 Carsten Steger. Machine Vision Algorithms and Applications / Carsten Steger, Markus Ulrich, Christian Wiedemann., 2018. – 516 с.

2 Guanghui Lan. First-order and Stochastic Optimization Methods for Machine Learning / Guanghui Lan., 2020. – 582 с.

Жученко О. С., к.т.н., доцент (УкрДУЗТ),
Карпук В. Ю., магістрант (Національний університет «Полтавська політехніка імені Юрія Кондратюка»)

УДК 621.39

РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ ТА ЙОГО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В УМОВАХ ЄВРОІНТЕГРАЦІЇ

У сучасному світі є популярним використання дистанційно-керованої техніки такої як: обладнання розумного будинку чи безпілотні літальні апарати (БПЛА). Для можливості їх роботи потрібні протоколи прикладного рівня, що зможуть описати правила обміну інформацією між ними та між людиною, яка є їх оператором. Наявність такої потреби підтверджується існуванням таких протоколів для дистанційно-керованої техніки як ModBus, UAVCan, UranusLink, MavLink, MQTT, CoAP тощо [1]. Однак, ці протоколи мають недоліки, які за певних умов можуть перешкоджати їх безпосередньому застосуванню, наприклад ModBus, UranusLink, MavLink передають дані у відритому форматі без можливості шифрування, MQTT та CoAP не мають можливості потокової передачі відео, MavLink маючи багатий функціонал є доволі складним для його імплементації розробником.

Альтернативним рішенням є розробка та реалізація власного протоколу для телеметрії та дистанційного управління, який може передавати потокові аудіо та відео дані, легкість реалізації для розробника та надає додаткову можливість шифрування трафіку.

Протокол є прикладним та призначений для обміну даними між керуючим оператором та дистанційно-керованою технікою. Цей протокол може використовуватися для віддаленого управління, моніторингу чи налаштування різної техніки.

Протокол працює поверх транспортного рівня тому він займає місце 7-го рівня у мережевій моделі OSI або 4-й прикладний рівень у моделі TCP/IP [2]. За рахунок цього він може бути переданий по будь-якому середовищу передачі.

Однією із головних особливостей є легкість його реалізації та використання для розробника. Зазначимо основні особливості протоколу телеметрії:

- протокол може передавати різні типи даних: команди керування, телеметричні дані, аудіо і відео потоки та спеціальні службові повідомлення;

- в залежності від типу даних, що передаються, протокол має різні механізми гарантії їх доставки та перевірки цілісності;

- у протоколі є необов'язкова можливість шифрування трафіку;

- більшість типів повідомлень мають представлення у двійковому вигляді, але частина має текстовий формат. Такий підхід забезпечує гнучкість розробки та невелике використання надлишкових даних;

- протокол може визначати пріоритет пакетам передачі. Такі пакети будуть оброблятися швидше.

Для розробки протоколу телеметрії створено декілька можливих типів повідомлень (рис. 1), керуючись загальноприйнятими рекомендаціями розробників протоколів [3]. Особливу увагу приділялося зменшенню розміру пакету із збереженням функціональності та гнучкості розробки для подальшої можливої модернізації.

