Sprawozdanie  
Projekt zintegrowany  
Komunikacja LORA

18.09.2023

Szymon Nowacki

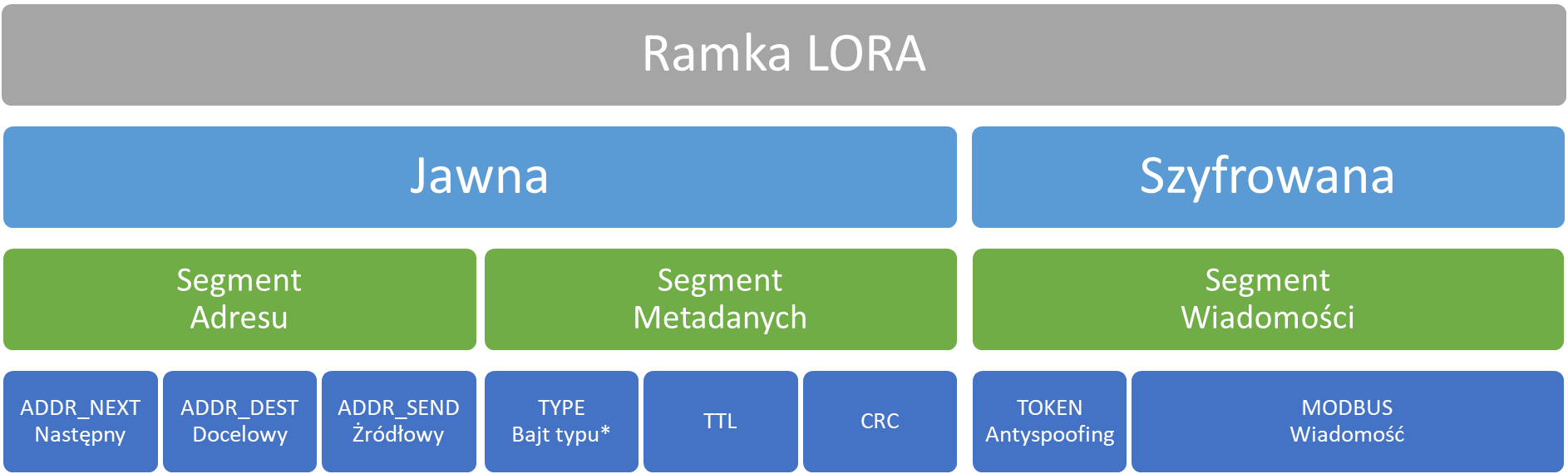
Michał Ficek

Piotr Jureczko

Repozytorium z kodem:  
<https://github.com/Szymon300101/L-CORE/tree/simple-example>

Aby pobrać kod należy kliknąć na zielony przycisk CODE i pobrać .zip. Jest to gotowy projekt PlatformIO z prostym przykładem (omawianym poniżej).

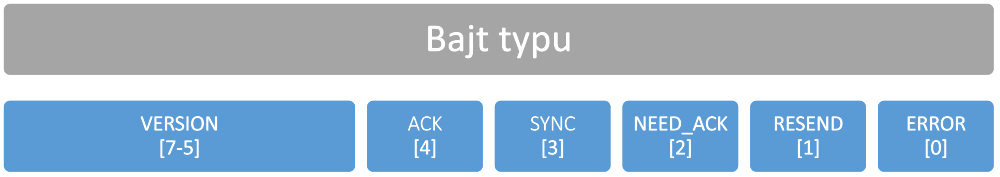
# Opis ramek



Wszystkie pola ramki w segmentach adresu i metadanych, zajmują po 1 bajcie. Wiadomość może mieć dowolną długość jednak cała ramka musi być krótsza od zdefiniowanej w kodzie max. długości ramki. Token domyślnie ma 2 bajty (można to zmienić).

