



UMCS

UNIWERSYTET MARII CURIE-SKŁODOWSKIEJ
W LUBLINIE
Wydział Matematyki, Fizyki i Informatyki

Kierunek: **informatyka**

Jan Bylina

nr albumu: 303827

Projekt oraz implementacja systemu gromadzenia rozproszonych danych z wykorzystaniem technologii LoRa

Design and implementation of the distributed data collection system using LoRa technology

Praca licencjacka
napisana w Katedrze Oprogramowania Systemów Informatycznych
Instytutu Informatyki UMCS
pod kierunkiem **dr hab. Przemysława Stpiczyńskiego**

Lublin 2023

Spis treści

Abstract	7
Wstęp	9
1 Wykorzystane narzędzia, technologie i protokoły	11
1.1 LoRa	11
1.2 Urządzenia wykorzystywane w projekcie	12
1.2.1 ESP32	12
1.2.2 Rasberry Pi Pico	13
1.3 Języki programowania i technologie	13
1.3.1 MicroPython for Raspberry Pi Pico	13
1.4 Protokoły komunikacyjne	13
1.4.1 MQTT	13
1.5 Bazy danych i pozostałe technologie	14
1.5.1 InfluxDB 2	14
1.5.2 Docker	14
1.5.3 PlatformIO	14
2 Istniejące rozwiązania	17
2.1 LoRaWAN	17
2.2 Artykuły	17
2.2.1 LoRaMesher	17
2.2.2 Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation	18
2.2.3 A LoRa-Based Mesh Network for Peer-to-Peer Long-Range Communication	19
3 Założenie i Implementacja	21
3.1 Podstawowe cele sieci	21
3.2 Części sieci	21

3.2.1	Węzły sieci	22
3.2.2	Stacja przekaźnikowa	22
3.2.3	Broker wiadomości	22
3.2.4	Baza danych	22
3.3	Wiadomości	24
3.4	Obsługa protokołu	24
3.4.1	Działanie węzłów	24
3.4.2	Działanie przekaźnika	25
3.4.3	Działanie bazy danych i programu zapisującego dane	25
3.5	Implementacja	25
3.5.1	Implementacja węzłów sieci	25
3.5.2	Implementacja stacji przekaźnikowej	26
3.5.3	Implementacja brokera wiadomości	27
3.5.4	Baza danych	27
3.5.5	Program zapisujący dane	27
4	Wdrożenie i testy	29
4.1	Wdrożenie	29
4.1.1	Węzły sieci	29
4.1.2	Stacja przekaźnikowa	29
4.1.3	Serwer testowy	29
4.1.4	Broker wiadomości	32
4.1.5	Baza danych i program zapisujący dane	32
4.1.6	Redis	33
4.2	Testy	33
4.2.1	Poprawność działania	33
4.2.2	Wydajność	35
4.2.3	Zauważone problemy	36
5	Perspektywy rozwoju i dalsze badania	37
5.1	Rozwój projektu	37
5.1.1	Optymalizacja energetyczna węzłów sieci	37
5.1.2	Bezpieczeństwo	38
5.1.3	Komunikacja dwukierunkowa	39
5.1.4	Przygotowanie nowych węzłów sieci	39
5.2	Dalsze badania	39
5.2.1	Badania wydajnościowe dużej sieci	39
5.2.2	Badania konsumpcji energii	40

5.2.3 Badania zasięgu sieci	40
Podsumowanie	41
Spis listingów	43
Spis rysunków	45
Bibliografia	49

Abstract

Wstęp

Celem pracy jest stworzenie systemu zbierającego dane z rozproszonych czujników z wykorzystaniem sieci LoRa oraz ich zapisywanie w bazie danych w celu dalszego przetworzenia. W ramach pracy został stworzony system zbierający dane z czujników temperatury i wilgotności. Dane są zbierane przez autorskie punkty dostępowe LoRa, które następnie przekazują je do serwera. Serwer przechowuje dane w bazie danych.

Rozdział 1

Wykorzystane narzędzia, technologie i protokoły

1.1 LoRa

LoRa (Long Range) to protokół komunikacji bezprzewodowej, stworzony z myślą o urządzeniach IoT. Przeznaczony jest do komunikacji na duże odległości i z niskim zużyciem energii. Został opracowany przez Semtech w 2013 roku, stał się wtedy otwartym standardem. LoRa wykorzystuje zastrzeżoną technologię modulacji widma rozproszonego, która umożliwia daleki zasięg, przy ograniczonym zużyciu energii. Implementuje on warstwę fizyczną sieci [29].

Teoretyczny zasięg sieci LoRa wynosi około 15 km w terenie wiejskim i 2–5 km w terenie zurbanizowanym. W praktyce zasięg ten jest ograniczony przez wiele czynników, takich jak:

- moc nadajnika
- czułość odbiornika
- przeszkody fizyczne
- zakłócenia
- wysokość anteny

Więcej na temat wydajności sieci można przeczytać w artykule [3].