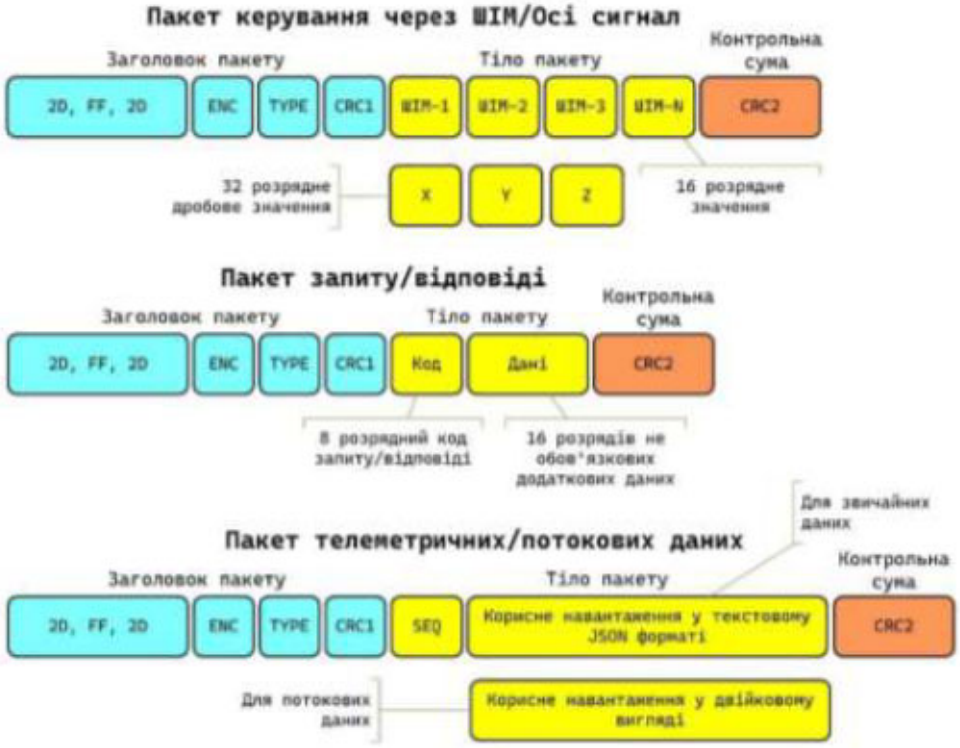


Рис. 1. Розроблені структури пакетів повідомлень протоколу телеметрії

Таким чином розроблена система повідомлень для протоколу телеметрії, яка дозволяє:

- в залежності від типу повідомлення передавати дані у текстовому чи двійкову форматі;
- контролювати послідовність даних для забезпечення їх цілісності;
- гарантувати доставлення важливих даних;
- виявляти помилки при передачі;
- шифрувати корисне навантаження.

Напрямком подальшої роботи є дослідження роботи протоколу телеметрії в середовищах симуляції наприклад jMAVSim чи GazeboSim. Таким чином отримані дані результатів тестування дозволять провести оновлення протоколу телеметрії для управління реальними БПЛА чи іншою дистанційно-керованою технікою.

Список використаних джерел

1. Comparative study of IoT protocols / Sakina Elhadi, Abdelaziz Marzak, Nawal Sael, Soukaina Merzouk, 2018. – 5 с.
2. Book cover Book cover Guide to OSI and TCP/Mohammed M. Alani. Guide to OSI and TCP/IP Models / Mohammed M. Alani., 2014.
3. Rescorla E. Writing Protocol Models RFC 4101 - IETF/ Rescorla E., 2005. – 22 с.

Гаєвський В. В., к.т.н.
(ТОВ «НВП «Залізничавтоматика», Харків)

УДОСКОНАЛЕННЯ МЕТОДОЛОГІЇ РОЗРОБКИ СИСТЕМ КЕРУВАННЯ РУХОМ ПОЇЗДІВ

Індустріальний світ все твердіше стає на шлях підтримки Індустрії 4.0. Але ще багато галузей промисловості застрягли в нейтральному стані. Нажаль це не оминуло і залізничний транспорт.

Промисловий Інтернет речей (IIoT), штучний інтелект (AI) та машинне навчання (ML), Digital Twins, Big Data предиктивна аналітика та обслуговування є основними інструментами щодо впровадження Технічного Обслуговування 4.0 (ТО4.0) яке має змінити сам підхід до експлуатації як технічних засобів залізничного транспорту так і промисловості в цілому. Це дозволить максимізувати час безперебійної роботи систем керування за рахунок мінімізації їх незапланованого та реактивного обслуговування.

Більшість систем керування рухом поїздів (СКПР) що експлуатуються на залізницях загального та незагального користування, муніципального рейкового транспорту засновані на функціонуванні, що не залежить від процесу технічного обслуговування (ТО). Системи керування не знаходяться у діалоговому

ДОДАТОК Г

Демонстраційний матеріал

Національний університет «Полтавська політехніка імені Юрія Кондратюка»
Навчально-науковий інститут інформаційних технологій і робототехніки
Кафедра автоматики, електроніки та телекомунікацій

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРАНТА

НА ТЕМУ РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ ТА ЙОГО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В УМОВАХ ЄВРОІНТЕГРАЦІЇ

Виконав:
студент групи 601-ТТ Карпук В.Ю.
Керівник:
канд. техн. наук, доцент Жученко О.С.