Opis pól ramki:

* ADDR\_NEXT – następne ogniwo w łańcuchu routowania. Urządzenie pod tym adresem powinno odebrać wiadomość i przekierować ją dalej lub zaakceptować.
* ADDR\_DEST – adres docelowy. Wiadomość powinna zakończyć podróż w tym urządzeniu.
* ADDR\_SEND – adres, który wysłał pierwotną wiadomość. Przez cały łańcuch routowania pozostaje ta sama.
* TYPE – opisane poniżej.
* TTL – ilość przekierowań, po których wiadomość powinna się przeterminować.
* CRC – 8-bitowa suma kontrolna. Podczas jej wyliczania, wyzerowane muszą być pola: ADDR\_NEXT, ADDR\_DEST, ADDR\_SEND, CRC (nie są uwzględnione w sumie kontrolnej).
* TOKEN – liczba, która zapobiega próbom wielokrotnego wysyłania tej samej ramki. Urządzenie nadające, podczas wysyłania kolejnych pakietów do danego adresata, stopniowo zwiększa wartość tego pola. Odbiorca musi uważać, czy tokeny się nie powtarzają, dzięki czemu wie, że każda ramka jest unikalna.



Bajt typu przede wszystkim przechowuje flagi, określające funkcję wiadomości. Segment przechowujący wersję ramki, ma na celu zapobieganie błędom odczytu pakietów przy wprowadzaniu zmian do sieci.

* ACK – ramka potwierdzająca otrzymanie poprzedniej wiadomości
* SYNC – ramka wymuszająca synchronizację tokena, wysyłana jako odpowiedź na wiadomość, która miała niepoprawny token. W segmencie wiadomości tej ramki przesyłany jest poprawny token, który należy wykorzystać by wysyłać do tego odbiorcy.
* NEED\_ACK – flaga występująca u większości zwykłych ramek zawierających wiadomości. Oznacza, że nadawca będzie czekał na odpowiedź, więc należy ją zapewnić.
* RESEND – występuje w ramkach odpowiedzi, gdy odbiorcę prosi nadawcę o ponowne wysłanie
* ERROR - występuje w ramkach odpowiedzi, gdy po stronie odbiorcy wystąpił błąd uniemożliwiający przetworzenie ramki. Nie wiadomo czy ponowne wysłanie ramki by pomogło.

# Warstwy protokołu

Poniższe warstwy są inspirowane modelem ISO/OSI jednak nie są z nim całkiem zgodne, role przez nie wykonywane są dopasowane do konkretnych wymagań tej aplikacji.

* Warstwa fizyczna – Protokół LoRa, biblioteka Lora.h (np. moduł TTGO Lora)
* Warstwa łącza danych – Plik LoraRadio.h

