УДК 629.33 + 681.3.01

https://doi.org/10.15407/fmmit2026.42.119

Розроблення та верифікація програмно-апаратного середовища для виявлення та візуалізації помилок у мережах CAN

Павло Дзелендзяк[1], Ростислав Наконечний[2]

[1]Магістр, Національний Університет «Львівська Політехніка», 79013, Львів, вул. Степана Бандери 12, e-mail: pavlo.dzelendziak.mknuo.2025@lpnu.ua

[2]к.т.н., доцент, Національний Університет «Львівська Політехніка», 79013, Львів, вул. Степана Бандери 12, e-mail: rostyk.ws@gmail.com

У статті розглядається розробленняпрограмного середовища для симуляції, виявлення та візуалізації помилок у мережах CAN (Controller Area Network) на базі фреймворку Flutter. Запропонований підхід базується на впровадженні розширеної об'єктної моделі даних з вбудованими мітками достовірності (Ground Truth), що дозволяє поєднати контрольовану ін'єкцію п'яти типів аномалій з автоматизованим розрахунком метрик достовірності детекції. Розроблена система реалізує п'ять сценаріїв ін'єкції аномалій: побітові мутації, пакетні помилки, ін'єкцію статичних паттернів, порушення порядку байтів та CRC-фолти. Для оцінки ефективності алгоритмів детекції використовується матриця невідповідностей (Confusion Matrix) з розрахунком показників Precision, Recall та F1-Score в режимі реального часу. Локальне збереження даних реалізовано засобами SQLite, а архітектура системи базується на принципі розмежування логіки обробки та користувацького інтерфейсу з використанням асинхронних механізмів Dart. Практичне тестування на вибірці з 65 кадрів підтвердило працездатність системи та виявило напрямки для подальшого вдосконалення алгоритмів детекції. Створений інструментарій може бути використаний для бенчмаркінгу нових методів виявлення вторгнень у системах кібербезпеки сучасного транспорту.

***Ключові слова**: шина CAN, ідентифікація помилок, циклічний надлишковий код (CRC), ін'єкція помилок, кросплатформенна розробка, реальний час (real-time), мобільний моніторинг, аналіз мережевого трафіку, архітектура ПЗ, обробка потокових даних, фрейм.*

Вступ. Сучасні електронні системи, зокрема автомобільні та промислові, неможливо уявити без ефективного та надійного обміну інформацією між різними вузлами та контролерами. Одним із найпоширеніших стандартів для організації такої комунікації автомобільних систем є протокол CAN (Controller Area Network) [1, 2]. Основна мета даного протоколу полягає у забезпеченні швидкого, безпечного та децентралізованого обміну даними між мікроконтролерами, сенсорами та виконавчими механізмами без необхідності наявності центрального комп’ютера.

Незважаючи на високу надійність базового протоколу, передача даних у реальних умовах завжди піддається впливу різноманітних перешкод: електромагнітних завад, апаратних збоїв, механічних пошкоджень кабелів або помилок програмного забезпечення. У зв'язку з цим виявлення та сигналізація помилок стає критично важливою функцією для забезпечення безпечної та стабільної роботи мережі. Відомо, що сучасні системи CAN оснащені базовими механізмами контролю, такими як CRC, контролю бітів та підтвердження прийому, які дозволяють виявляти більшість типових помилок [1].

Мета даної роботи полягає у дослідженні програмних методів виявлення та сигналізації помилок у протоколі CAN, аналізі існуючих підходів та розробці програмних рішень, які дозволяють підвищити надійність системи та ефективність обробки помилок. Основні завдання даної роботи передбачають аналіз сучасних програмних підходів та алгоритмів, оцінку їх переваг та недоліків, розроблення структурної моделі програми для обробки помилок і демонстрацію її роботи у реальному середовищі.