1.2 Urządzenia wykorzystywane w projekcie

1.2.1 ESP32

ESP32 to jednoukładowy mikrokontroler, zaprojektowany i produkowany przez firmę Espressif Systems. Jego najważniejsze cechy to:

- energooszczędny procesor RISC o częstotliwości do 240 MHz
- 520 kB pamięci SRAM
- WiFi 802.11 b/g/n
- Bluetooth
- liczne interfejsy cyfrowe i analogowe, w tym:
 - UART
 - I2C
 - SPI
 - I2S
 - CAN
 - ADC
 - DAC
 - PWM
 - Ethernet MAC
 - USB 2.0
- ...

[27]

Powstało wiele wersji tego układu, różniące się m.in. szybkością procesora, wielkością pamięci flash, ilością pinów, interfejsów cyfrowych i analogowych, a także możliwością pracy w trybie bezprzewodowym (WiFi) lub przewodowym (Ethernet) [28]. Najczęściej układ te wykorzystywane różnych projektach IoT, zarówno jako czujniki, jak i serwery [27].

W projekcie ESP32 zostało wykorzystane w dwóch płytach TTGO T3 V1.6.1. Jedna z płyt została wykorzystana jako przekaźnik danych pomiędzy siecią LoRa a siecią WiFi, a druga jako część systemu zbierania danych z wykorzystaniem LoRa

1.2.2 Raspberry Pi Pico

Raspberry Pi Pico to płytka z mikrokontrolerem RP2040, zaprojektowana i produkowana przez firmę Raspberry Pi Foundation. Charakteryzuje się ona dwurdzeniowym procesorem ARM Cortex-M0+ o częstotliwości 133 MHz, 264 kB pamięci SRAM oraz 2 MB pamięci flash. Płytką posiada również wiele interfejsów cyfrowych i analogowych, w tym:

- UART
- I2C
- SPI
- I2S
- ADC
- DAC
- PWM
- USB 1.1

[19, 18] Płytką ta jest często wykorzystywana przez hobbystów do różnych projektów IoT, a także jako sterownik silników, czy kontroler robotów.[18]

W projekcie dwie płytki zostały wykorzystane jako część systemu zbierania danych.

1.3 Języki programowania i technologie

1.3.1 MicroPython for Raspberry Pi Pico

MicroPython to lekka i wydajna implementacja języka Python 3, która została zaprojektowana z myślą o urządzeniach wbudowanych. Zawiera on okrągłą wersję biblioteki standardowej Pythona. MicroPython jest dostępny na wiele platform, w tym na Raspberry Pi Pico. [23]

1.4 Protokoły komunikacyjne

1.4.1 MQTT

MQTT(MQ Telemetry Transport) to lekki protokół przesyłania wiadomości, często wykorzystywany w IoT, gdzie ważna jest energooszczędność. Oparty jest o model

publish/subscribe. Oznacza to zorganizowanie wiadomości w tematy, a klienci mogą subskrybować określone przez siebie tematy, aby otrzymywać odpowiednie wiadomości. Zapewnia to wydają komunikację przy małym obciążeniu systemu. [4]

1.5 Bazy danych i pozostałe technologie

1.5.1 InfluxDB 2

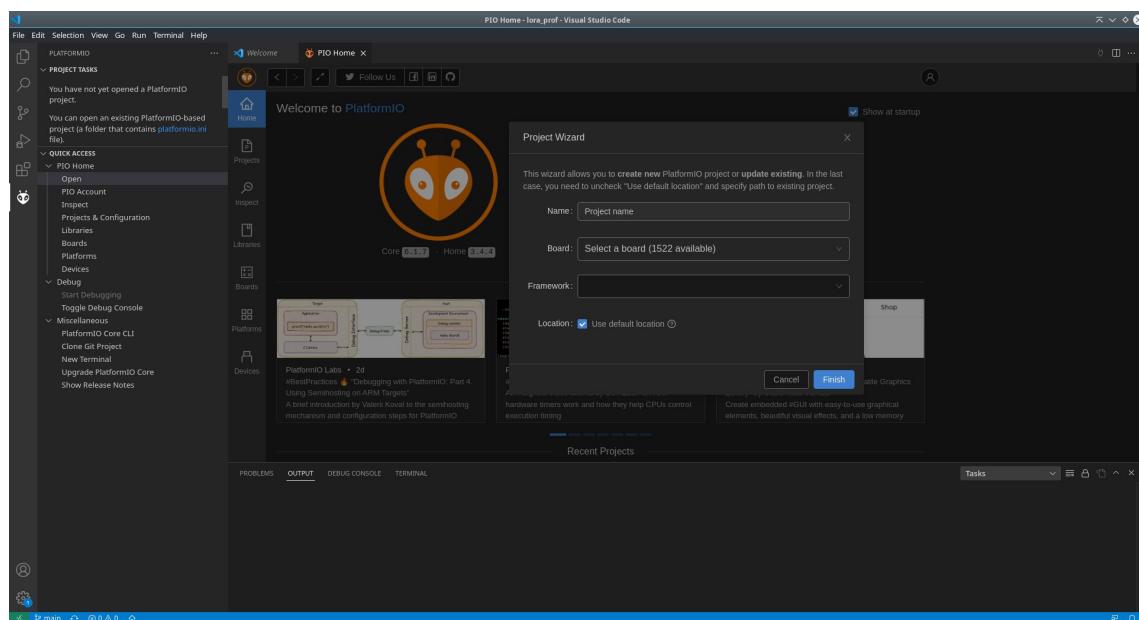
InfluxDB 2 to baza danych szeregow czasowych, stworzona przez InfluxData, Została napisana w języku Go. Wykorzystywana jest do zbierania danych z, między innymi urządzeń IoT, metryk aplikacji, monitoringu infrastruktury IT. [20]