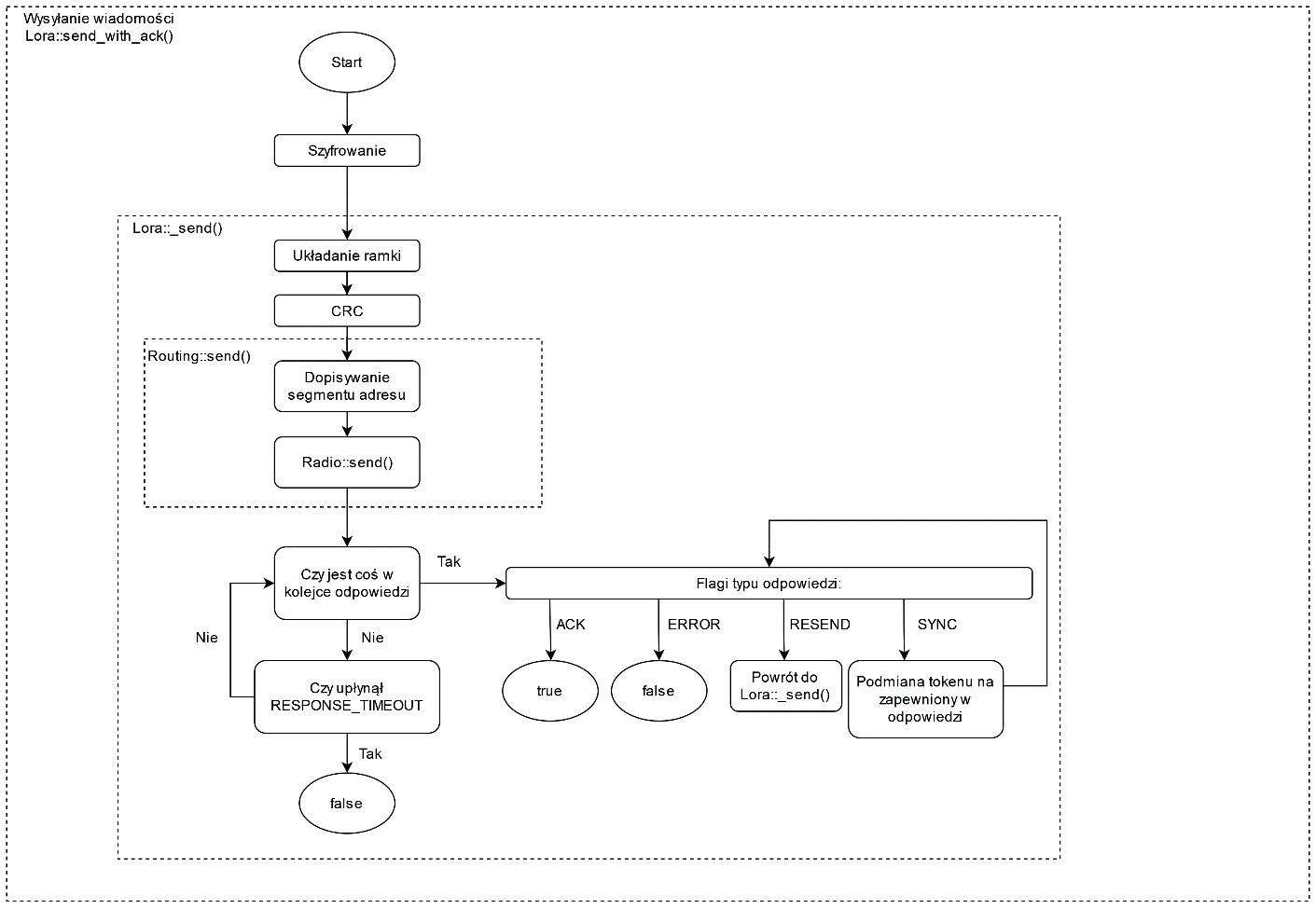
Warstwa odpowiada za obsługę biblioteki Lora.h, opakowując ją w proste funkcje.