Програмний підхід до вирішення цієї задачі дозволяє гнучко адаптувати систему під конкретні потреби, забезпечує можливість модернізації алгоритмів виявлення помилок та зменшує залежність від апаратних засобів, що робить його ефективним та економічно доцільним у сучасних умовах.

1. Огляд предметної області

Відомо, що мережний протокол обміну інформацією Controller Area Network (CAN) є стандартом промислових мереж, орієнтований на обмін даними у реальному часі між інтелектуальними пристроями. Розроблений компанією Robert Bosch GmbH, він став де-факто стандартом у сучасній електроніці та системах промислової автоматизації завдяки високій надійності, завадостійкості та ефективності механізму пріоритетності задач [1, 3].

Функціонування протоколу CAN базується на постійному «прослуховуванні» шини. Перед відправкою кожен пристрій, за допомогою спілкування, перевіряє чи вільна лінія. Якщо вона зайнята, пристрій чекає її звільнення. У випадку, коли декілька пристроїв одночасно відправили сигнал, спрацьовує механізм арбітражу, який визначає пріоритетність. Перевагу отримують кадри з меншим числовим ідентифікатором.

Стандартний кадр CAN 2.0A складається з наступних полів:

- Identifier (ID) – визначає пріоритет повідомлення та його тип (11 або 29 біт).
- Data Length Code (DLC) – вказує на кількість байтів даних (від 0 до 8).
- Data Field – безпосередньо корисне навантаження.
- CRC (Cyclic Redundancy Check) – 15-бітна контрольна сума для перевірки цілісності даних.

- ACK (Acknowledgement) - підтвердження отримання кадру вузлами [1, 2].

Однією з головних переваг CAN є вбудована багаторівнева система виявлення помилок. Протокол визначає п’ять основних типів помилок, які можуть виникати під час передачі [1]:

- Bit Error – виникає, коли вузол-передавач порівнює власне надіслане значення біта з тим, що реально спостерігається на шині. Якщо значення відрізняються, фіксується помилка;
- CRC Error – вузол-приймач самостійно обчислює контрольну суму отриманого повідомлення. Якщо обчислений результат не збігається з кодом у полі CRC самого кадру, дані вважаються пошкодженими;
- Form Error – помилка формату даних, яка фіксується при порушенні структури кадру;
- ACK Error – виникає, якщо передавач не отримує домінантний сигнал у полі ACK, що означає, що жоден вузол у мережі не прийняв повідомлення коректно.
- Stuff Error – виникає при порушенні правила бітового стафінгу: після п'яти послідовних однакових біт передавач зобов'язаний вставити інверсний біт. Якщо цього не відбулось, приймаючий вузол фіксує помилку формування кадру.

Задля забезпечення стабільності роботи системи використовується механізм “завершення помилки”. При цьому робота залежить від інформації, записаної у лічильник помилок, який має кожен вузол. Залежно від значень лічильника є три стани:

1. Error Active – нормальний режим роботи;
2. Error Passive – вузол може брати участь у комунікації, але не має права переривати роботу мережі при виявленні помилок;
3. Bus Off – вузол автоматично відключається від шини при досягненні критичної кількості помилок (255) [1].

2. Аналіз існуючих рішень

Функціонування мереж, що використовують протокол передачі інформації CAN вимагає спеціалізованого набору інструментів, які забезпечують взаємодію з шиною: від підтвердження кадрів до комплексної симуляції даних. Основним завданням такого програмного забезпечення є: отримання пакетів, дешифрація даних пакетів та їх візуалізація для покрокового аналізу. При цьому ринок програмних рішень ділиться на дві категорії: безкоштовні та комерційні.

Комерційні рішення, наприклад, продукти компаній Kvaser або Vector, основною перевагою яких є наявність широкого вибору інструментів для взаємодії, а також наявність сертифікованих методів аналізу для протоколів

вищих рівнів (CAN open, або J1939) [4]. Натомість, закритий доступ до вихідного коду та висока вартість ліцензій обмежують її використання у проектах.