1.5.2 Docker

Docker to narzędzie do wirtualizacji na poziomie systemu operacyjnego. Pozwala ono na uruchamianie aplikacji w kontenerach, które są odizolowane od siebie i od systemu operacyjnego. Kontenery są lżejsze od wirtualnych maszyn, ponieważ nie muszą zawierać systemu operacyjnego, a tylko potrzebne do działania aplikacji biblioteki i pliki. [14]

1.5.3 PlatformIO

PlatformIO to narzędzie do tworzenia i zarządzania projektami z wykorzystaniem mikrokontrolerów. Pozwala ono na tworzenie projektów w językach C i C++. PlatformIO oferuje również możliwość zarządzania zależnościami projektu, zarządzania bibliotekami, a także komplikację i wgrywanie projektu na płytę. Najważniejszą cechą jest możliwość tworzenia projektów dla wielu platform i z wykorzystaniem różnych narzędzi, w tym dla ESP32 i Raspberry Pi Pico, będąc jednocześnie niezależnym od jednego producenta. [25]



Rysunek 1.1: Interfejs narzędzia PlatformIO w edytorze Visual Studio Code

Rozdział 2

Istniejące rozwiązania

2.1 LoRaWAN

LoRaWAN (ang. LoRa Wide Area Network) to protokół komunikacji stworzony przez **LoRa Alliance** [29]. Protokół ten został stworzony z myślą o zastosowaniach w sieciach IoT. Protokół ten jest oparty o protokół LoRa i pozwala na budowę sieci LoRa w oparciu o bramy. Węzły sieci komunikują się z bramami, a bramy przekazują dane do serwera. Serwer jest odpowiedzialny za przetworzenie danych i przekazanie ich do aplikacji użytkownika.

2.2 Artykuły

2.2.1 LoRaMesher

Badacze w artykule zaproponowali otwartoźródłową implementację autorskiego protokołu **LoRaMesher** [10] w postaci darmowej biblioteki. Biblioteka ta pozwala na budowę sieci opartej o LoRa bez użycia bram. Autorzy zaimplementowali protokół w języku C++ z użyciem systemu FreeRTOS. Biblioteka została przetestowana na platformie ESP32.

W artykule przedstawiono wyniki testów przeprowadzonych w rzeczywistych warunkach. Wiadomości były wysyłane co 120 sekund, a wiadomości routingowe co 300 sekund. Testy zostały przeprowadzone w trzech różnych scenariuszach:

- Sieć złożona z 10 węzłów sieci, oddalone od siebie o jeden skok. W takiej sieci wskaźnik dostarczenia pakietów wynosił 90%.
- Sieć złożona z 10 węzłów, ułożonych w łańcuch. W tym wypadku wskaźnik dostarczenia pakietów wynosił 96%.

- Sieć złożona z 10 węzłów, w tym 5 węzłów działało tylko jako przekaźniki. Architektura sieci symulowało zastosowanie biblioteki w prawdziwym zastosowaniem. W tym wypadku wskaźnik dostarczenia pakietów wynosił 86%.

Wyniki testów pokazują, że biblioteka działa poprawnie i może być zastosowana w prawdziwych zastosowaniach. Jednak w artykule nie zostały przedstawione wyniki testów wydajnościowych, które mogłyby pokazać, jak dużo pakietów może być obsługiwanych przez bibliotekę.

W odróżnieniu od rozwiązania zaproponowanego w artykule, w projekcie została zaproponowane rozwiązanie wymagające jednokierunkowej komunikacji z węzłami sieci. Dzięki temu można zastosować węzły sieci o mniejszej wydajności, co pozwala zarówno na zmniejszenie kosztów projektu jak i działania węzłów na baterii.

Biblioteka ta została również przygotowana z myślą o zastosowaniach, w których część działania logiki systemu odbywa się na węzłach sieci. W projekcie zastosowano rozwiązanie, w którym cała większego systemu jest zaimplementowana w stacji przekaźnikowej i aplikacji zapisującej do bazy danych. Dzięki temu można zastosować węzły sieci o mniejszej wydajności i zmniejszyć poziom skomplikowania systemu.

2.2.2 Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation

Badacze w podanym artykule [2] zaproponowali kompletne rozwiązanie umożliwiające monitorowanie dużego obszaru kampusu uniwersyteckiego, w oparciu tylko o protokół LoRa. W odróżnieniu od wyżej opisanego artykułu, autorzy zaproponowali rozwiązanie, w którym to stacja bazowa pobiera dane od poszczególnych węzłów, a nie węzły samoistnie wysyłają te dane do stacji bazowej.