Полтава 2022

ТЕМА: РОЗРОБКА ПРОТОКОЛУ ТЕЛЕМЕТРІЇ ДЛЯ ДИСТАНЦІЙНО-КЕРОВАНОЇ ТЕХНІКИ ТА ЙОГО ПРОГРАМНОЇ РЕАЛІЗАЦІЇ В УМОВАХ ЄВРОІНТЕГРАЦІЇ

Мета: розробка протоколу телеметрії для дистанційно керованої техніки та його програмна реалізація.

Об'єкт дослідження: протокол прикладного рівню для телеметрії та дистанційного керування технікою.

Предмет дослідження: основні принципи та методи розроблення прикладних протоколів телеметрії.

Задачі даної кваліфікаційної роботи:

- аналіз внутрішньої будови сучасних протоколів телеметрії;
- виведення методів та принципів розробки прикладних протоколів передачі даних;
- вибір інструментів для розробки протоколу;
- розробка специфікації для протоколу телеметрії, опис його внутрішньої будови;
- розробка моделі тестування та дослідження роботи протоколу.

ОСНОВНА ТЕОРІЯ

- **Дистанційне керування** – це передача сигналів керування віддаленому об'єкту.
- **Телеметричні дані** – це інформація про стан віддаленого об'єкту.
- **Протокол телеметрії** – це опис правил обміну даними між віддаленими об'єктами.

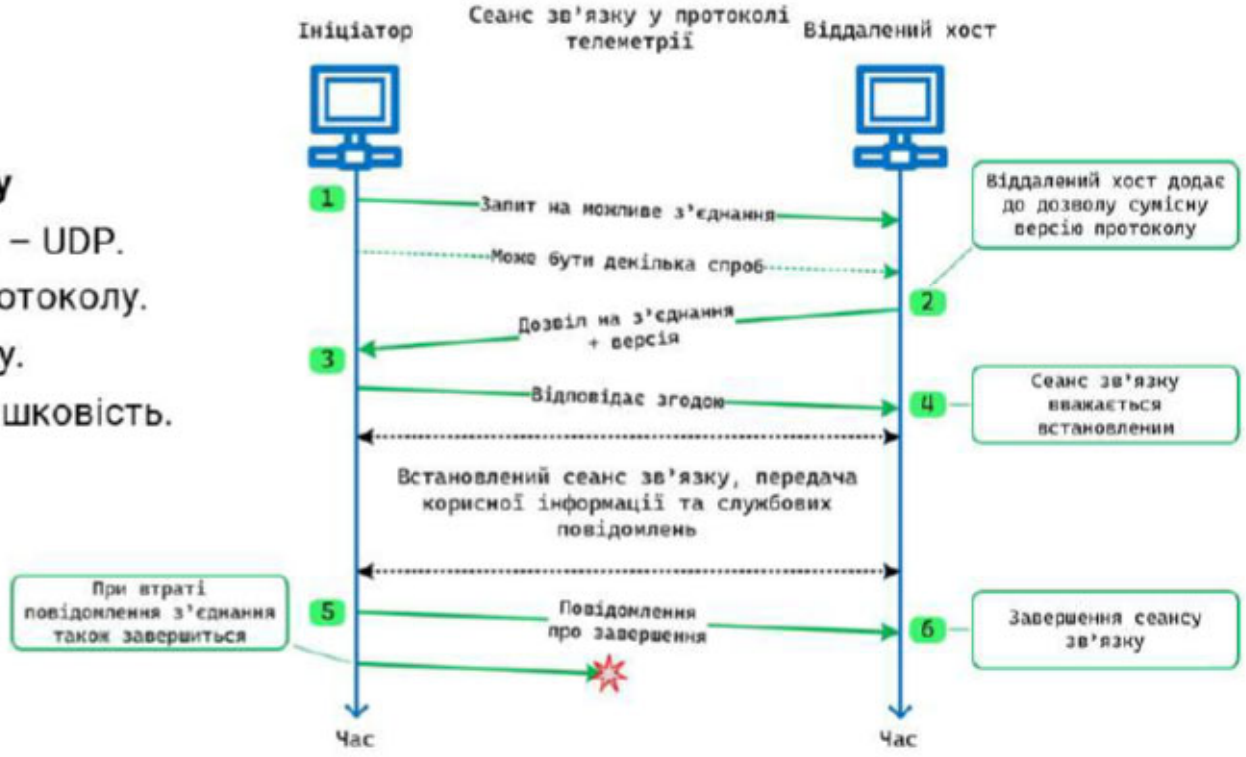
ВНУТРІШНЯ БУДОВА ПАКЕТУ ПРОТОКОЛУ ТЕЛЕМЕТРІЇ

1. **Заголовок –** ідентифікація, службові дані.
2. **Блок корисних даних або повідомлення.**
3. **Кінцеві дані –** контрольна сума.