Безкоштовні рішення, зазвичай, базуються на використанні системних інструментів, серед яких Socket CAN один з найпопулярніших інструментів. Він інтегрований прямо у ядро Linux OS, а його архітектурна перевага полягає у використанні моделі Berkeley Sockets [5]. Вона дозволяє операційній системі взаємодіяти з CAN-інтерфейсом як з звичайним пристроєм. Такий підхід дозволив вирішити проблему доступності до шини: декілька незалежних один від одного додатків можуть зчитувати дані з одного приладу, саме це робить розробку програмних продуктів легшою та доступнішою [5].

Прикладом реалізації такого підходу є набір інструментів python-can [6]. Завдяки високому рівню абстракції він став стандартом для прототипування. Такий набір інструментів дозволяє оперувати повідомленнями, як об'єктами класів, ігноруючи специфіку роботи USB-to-CAN адаптерів, або PCI-карт [6]. Проте відкриті рішення вимагають розроблення унікального інтерфейсу, мета якого візуалізувати моніторинг трафіку, або аномалій.

На основі проведених досліджень, з метою усунення низки вказаних недоліків та забезпечення безпосереднього моніторингу і гнучкості контролю, в роботі запропонована система генерації та візуалізації роботи CAN-мережі. Розроблювана система є проміжною ланкою між обмеженими комерційними продуктами, та маломасштабованими консольними утилітами. При цьому створений додаток дозволяє реалізувати автоматичний фоновий аналіз кадрів та одночасний вивід інформації про помилки на графік. Вибір фреймворку Flutter, завдяки оптимізованій системі, вирішує задачу з повільною роботою програмного продукту, яка зустрічається в аналогах на Python при великих потоках даних.

3. Моделювання функціональних сценаріїв взаємодії

В роботі запропоновано і розроблено UML структуру (Рис.1), яка передбачає чотири основні напрями роботи програми, а саме: управління доступом, налаштування середовища, симуляцію та аналітичну обробку.

Кожен зі зазначених напрямів відповідає за конкретний етап взаємодії користувача з програмним продуктом. У межах управління доступом описано процедуру авторизації користувача. Це дозволяє зберігати дані про сесії симуляції, відображення давніших сесій та їх уніфікацію.

На етапі налаштування середовища, користувач налаштовує візуальні елементи програми, фільтрацію вхідних даних.