Główny test został poprzedzony kilkoma mniejszymi testami, które pozwoliły ustalić najlepsze parametry połączenia między węzłami jak i ich najlepsze umiejscowienie.

System był testowany przez 8 dni na obszarze 800 na 600 metrów. Test zawierał jednąację bazową i 20 węzłów sieci. Dane były pobierane co 60 sekund. Wskaźnik dostarczalności pakietów był w okolicach 88.49%.

Cechą charakterystyczną tego rozwiązania jest sposób budowania sieci. Węzeł podczas dołączenia do sieci sam decyduje przez który z sąsiadujących węzłów nawiąże komunikację. Oczywiście komunikacja może zostać również nawiązana bezpośrednio ze stacją bazową. Niestety sam proces dołączania nowego urządzenia do sieci i wyboru węzła-przekaźnika nie jest opisany w tym artykule. Największą różnicą między tym rozwiązaniem a rozwiązaniem zaproponowanym w projekcie jest sposób komunikacji między węzłami. W projekcie zastosowano rozwiązanie, w którym węzły nie komunikują się każdy z każdym, a jedynie z wybranym węzłem (rodzicem). Takie rozwiązanie

pozwala na zmniejszenie ilości pakietów, które muszą być przetworzone przez węzły sieci, co pozwala na zastosowanie węzłów o mniejszej wydajności. Jednak węzły muszą być w stanie komunikować się z wybranym w nieznanym procesie rodzicem, co może być problematyczne w przypadku dużych odległości między węzłami. Co więcej, w przypadku zerwania połączenia z przekaźnikiem, węzeł musi ponownie nawiązać połączenie z siecią, co może prowadzić do straty danych.

Drugim znaczącym różnicą jest sposób komunikacji między węzłami sieci. W projekcie zastosowano rozwiązańe, w którym to stacja bazowa odpowiada za pobieranie danych od węzłów sieci. Jednak w takim rozwiązańiu stacja bazowa musi być w stanie obsługiwać wszystkie węzły sieci, co może być problematyczne w przypadku dużych sieci. Jeżeli stacja ulegnie awarii, tracone.

W przypadku rozwiązania zaproponowanego w projekcie, stacja bazowa może być zaimplementowana w taki sposób, aby obsługiwać tylko część węzłów sieci, co pozwala na zastosowanie stacji bazowej o mniejszej wydajności. W prosty sposób można również zwiększyć wydajność stacji bazowej, poprzez zastosowanie większej ilości stacji bazowych.

2.2.3 A LoRa-Based Mesh Network for Peer-to-Peer Long-Range Communication

W podanym artykule [9] badacze zaproponowali implementację sieci mesh opartej o protokół LoRa z wykorzystaniem biblioteki RadioHead i systemu FreeRTOS na mikrokontrolerach ESP32. W odróżnieniu od wyżej opisanych artykułów, autorzy zaproponowali rozwiązanie, w którym to węzły sieci komunikują się ze sobą bezpośrednio, a nie za pośrednictwem stacji bazowej. W podanym artykule zostały również przedstawione wyniki testów wydajnościowych, takich jak:

- czas potrzebny na przesłanie pakietu między węzłami sieci,
- przetwarzanie kilku pakietów między węzłami sieci jednocześnie
- skuteczność przesyłu pakietów między węzłami sieci w zależności od odległości między nimi i ustawień transmisji.

Wyniki testów pokazują, że biblioteka działa poprawnie i może być zastosowana w prawdziwych aplikacjach. Jednak w artykule nie zostały przedstawione wyniki testów wydajnościowych, które mogłyby pokazać, jak dużo pakietów może być obsłużonych przez bibliotekę.

Główną różnicą między tym rozwiązaniem a rozwiązaniem zaproponowanym w projekcie jest sposób komunikacji między węzłami sieci. W artykule zaproponowano

architekturę bez stacji bazowej, więc by dane mogły być przetwarzane poza węzłami sieci, stację bazową należałoby stworzyć we własnym zakresie. Nie powinno być to trudne, ponieważ mikrokontrolery przesyłają odebrane pakiety przez interfejs UART, więc stacja bazowa mogłaby być zaimplementowana na komputerze PC.

Rozdział 3

Założenie i Implementacja

W poniższym rozdziale zaprocentowano założenia, potrzeby i projekt projektu systemu na bazie LoRa

3.1 Podstawowe cele sieci

System ma kilka podstawowych założeń:

- Jeden centralny punkt gromadzenia danych
- Zapisywanie danych do bazy danych szeregów czasowych, w celu ich dalszego przetwarzania
- Zbieranie danych z dużego obszaru
- Niezależność od istniejących metod przesyłu danych (WiFi, Sieci komórkowe, Łączność satelitarna)
- Niezależność od platformy sprzętowej
- Zapewniać możliwie dużą dostarczalność pakietów
- Dane czasowe nie muszą być super dokładne (dopuszczalne są drobne opóźnienia)
- Węzły sieci tylko wysyłają dane, same nie konsumują przychodzących wiadomości

3.2 Części sieci

W celu uzyskania wyżej wspomnianych założeń, zaproponowano system złożonych z kilku części:

- Węzły sieci
- Stacje przekaźnikowe
- Broker wiadomości
- Baza danych

Na rysunku 3.1 przedstawiono schemat ideowy systemu.