ДІАГРАМИ СЕАНСУ ЗВ'ЯЗКУ

Особливості сеансу зв'язку

- Транспортний протокол – UDP.
- Перевірка сумісності протоколу.
- Моніторинг стану зв'язку.
- Порівняно менша надлишковість.



ДІАГРАМИ СТАНІВ ПЕРДАЧІ ДАННИХ

Два види керування:

- Осі – XYZ, 32 бітне дробове значення.
- ШІМ сигнали – 16 бітне ціле значення.

Передача ШІМ сигналів керування

Тип запит/відповідь:

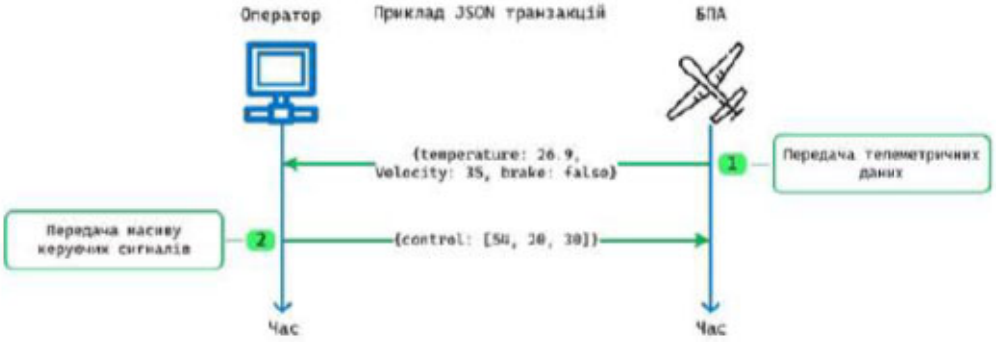
- Передача короткого повідомлення – код + необов'язкові дані.
- Необов'язкова можлива відповідь.

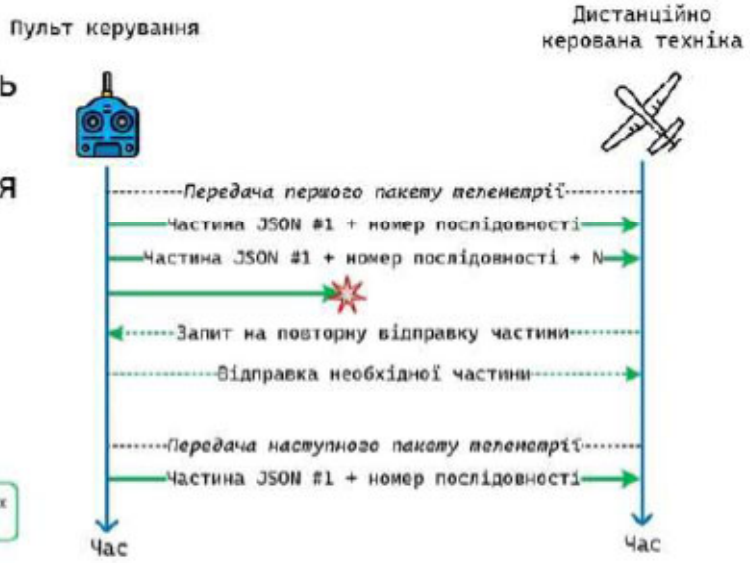
Передача типу «Запит/Відповідь»

ДІАГРАМИ СТАНІВ ПЕРДАЧІ ТЕЛЕМЕТРИЧНИХ ДАНИХ

Особливості передачі телеметрії:

- Потокові дані – двійковий формат, не мають гарантії доставки.
- Телеметричні дані – текстовий формат, є гарантія доставки.
- За необхідності дані можуть бути сегментовані.