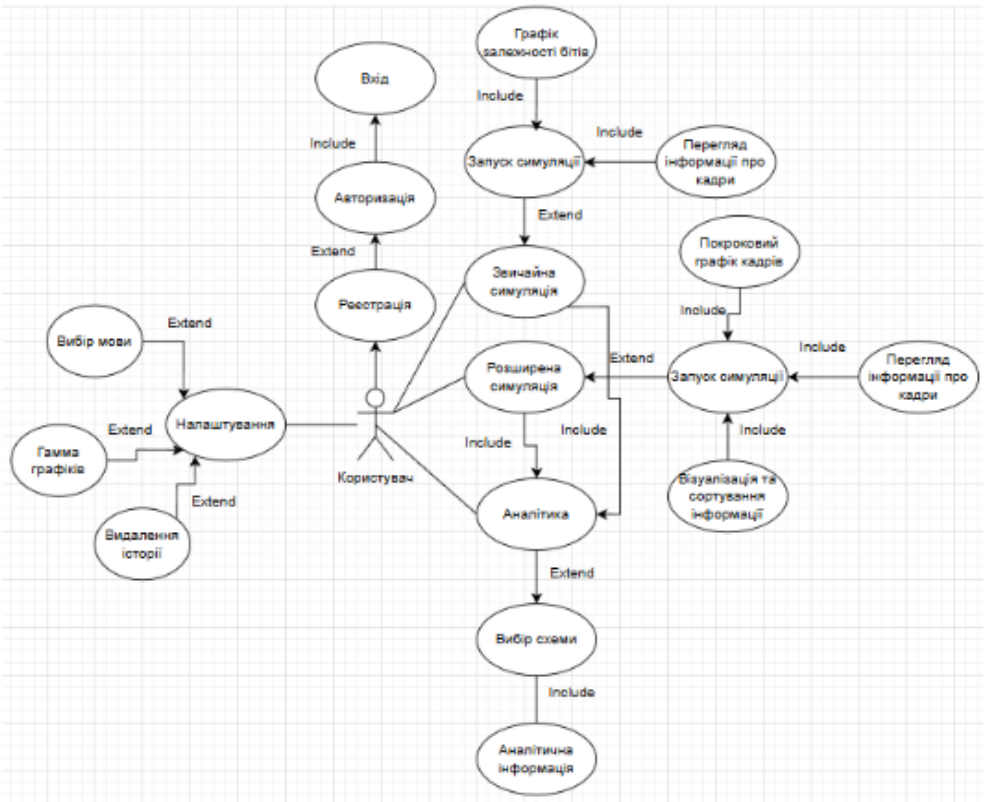

Рис. 1. UML структура роботи програми

На симуляційному етапі авторизований користувач отримує доступ до симуляції потоку CAN-кадрів, опису помилок, можливості відстеження відношення помилкових кадрів, до тих, які пройшли перевірку.

Аналітична обробка – завершує процес, забезпечуючи візуалізацію отриманих результатів у реальному часі, надаючи інструменти для пошуку конкретних помилок на певному відрізку часу.

Такий розподіл функцій дозволяє відокремити процес генерації графіку від його аналізу, що робить систему гнучкою в процесі експлуатації.

4. Обгрунтування вибору технологічного стеку та програмних засобів

Для реалізації програмного рішення обрано платформу Flutter та мову Dart [7, 8]. Такий вибір обумовлений тим, що Flutter місить у собі графічний рушій flame і дозволяє забезпечувати швидке та повільне відображення графіків аналізу [7]. Крім того додатковою перевагою є використання механізму компілятора перед виконанням, що значно збільшує швидкодію роботи програмного продукту. Такий підхід мінімізує затримки у відображені даних, важливих для

протоколу CAN, у якому критично важливим є досягнення мінімального часу відгуку.

В основі системи закладено принцип чіткого розмежування функцій: обрахунки та обробка даних відокремлені від користувацького інтерфейсу. Головна логіка полягає в ізоляції середовища, мета якого – обчислення, формування повідомлень та фільтрація помилкових кадрів. Завдяки механізмам асинхронності, обробка інформації про кадри відбувається паралельно з роботою інтерфейсу, що дозволяє користувачу спостерігати повільні графіки та швидку реакцію на аномалії, навіть якщо мережа максимально завантажена [8].

Локальне збереження даних реалізовано за допомогою використання системи управління базами даних SQLite [9]. Вона забезпечує швидкий запис та читання даних і не потребує інтернет з'єднання. Для експорту звітів та їх читання використано механізм серіалізації JSON. Таке поєднання інструментів забезпечує високу масштабованість та швидкість роботи системи.

5. Програмна реалізація

В основу програмної реалізації закладено об'єктно-орієнтовану модель даних, яка реалізована на основі CanFrame. Дана модель повністю відповідає специфікації ISO 11898-1 [1], що дозволяє оперувати як стандартними, так розширеними ідентифікаторами кадрів. Структура класу містить в собі ключові параметри протоколу, а саме: ідентифікатор, код довжини даних, корисне навантаження, поле циклічності надлишкового коду (CRC-15). В роботі використано механізм тверджень (assertions) на рівні конструктора, що гарантує цілісність об'єкту на етапі створення, виключаючи можливість появи некоректних значень.