3.2.1 Węzły sieci

Węzły sieci to urządzenia wyposażone w moduł LoRa i odpowiednie oprogramowanie pozwalające na pełną obsługę sieci. Gdy węzeł odbierze wiadomość, sprawdza jej poprawność i rozsyła ją dalej w celu zapewnienia jak największego zasięgu i dostarczalności.

Urządzenie to może być również wyposażone w różnego rodzaju czujniki, które dostarczają danych bazy danych.

3.2.2 Stacja przekaźnikowa

Stacja przekaźnikowa to urządzenie wyposażone zarówno w moduł LoRa jak i moduł umożliwiający komunikację z siecią Internet (np. moduł WiFi lub moduł bazy Ethernet).

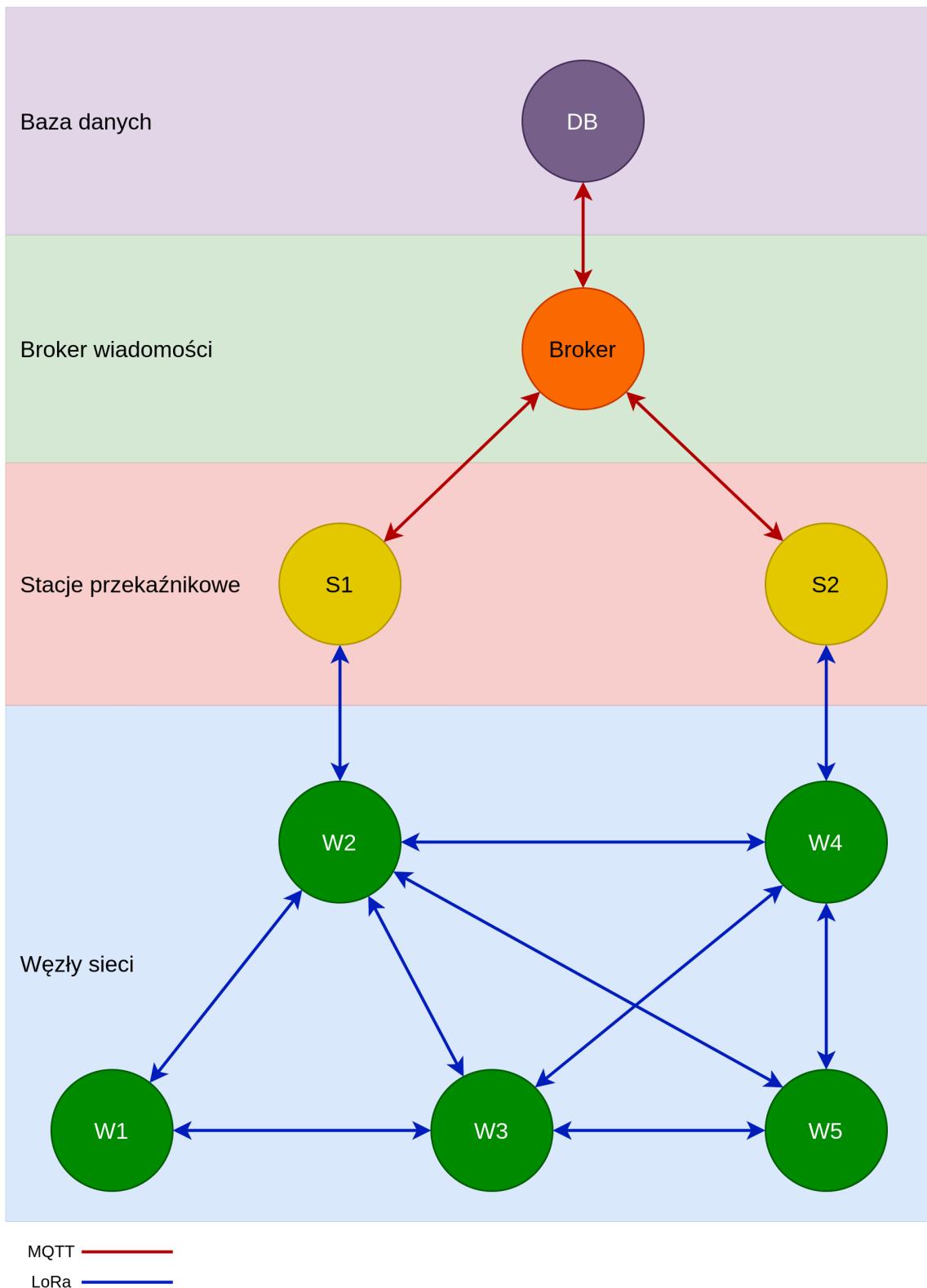
Urządzenie to odbiera przychodzące wiadomości LoRa i przesyła je do brokera wiadomości

3.2.3 Broker wiadomości

Broker wiadomości to program, działający na komputerze mającym dostęp do sieci, umożliwia on wydajną komunikację pomiędzy Stacją Przekaźnikową a Bazą Danych