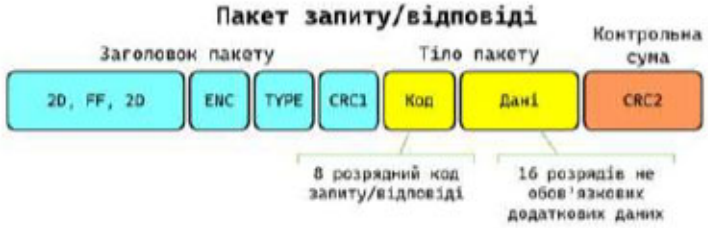


Передача довільних телеметричних даних

ДЕТАЛЬНА СТРУКТУРА ТИПІВ ПАКЕТІВ ПРОТОКОЛУ ТЕЛЕМЕТРІЇ

Типи пакетів:

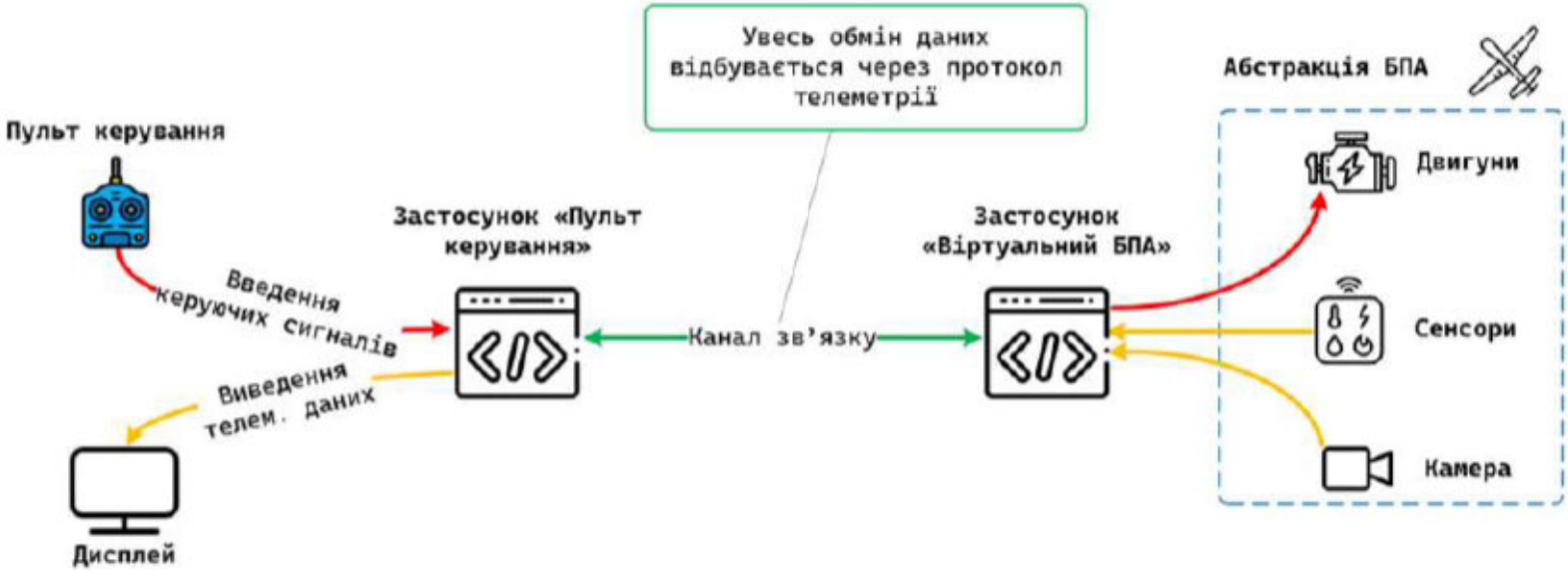
- Довільні дані (телеметрія).
- Керування.
- Запит/відповідь.
- Потокові дані (аудіо/відео).



МОДЕЛЬ НАЛАГОДЖУВАННЯ ПРОТОКОЛУ

Переваги моделі налагоджування для розробника:

- Можливість швидкого тестування.
- Немає ризику для обладнання.
- Без обмежень реальної техніки.
- Гнучкість різноманітних сценаріїв для тестування.

