Вот перевод документа на русский язык.

---

final.md (Страница 1/6)

Исходный код системы обмена данными БПЛА с высокой степенью защиты на основе технологии безопасности физического уровня

Автор: Цзоу Ин-Чи | **Дата:** 9.06.2023

В этом документе систематизирован исходный код, используемый в данном проекте. Чтобы облегчить пользователям внесение изменений в будущем, автор удалил части кода, фактически используемые при развертывании, и переработал некоторые названия. Пожалуйста, обязательно ознакомьтесь с использованием классов.

Список программных модулей:

- **datastream:** Содержит все функции для обмена данными между различными интерфейсами. Включает интерфейс передачи данных CSI ESP32 через serial (последовательный порт) и интерфейс передачи данных Raspberry Pi через сокет.
- **IoD_UI:** Модуль пользовательского интерфейса, разработанный для демонстрации результатов реализации проекта. Создан с использованием QtDesigner.
- **plkg:** Реализует код, связанный с PLKG, и код шифрования данных.

Немодульные части:

- Файлы, хранящиеся в папке `demo`, демонстрируют базовые функции модулей. Вам просто нужно перетащить их в основную папку (папку, где находятся модули), чтобы увидеть их работу.
- Остальные неупакованные внешние программы являются демонстрационными файлами результатов реализации проекта. Они были написаны довольно ситуативно. Будущим пользователям, которые захотят опираться на это исследование, рекомендуется использовать три предоставленных модуля для создания собственной тестовой среды с целью оптимизации производительности.

Среда:

- python 3.8
- Пакетный менеджер: conda

datastream

chat.py

Используется для создания сокета между устройствами для обмена данными. Устройства должны быть подключены к одной сети. Обмен данными может осуществляться после ввода IP/PORT устройства в этой сети. (Подробности см. в учебнике по основам компьютерных сетей).

Необходимые модули:

- socket
- threading

- re

Описание кода:

Имя функции	Описание
`chat_manager(<string ip>,<int port>)`	Оба устройства вводят IP/PORT устройства, с которым хотят обмениваться данными. Порт (PORT) по умолчанию — 5000. Если вы хотите установить два соединения с одним и тем же устройством, используйте разные порты (основы сетей).

final.md (Страница 2/6)

Имя функции	Описание
`chat_manager.receive_task()`	Постоянно выполняет задачу приема данных.
`chat_manager.recv_init()`	Инициализирует функцию приема данных.
`chat_manager.send_init()`	Инициализирует функцию передачи данных.
`chat_manager.chat_init()`	Инициализирует функции приема и передачи одновременно.
`chat_manager.close_socket()`	Одновременно закрывает сокеты на прием и отправку.
`chat_manager.pop()`	Возвращает первый символ, используя метод FIFO. Формат возврата — код utf-8.
`chat_manager.pop_line()`	Возвращает непрерывные данные, переданные через `chat_manager.send_line()`, используя метод FIFO. Пожалуйста, избегайте использования "-end", так как это символ окончания. Формат возврата — строка (string).
`chat_manager.read_queue()`	Единовременно считывает все данные, находящиеся в памяти.
`chat_manager.queue_clear()`	Очищает все накопленные данные в памяти.
`chat_manager.send(<string message>)`	Передает данные получателю. Автоматически выполняет кодирование в utf-8.
`chat_manager.send_line(<string message>)`	Передает данные получателю. Получатель может использовать `chat_manager.pop_line()` для получения всего сегмента данных.
`chat_manager.send_original(<utf-8 message>)`	Передает данные напрямую после получения данных в кодировке utf-8.

Демонстрация базовых функций:

```
from datastream import chat
import time
import threading

def show(chat):
    while True:
        time.sleep(0.3)
        message = chat.read_queue()
        if len(message) > 0:
            print(message.decode("utf-8"))

Alice = chat.chat("192.168.0.143") # Введите IP другой стороны
Alice.chat_init()
show_thread = threading.Thread(target=show, args=(Alice,))
show_thread.start()

while True:
    Alice.send(input('>>'))
```

---

final.md (Страница 3/6)

Приведенный выше код реализует одновременную передачу и прием данных и может быть использован в качестве образца для вашего проекта.

csi_interface.py

Интерфейс для обмена данными с ESP32 при сборе данных CSI (Channel State Information). Здесь также реализован интерфейс зондирования канала (Channel probing). Тем, кто захочет изменить стратегию сбора данных в будущем, следует смотреть здесь.

Необходимые модули:

- serial
- threading
- platform
- re
- csv

Описание кода:

Имя функции	Описание
`send(<string comport>, <string message>)`	Передает данные другим serial-устройствам через последовательный порт.
`read(<string comport>)`	Принимает данные с определенного последовательного порта (serial port).
`savetocsv(<string filename>, <string data>)`	Сохраняет данные, полученные из `com_esp.aquire_csi()`. Сначала введите имя файла, затем объект для сохранения.