3.2.4 Baza danych

Baza Danych umożliwiająca zapisywanie sporej ilości danych, uwzględniając również ich czas (baza danych szeregow czasowych). O zapisy danych z brokera wiadomości do bazy danych dba osobny program, który powinien sprawdzać również poprawność tych wiadomości, jak i dbać o to by nie zapisywać powtórzonych wiadomości



Rysunek 3.1: Diagram prezentujący schemat ideowy systemu

3.3 Wiadomości

Każda z wiadomości przesyłanych za pomocą tych sieci powinna mieć formę jak zaprezentowano na listingu 3.1

Zawiera ona pola:

- **ttl** — [Ang. time to live — czas życia] Wartość określająca maksymalną liczbę skoków pomiędzy węzłami sieci. Domyslnie wynosi 10, może zostać wydłużona w zależności od wielkości planowanej sieci
- **m_id** — UUID [1] wiadomości, gwarantujący niepowtarzalność tej wiadomości. Ułatwia również jej dalsze przetwarzanie
- **d_id** — numer identyfikacyjny urządzenia z którego pochodzi wiadomość
- **values** — słownik zawierający dane z urządzenia, do zapisania w bazie

```

1  {
2      "d_id": "id_233",
3      "values": {
4          "temp": "21",
5          "hum": "50",
6          "press": "1000",
7          "light": "100",
8          "co2": "1000",
9          "pm25": "10",
10         "pm10": "20"
11     },
12     "ttl": 10,
13     "m_id": "eaa17a7b-9388-43b6-9310-731c942fc6b9"
14 }
```

Listing 3.1: Przykładowa wiadomość przesyłana przez system

3.4 Obsługa protokołu

3.4.1 Działanie węzłów

Wiadomości generowane są przez węzły sieci, zawierając wszystkie niezbędne pola (wymienione wyżej) i odczyty z czujników zamieszczonych na węźle. Następnie zostaje ona rozesłana do wszystkich węzłów w zasięgu (broadcasting).

Węzeł odbierając wiadomość, sprawdza jej poprawność (czy jest odpowiednio sformatowana, czy zawiera wszystkie potrzebne pola), i jeżeli wiadomość jest poprawna, a pole `ttl` jest większe od 0 rozsyła wiadomość dalej. Sprawdzanie wiadomości odbywa się by wyeliminować wiadomości niepoprawne z sieci.

3.4.2 Działanie przekaźnika

Przekaźnik odbiera wiadomości, i przesyła je do brokera wiadomości. Nie sprawdza poprawności wiadomości by zapewnić maksymalną wydajność i niezawodność.

3.4.3 Działanie bazy danych i programu zapisującego dane

Program pobiera kolejne wiadomości od brokera i przetwarza je w kolejności:

1. Sprawdzenie poprawności wiadomości
2. Sprawdzenie czy wiadomość nie została już sprawdzona (na podstawie pola `m_id`)
3. Zapisanie danych ze słownika `values` do bazy danych i przyporządkowanie ich do `d_id`, oznaczenie ich znacznikiem czasowym.
4. Zapisanie w pamięci operacyjnej `m_id`, potem do wykorzystania w kroku 2

Baza danych powinna przechowywać dane, takie jak:

- `d_id` — identyfikator urządzenia
- znacznik czasowy
- wartość pomiaru
- nazwa pomiaru

3.5 Implementacja

3.5.1 Implementacja węzłów sieci

W wyniku pracy nad systemem zbierania danych przygotowano implementację węzłów sieci w wykorzystaniem 2 platform sprzętowych:

- Raspberry Pi Pico
- TTGO LoRa32

Raspberry Pi Pico

W wyniku pracy nad systemem zostały przygotowane dwa, identyczne urządzenia oparte o Raspberry Pi Pico. Urządzenie zostało wyposażone w moduł LoRa SX1262 [5](z wykorzystaniem płytki rozwojowej Waveshare SX1262 LoRa Node Module [6]) oraz czujnik temperatury i wilgotności DHT11. Urządzenie zostało zaprogramowane w języku MicroPython z wykorzystaniem biblioteki `micropySX126X` [22]. Urządzenie wysyła wiadomości zawierające odczyty z czujnika co 5 minut. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie.

TTGO LoRa32

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [8]. Urządzenie zostało zaprogramowane w języku C++ z użyciem PlatformIO, ekosystemu do programowania urządzeń IoT. [25]. W programie została wykorzystane biblioteki:

- `arduino-LoRa` — biblioteka do obsługi modułu LoRa [12]
- `ArduinoJson` — biblioteka do obsługi formatu JSON [13]
- `Adafruit-SSD1306` — biblioteka do obsługi wyświetlacza [11] OLED
- `ESPRandom` — biblioteka do obsługi sprzętowego generatora liczb pseudolosowych [16]

Urządzenie wysyła wiadomości zawierające odczyty z czujnika co 5 minut. Wiadomości zawierają wartości losowe, wygenerowane za pomocą sprzętowego generatora liczb pseudolosowych zintegrowanego z mikrokontrolerem ESP32. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie.