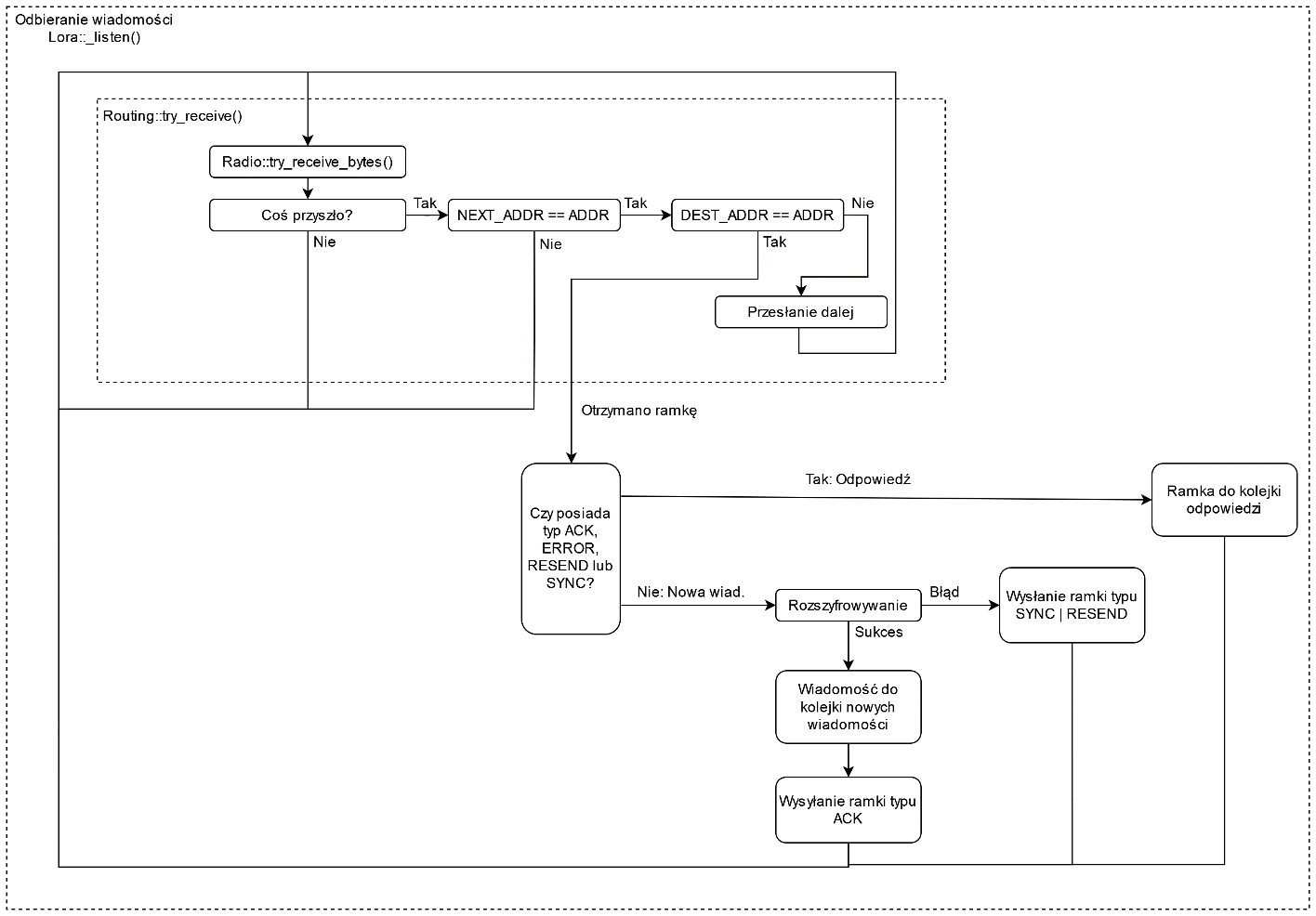
* Warstwa sieciowa – Plik LoraRouting.h

Odpowiada za adresowanie i routowanie pakietów po sieci. Korzystając ze statycznej tabeli routowania przekierowuje nadchodzące pakiety w stronę ich odbiorcy.

* Warstwa transportowa – Plik LoraTransport.h  
  Odpowiada za składanie ramek, crc, szyfrowanie. Obsługuje też cały system odpowiedzi na wiadomości, takie jak ack czy powiadamiania o błędach. Jest to największa warstwa jeśli chodzi o objętość kodu, jak i ilość zadań.

# Schemat struktury najważniejszych funkcji programu

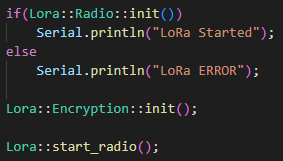




# Wskazówki odnośnie używania i rozwoju

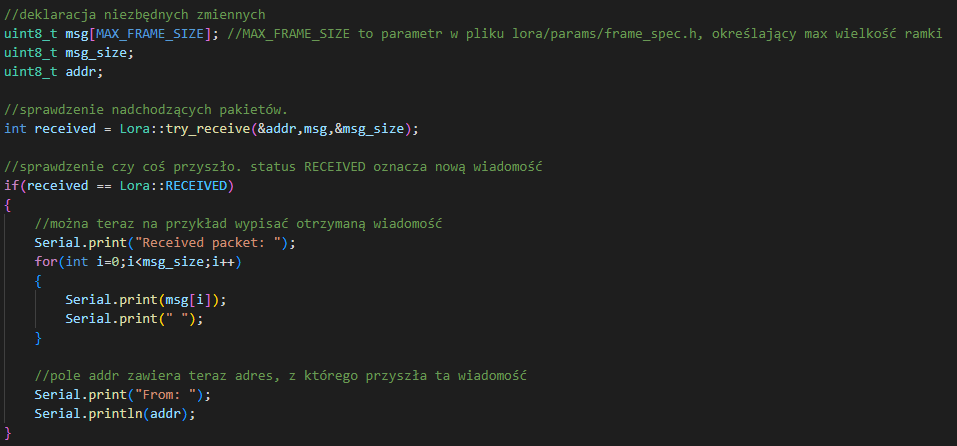
Na początek, żeby było jasne, biblioteka powstała w ciągu paru tygodni w ramach przedmiotu, więc mimo że starałem się by była jak najbardziej stabilna i kompletna, nie jest skończona w takim stopniu jak bym chciał. Wszystkie aspekty komunikacji powinny działać, jednak na pewno znajdą się warunki powodujące błędy. Całość nie została przetestowana w stopniu, z którego byłbym zadowolony, jako że systemy tego typu są trudne w testowaniu domowymi metodami, a dodatkowo dysponowałem ograniczoną ilością modułów. Mimo że starałem się zachować pewną strukturę i logikę w kodzie, nie jest on pozbawiony błędów stylistycznych i żeby był czytelny, pewnie powinien zostać całkiem przepisany. Jako że nie mam na to czasu, pozostaje mi liczyć, że jeżeli ktoś go kiedyś będzie czytał to coś zrozumie.