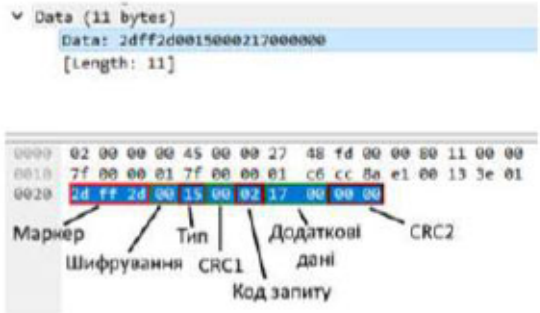


ГРАФІЧНИЙ ІНТЕРФЕЙС МОДЕЛІ НАЛАГОДЖУВАННЯ – РОЗГЛЯД ПАКЕТІВ

Вікно розгляду пакетів дозволяє:

- Отримати список пакетів відповідно до часу.
- Розгорнути будь-який пакет для розгляду.

Інтерфейс вікна розгляду пакетів

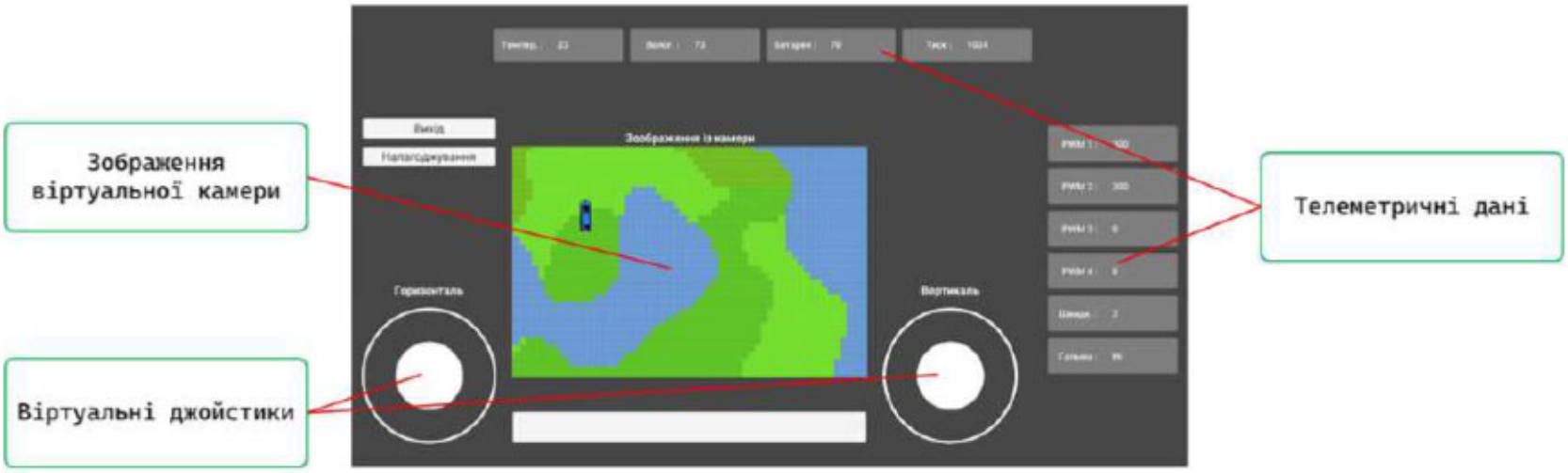


Перехоплений пакет запиту

ГРАФІЧНИЙ ІНТЕРФЕЙС МОДЕЛІ НАЛАГОДЖУВАННЯ – ТЕРМІНАЛ

Термінал керування дозволяє:

- Керувати віртуальною технікою на мапі.
- Отримувати відеозображення та телеметричні показники.



Віртуальний термінал для керування

ЗАГАЛЬНІ ВИСНОВКИ

У ході виконання кваліфікаційної роботи було виконано наступні завдання:

- проведено аналіз сучасних протоколів для керування дистанційною технікою та передачі телеметричних даних;
- обрано інструменти для розробки протоколу – мова програмування C#, мультимедійна платформа Unity, мережевий аналізатор Wireshark;
- розроблено специфікацію протоколу із описом його внутрішньої структури та принципу роботи;
- Проведено дослідження роботи та тестування працездатності протоколу телеметрії у моделі для налагоджування.