3.5.2 Implementacja stacji przekaźnikowej

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o płytę TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [8] oraz moduł WiFi ESP32 [27]. Urządzenie zostało zaprogramowane w języku C++ z wykorzystaniem PlatformIO [25] W programie została wykorzystane biblioteki:

- `arduino-LoRa` — biblioteka do obsługi modułu LoRa [12]
- `Adafruit-SSD1306` — biblioteka do obsługi wyświetlacza OLED [11]

- WiFi — biblioteka do obsługi modułu WiFi ESP32, zawarta w pakiecie arduino-esp32 [17]
- PubSubClient — biblioteka do obsługi protokołu MQTT [7]

Urządzenie odbiera wiadomości LoRa i przesyła je do brokera wiadomości za pomocą protokołu MQTT. Wysyłanie wiadomości odbywa się w sposób asynchroniczny, dzięki czemu urządzenie może wykonywać inne operacje w tym czasie. Urządzenie wyświetla na ekranie OLED informacje o stanie sieci LoRa, oraz otrzymanych wiadomościach. Urządzenie nie sprawdza poprawności wiadomości, by zapewnić jak największą wydajność i niezawodność.

3.5.3 Implementacja brokera wiadomości

Aby zapewnić niezawodność i wysoką wydajność komunikacji zdecydowano użyć otwartoźródłowego brokera wiadomości Mosquitto [24]. Jest to sprawdzony, wydajny i niezawodny program, który jest szeroko stosowany w systemach IoT, również w przemyśle. [24]

3.5.4 Baza danych

W celu zapisywania danych zdecydowano użyć bazy danych InfluxDB 2.0 [20]. Jest to baza danych szeregow czasowych, która jest wydajna i niezawodna. [20]

3.5.5 Program zapisujący dane

W celu zapisywania danych zdecydowano użyć programu napisanego w języku Python. Program został napisany w języku Python, z wykorzystaniem bibliotek:

- paho-mqtt — biblioteka do obsługi protokołu MQTT [15]
- influxdb-client — biblioteka do obsługi bazy danych InfluxDB [21]
- redis-py — biblioteka do obsługi bazy danych Redis [26]

Ostania, wymieniona biblioteka została użyta do połączenia się z bazą danych w pamięci w celu przechowywania identyfikatorów wiadomości, które zostały już sprawdzone. Dzięki temu program nie zapisuje powtórzonych wiadomości do bazy danych.

Rozdział 4

Wdrożenie i testy

4.1 Wdrożenie

System został wdrożony 1 maja 2023 roku. Wdrożenie polegało na uruchomieniu serwera z bazą danych, jedną stację przekaźnikową oraz uruchomieniu 3 urządzeń pomiarowych.

4.1.1 Węzły sieci

Trzy węzły sieci zostały rozmieszczone na małym obszarze, około 30 metrów od siebie, w dwóch budynkach. Dwa urządzenia (jedno Raspberry Pi Pico i jedno TTGO LoRa32) zasilane są z sieci, jedno za pomocą baterii. Urządzenia zostały połączone z czujnikami za pomocą zwykłych kabli prototypowych, jak na zdjęciu 4.1 i 4.3. Przykład urządzenia wykorzystującego wbudowane czujniki został zaprezentowany na zdjęciu 4.3

4.1.2 Stacja przekaźnikowa

Stacja przekaźnikowa została umieszczona w pobliżu głównego serwera, w zasięgu sieci WiFi, wewnątrz budynku. Urządzenie zostało zasilone z sieci elektrycznej. Na ekranie urządzenia wyświetlane są informacje o stanie urządzenia, jak na zdjęciu 4.4.