Базуючись на специфікації ISO 11898 розроблена модель даних кадру, яка реалізована на основі класу CanFrame. Структура містить в собі усі властивості кадру CAN. Таким чином, саме це дозволило вийти за межі простих ідентифікаторів і, натомість, використовувати стандартну 11-бітну версію та розширену 29-бітну.

Лістинг 1. Програмна реалізація структури CAN – кадру з механізмами валідації.

```
final int id;

final int dlc;

final List<int> data;

final int crc;

final DateTime timestamp;

final bool isCorrupted;
```

```
      final List<dynamic> errors;
      CanFrame({
      required this.id,
      required this.dlc,
       required this.data,
      required this.crc,
      required this.timestamp,
      this.isCorrupted = false,
      this.errors = const [],
      }) : assert(dlc >= 0 && dlc <= 8, 'DLC must be between 0
and 8'),
      assert(data.length <= 8, 'Data length cannot exceed 8
bytes'),
      assert(data.every((byte) => byte >= 0 && byte <= 255),
'Data bytes must be 0-255');
      }
```

5.1. Автоматизація обчислення метрик достовірності. Наявність вбудованих міток дозволяє системі в режимі реального часу розраховувати матрицю невідповідностей (Confusion Matrix). Порівнюючи стан isCorrupted моделі з вихідним вердиктом алгоритму, система автоматично обчислює ключові показники:

Detection Rate (True Positive Rate): здатність алгоритму ідентифікувати реальну атаку.

False Discovery Rate: частоту хибних спрацювань на легітимному трафіку.

Лістинг 2. Диспетчер вибору типу мутації – рівномірний вибір з 5 сценаріїв ін'єкції аномалій

```
_BitFaultResult? _generateBitFault({
  required int id,
  required int dlc,
  required List<int> data,
  required bool isExtended,
  Random? random,
}) {
```

```
        final rnd = random ?? Random();
        final candidates = <_BitFaultType>[];

        if (dlc > 0) {
          candidates.add(_BitFaultType.dataBitFlip);
          candidates.add(_BitFaultType.patternInjection);
        }
        if (dlc > 1) {
          candidates.add(_BitFaultType.sequenceAnomaly);
          candidates.add(_BitFaultType.burstError);
        }
        candidates.add(_BitFaultType.identifierBitFlip);

        final selected = candidates[rnd.nextInt(candidates.length)];
        switch (selected) {
          case _BitFaultType.dataBitFlip:
            return _applyDataBitFlip(rnd, dlc, data);
          case _BitFaultType.burstError:
            return _applyBurstError(rnd, dlc, data);
          case _BitFaultType.identifierBitFlip:
            return _applyIdBitFlip(rnd, id, isExtended);
          case _BitFaultType.patternInjection:
            return _applyPatternInjection(rnd, dlc);
          case _BitFaultType.sequenceAnomaly:
            return _applySequenceAnomaly(rnd, dlc, data);
        }
}
```

Розроблене середовище підтримує п’ять сценаріїв ін’єкції аномалій:

1. Bit-level mutations – одиночні інверсії бітів у полі ідентифікатора або даних.
2. Burst errors – послідовні спотворення групи бітів (від 2 до 6), що імітують електромагнітні завади.
3. Pattern injection – заповнення кадру статичними або циклічними паттернами (наприклад, 0xFF або 0xAA/55).
4. Sequence anomaly – порушення логічного порядку байтів у корисномунавантаженні.

5. CRC faults – навмисне спотворення даних при збереженні старого CRC дляперевірки здатності алгоритму виявляти аномалії, які «пропускає» стандартне залізо.

5.2. Ізоляція процесів генерації та аналізу. Використання об’єкту Stopwatch дозволяє фіксувати затримку обробки (processing latency) з точністю до мікросекунд, що дозволяє встановити кореляцію між складністю алгоритму детекції та його здатністю працювати у високошвидкісних CAN-мережах без накопичення черги повідомлень.