Jeśli chodzi o samo API (funkcje wystawione użytkownikowi), obsługa biblioteki jest bardzo prosta. W celu inicjalizacji, trzeba wywołać 3 funkcje uruchamiające różne fragmenty biblioteki:



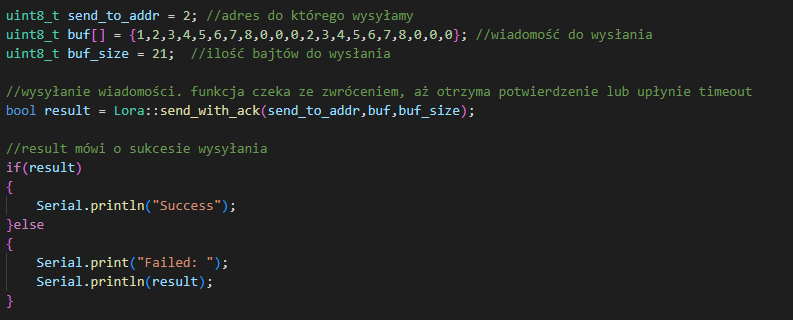
Lora::Radio::init() zwraca informację, czy udało się skomunikować z modułem LoRa.

Odbieranie wiadomości jest bardzo proste:



Funkcja try\_receive(), sprawdza czy przyszły nowe wiadomości (ich odbieranie odbywa się asynchronicznie w tle), i przekazuje je przez parametr. Dlatego trzeba jej zapewnić bufor na dane. Musi on byś wystarczająco duży, więc najlepiej dać mu rozmiar MAX\_FRAME\_SIZE, chyba że użytkownik jest pewien, że będzie dostawał tylko małe wiadomości.

Wysyłanie jest jeszcze prostsze:



Wystarczy przekazać bajty do wysłania i biblioteka zajmuje się resztą.

Uwagi implementacyjne:

* Jako że biblioteka nie koryguje zderzania się pakietów, warstwa aplikacji (użytkownik) musi zająć się tym samemu. Czasem po prostu send\_with\_ack() zwróci false i trzeba spróbować wysłać jeszcze raz lub się poddać. Oczywiście błąd wysyłania może mieć wiele powodów, zwykle jednak chodzi o problem w samym przesyle (rzadziej przy dekodowaniu wiadomości)
* Żeby załączyć bibliotekę do własnego projektu w PlatformIO należy:
  + Przekopiować foldery lib/Lora i lib/Aes do własnego folderu lib/
  + Przekopiować ostatnie wiersze pliku platform.ini (załączone biblioteki)
  + Stworzyć secrets.h według instrukcji w secrets\_TEMPLATE.h i przenieść do swojego projektu

Po tych krokach powinno być możliwe korzystanie z biblioteki. Należy pamiętać o poprawnej konfiguracji adresów i routingu w pliku lib/Lora/user\_params.h.

# Przeprowadzone testy

Testy systemu obejmowały 3 moduły lora:

* ID:0 - wysyła wiadomości zaadresowane do ID:2
* ID:1 – przekierowuje wiadomości między ID:0 i ID:2
* ID:2 – odbiera wiadomości

Moduły były rozmieszczane w różnych odległościach, często w innych pomieszczeniach. Nie wykryto problemów z zasięgiem.

Dodatkowo za testy można uznać udaną implementację biblioteki w oprogramowaniu konkretnych modułów: pilot + relay do światła i czujnik + baza do pomiaru temperatury.