Имя функции	Описание
`com_esp(<string comport>, <int baudrate>)`	Управляет протоколом обмена данными. Взаимодействует с настроенным протоколом ESP32. Введите comport и скорость передачи (baud rate).
`com_esp.set_port(<string comport>,<string baudrate>)`	Сбрасывает настройки порта и скорости передачи, настраивая целевое принимающее устройство.
`com_esp.set_channel(<int channel>)`	Устанавливает канал для сбора данных CSI.
`com_esp.set_ping_f(<int frequency>)`	Устанавливает частоту сбора данных ESP32.
`com_esp.set_timeout(<int t>)`	Устанавливает продолжительность сбора данных.
`com_esp.__monitor()`	Оптимизированное ядро сбора данных.
`com_esp.start_monitor()`	Запускает поток сбора данных.
`com_esp.stop_monitor()`	Останавливает сбор данных.
`com_esp.clear_queue()`	Очищает буфер сбора данных.
`com_esp.send(<string message>)`	Передает данные настроенному ESP32.
`com_esp.send_command(<string command>)`	Передает специальные команды на ESP32: `ping`: начать пинг CSI для сбора получателем; `recv`: начать сбор данных CSI; `check`: подтвердить работоспособность ESP32; `restart`: перезапустить программу ESP32.

final.md (Страница 4/6)

Имя функции	Описание
`com_esp.aquire_csi()`	Организует полученные данные в последовательность строк и возвращает их. Формат: "string", "string" ... "string", "string".
`com_esp.run_collection(<bool priority>,<int frequency>, <int timeout>)`	Выполняет разработанный протокол сбора данных. `priority` определяет порядок сбора данных, `frequency` определяет скорость сбора, `timeout` гарантирует, что время вычислений системы достаточно велико для обеспечения стабильности системы.

load.py

Преобразует данные из `csi_interface.py` в массивы для удобства использования. Удаляет неиспользуемые поднесущие (subcarriers) CSI.

Необходимые модули:

- csv
- re
- numpy
- math

Имя функции	Описание
`amplitude(<float real>, <float imag>)`	Вычисляет величину путем сложения действительной и мнимой частей.
`transform(<Данные CSI из com_esp.aquire_csi()>)`	Преобразует данные в пригодный для использования формат и возвращает `[CSIdata],[CSIdata],[CSIdata]...[CSIdata]`.
`load(<string filename>)`	Извлекает ранее сохраненные данные для воспроизведения прошлых экспериментальных результатов.

plkg

aes.py

Используется для шифрования AES. Принимает ключ и сообщение для выполнения шифрования/дешифрования.

Необходимые модули:

- Crypto
- binascii

Имя функции	Описание
`encrypt(<string plain_text>,<string secret_key>)`	Выполняет шифрование данных.
`decrypt(<utf-8 encrypted_text>,<string secret_key>)`	Дешифрует полученные данные, зашифрованные в utf-8.
`byte_check(<unKnownTypeText>)`	Проверяет тип строки.

ecc.py

Модуль коррекции битовых ошибок.

---

final.md (Страница 5/6)

Необходимые модули:

- bchlib
- random
- math

Имя функции	Описание
`bit_flip(<string binary>)`	Случайным образом инвертирует 0/1 в бинарных данных с вероятностью 10%.
`rand_sequence(<int num>)`	Генерирует случайную двоичную кодировку.
`binary_byte_convertor(<string data>)`	Преобразует строковые двоичные данные в байты.
`byte_binary_convertor(<string data>)`	Преобразует байтовые двоичные данные в строку.
`run_xor(<string binary_sequence1>, <string binary_sequence1>)`	Выполняет XOR (исключающее ИЛИ) между бинарными строками.
`check_same(<string binary_sequence1>, <string binary_sequence1>)`	Проверяет, являются ли два бинарных набора данных полностью идентичными.
`BCH_gen()`	Генерирует случайный код, снабженный кодами коррекции битовых ошибок.
`reconciliation_encode(<string quantization_result>)`	Кодирует результат квантования.
`reconciliation_decode(<string quantization_result>)`	Декодирует результат квантования.

greycode_quantization.py

Набор стратегий квантования.

Необходимые модули:

- copy

Имя функции	Описание
`average(<transform data>)`	Накапливает данные CSI в среднее значение.
`swap(bit,x)`	Алгоритм кода Грея (Greycode).
`gray_code_gen(<int number_of_bits>)`	Функция для генерации кода Грея, возвращает n-битный код Грея.
`quantization_1(<string probing_result>,<int number_of_bits_in_greycode>,<int how_many_bits_for_one_greycode>)`	Выполняет стратегию квантования и возвращает результат квантования.

sha256.py

Выполняет усиление конфиденциальности (privacy amplification).

Необходимые модули:

---

final.md (Страница 6/6)

- hashlib

Имя функции	Описание
`sha_byte(<string quantization_result>)`	Выполняет усиление конфиденциальности для результата квантования с использованием sha256.

plkg.py

Интегрированный подмодуль. Может запускать полный протокол PLKG.

Необходимые модули:

- time

Имя функции	Описание
`end_device(<string device_tag>)`	Настраивает независимое устройство сбора данных. Необходимо задать имя устройства: БПЛА "U", IoT-устройство "I".
`end_device.set_chatmanager(<class chatmanager>)`	Устанавливает `chat_manager` для передачи данных.
`end_device.save_probing_result(<string filename>)`	Устанавливает, нужно ли сохранять результат квантования в файл.
`end_device.time_synchronize()`	При сборе и обмене данными сначала выполняет синхронизацию времени для обеспечения стабильности.
`end_device.channel_probing()`	Выполняет зондирование канала.
`end_device.quantization()`	Выполняет квантование.
`end_device.information_reconciliation()`	Выполняет согласование информации (исправление ошибок).
`end_device.privacy_amplification()`	Выполняет усиление конфиденциальности.
`end_device.plkg()`	Выполняет полный цикл plkg.
`end_device.key`	Ключ plkg.
`end_device.quantization_result`	Результат квантования.