Лістинг 3. Вимірювання затримки детекції за допомогою stopwatch

```
final stopwatch = Stopwatch()..start();
final results = checker.analyzeFrame(frame);
stopwatch.stop();
print('Час аналізу: ${stopwatch.elapsedMicroseconds} мкс');
return results;
```

6. Візуалізація та аналіз результатів

Графічний інтерфейс додатку, розроблений на основі фреймворку Flutter, виступає не лише як засіб відображення трафіку, а й як інтерактивна аналітична модель. При цьому основний акцент зроблено на динамічному порівнянні еталонних станів кадру з результатами програмних детекторів.

Візуалізація реалізована через реактивний список кадрів, де кожен об’єкт класу CAN frame підсвічує відповідно своєму статусу. Кадри, марковані прапорцем is Corrupted, виділяються колірною індикацією, що дозволяє оцінити інтенсивність аномалій у відношенні до кадрів без помилок. При виборі конкретного кадру користувач отримує доступ до метаданих у полі errors, що розкриває деталі кадру (наприклад помилку цілісності).

Рис. 2. Головне вікно моніторингу та індикація аномальних кадрів.

На Рис. 2 наведено роботу реактивного списку кадрів. Завдяки об’єктній моделі CanFrame, система миттєво ідентифікує кадри з встановленим прапорцем is Corrupted. Користувач може розгорнути деталі будь-якого кадру, щоб побачити метадані з поля errors, де вказано конкретний тип внесеної мутації (наприклад, Bit Flip або Sequence Anomaly).

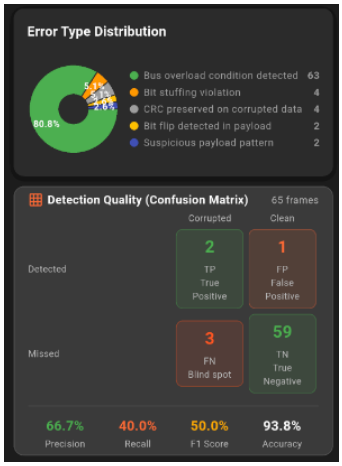

Рис. 3. Аналітичні графіки точності та продуктивності (Error Type Distribution, Detection Quality)

Для об’єктивної оцінки ефективності алгоритмів у системі реалізовано два типи динамічних графіків (рис. 3).

Error Type Distribution: відображає розподіл виявлених аномалій за типами. У тестовій сесії з 65 кадрів домінуючим типом стали перевантаження шини – Bus overload condition detected (63 випадки, 80.8%), що відповідає симульованому навантаженню. Також зафіксовано: Bit stuffing violation (4), CRC preserved on corrupted data (4), Bit flip detected in payload (2) та Suspicious payload pattern (2). Наявність усіх п'яти типів підтверджує коректну роботу механізмів ін'єкції аномалій.

Detection Quality (Confusion Matrix): забезпечує кількісну оцінку алгоритму детекції. При тестуванні на65 кадрах отримано: TP = 2, FP = 1, FN = 3, TN = 59. Точність (Precision) склала 66.7%, повнота (Recall) – 40.0%, F1-Score – 50.0%, загальна точність (Accuracy) – 93.8%. Значення FN = 3 вказує на наявність «сліпих зон» алгоритму, що є напрямком для подальшого вдосконалення детекторів.

Висновки. У межах даного дослідження розроблено та верифіковано програмно-технічне середовище для аналізу цілісності трафіку в мережах CAN на базі сучасного фреймворку Flutter. Проведений аналіз існуючих рішень показав, що комерційні інструменти (Kvaser, Vector) є надмірно закритими, а консольні утиліти на Python – маломасштабованими при високих потоках даних. Запропонована система займає проміжну нішу, поєднуючи доступність відкритих рішень із функціональністю промислових інструментів.

Розроблена об’єктна модель даних на основі класу CanFrame повністю відповідає специфікації ISO 11898-1:2024 та підтримує як стандартні 11-бітні, так і розширені 29-бітні ідентифікатори. Реалізовано п’ять сценаріїв ін’єкції аномалій – побітові мутації, пакетні помилки, ін’єкцію паттернів, порушення

порядку байтів та CRC-фолти – що охоплює весь спектр типових несправностей CAN-мереж.

Практичне тестування на вибірці з 65 кадрів підтвердило працездатність системи: загальна точність (Accuracy) склала 93.8%, що свідчить про ефективність базових детекторів на легітимному трафіку. Водночас значення Recall = 40.0% і FN = 3 вказують на наявність сліпих зон, усунення яких є пріоритетним напрямком подальших досліджень. Новизна роботи полягає у методологічному поєднанні процесів контрольованої ін'єкції помилок та автоматизованого розрахунку метрик достовірності детекції через аналіз метаданих повідомлень. Створений інструментарій формує відкриту дослідницьку платформу для бенчмаркінгу нових методів виявлення вторгнень у системах кібербезпеки сучасного транспорту [10].

Література

1. ISO 11898-1:2024. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical coding sublayer. – International Organization for Standardization, 2024. – 86 p.
2. Corrigan S. Introduction to the Controller Area Network (CAN) / S. Corrigan. – Texas Instruments, 2016. – 17 p.
3. Cena G. A Review on the Evolution of CAN-Based Automotive and Industrial Networks / G. Cena, A. Valenzano. – IEEE Transactions on Industrial Informatics, 2020. – 12 p.
4. Farsi M. CANopen Implementation: Applications to Industrial Networks / M. Farsi, K. Ratcliff, M. Barbosa. – Research Studies Press, 2019. – 256 p.
5. SocketCAN – The Linux CAN Subsystem / Volkswagen Research. – 2022. [Електронний ресурс]. Режим доступу: https://www.kernel.org/doc/html/latest/networking/can.html
6. python-can documentation / python-can contributors. – 2023. [Електронний ресурс]. Режим доступу: https://python-can.readthedocs.io/en/stable/
7. Flutter documentation. Architecture overview / Google. – 2024. [Електронний ресурс]. Режим доступу: https://docs.flutter.dev/resources/architectural-overview
8. Dart language documentation / Google. – 2024. [Електронний ресурс]. Режим доступу: https://dart.dev/guides
9. SQLite Documentation / SQLite Development Team. – 2023. [Електронний ресурс]. Режим доступу: https://www.sqlite.org/docs.html
10. Bozdal M. Evaluation of CAN Bus Security Challenges / M. Bozdal, M. Samie, S. Aslam, I. Jennions. – Sensors. – MDPI, 2020. – 19 p.

Design and Verification of a Hardware-Software Platform for CAN Network Error Detection and Visualization

Pavlo Dzelendziak, Rostyslav Nakonechnyi

This paper presents the development of a software environment for simulation, detection, and visualization of errors in Controller Area Network (CAN) systems based on the Flutter framework. Theproposedapproachintroducesanextendedobject-orienteddatamodelwithembeddedGroundTruthlabels,

enablingcontrolledinjectionoffiveanomalytypescombinedwithautomatedcalculationofdetectionaccuracy metrics. Thedevelopedsystemimplementsfivefaultinjectionscenarios: bit-levelmutations, bursterrors, staticpatterninjection, byteorderviolations, and CRC faults. To evaluate detection algorithm performance, a Confusion Matrix is computed in real time, providing Precision, Recall, and F1-Score metrics. Local data persistence is implemented using SQLite, while the system architecture follows a strict separation between processing logic and the user interface, leveraging Dart's asynchronous mechanisms. Practical testing on a dataset of 65 frames confirmed the system's operability and identified directions for further improvement of detection algorithms. Thedevelopedtoolsetcanserveas a benchmarkingplatformfornovelintrusiondetectionmethodsincybersecuritysystemsofmodernautomotivesystems.

Отримано 07.05.2026