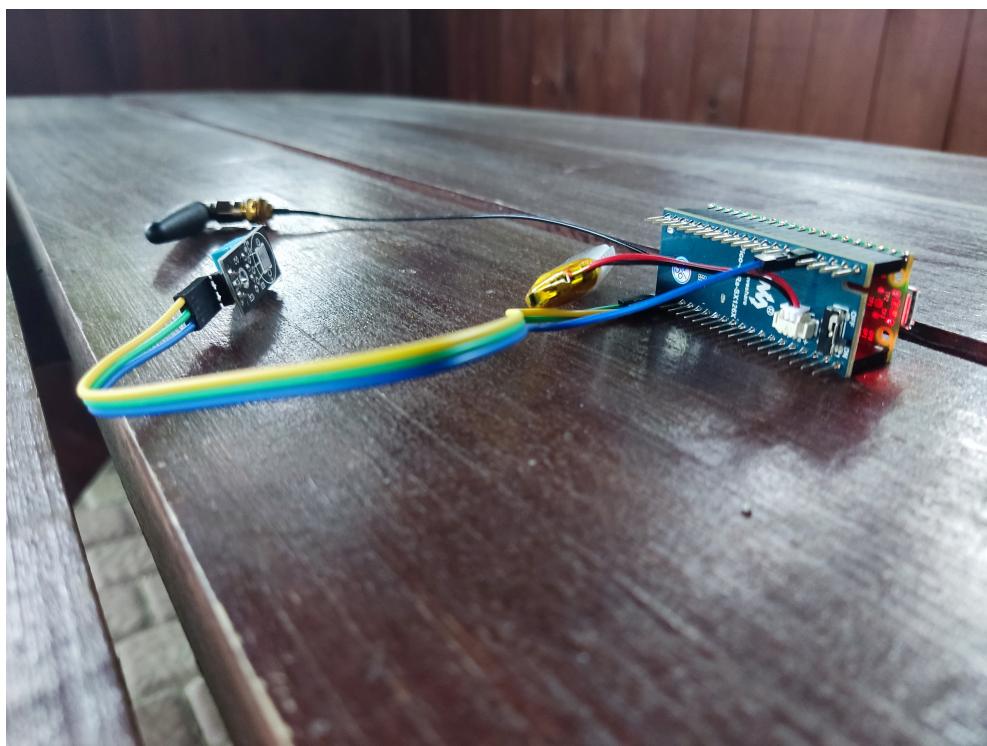
4.1.3 Serwer testowy

Na serwerze testowy zostały uruchomione wszystkie usługi, które zostały wykorzystane w projekcie, w szczególności:

- Mosquitto
- InfluxDB 2

- Program zapisujący do InfluxDB 2
- Redis

Serwer został uruchomiony na maszynie wewnętrz budynku, w sieci lokalnej. Na serwerze został uruchomiony system operacyjny Debian 11. Maszyna została wyposażona w procesor Intel Core i5-480M, 2 GB pamięci RAM oraz dysk SSD o pojemności 256 GB. Wszystkie usługi zostały uruchomione w kontenerach Dockera [14]. Wszystkie usługi



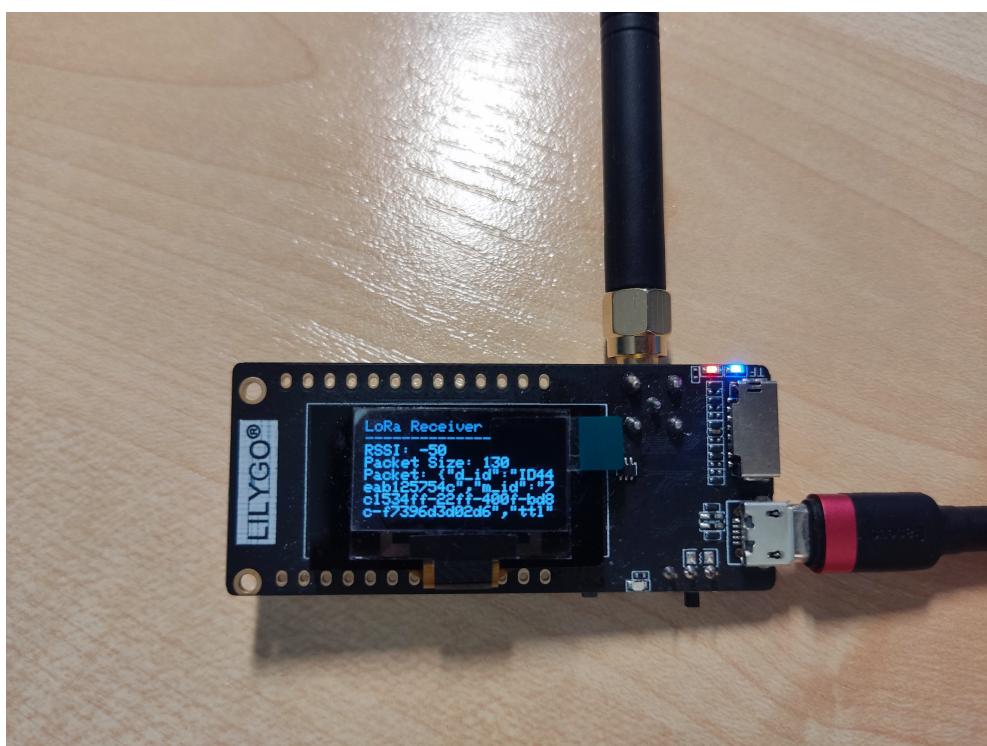
Rysunek 4.1: Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią

```
[23:13:53.937] INFO: Measuring...
[23:13:54.191] INFO: Temperature: 22 C
[23:13:54.194] INFO: Humidity: 40 %
[23:13:54.235] INFO: Sending: {"ttl": 10, "values": {"humidity": 40, "temperature": 22}, "m_id": "c9731485-667c-4faf-95fd-5639da88c076", "d_id": "ID_e6614104038c5938"}
[23:13:54.508] DEBUG: TX done.
[23:13:54.834] INFO: Message received: {"ttl": 9,"values": {"humidity":40,"temperature":22}, "m_id": "c9731485-667c-4faf-95fd-5639da88c076", "d_id": "ID_e6614104038c5938"}
[23:13:54.835] DEBUG: RX OK.
[23:18:21.836] INFO: Message received: {"ttl": 10, "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "d_id": "ID_e66118c4e36192a"}
[23:18:21.797] INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 66, "temperature": 17}, "ttl": 9, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:21.936] DEBUG: TX done.
[23:18:22.331] INFO: Message received: {"d_id": "ID_e66118c4e330192a", "values": {"humidity":66,"temperature":17}}, "ttl": 8, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"
[23:18:22.331] DEBUG: Message forwarded: {"d_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 7, "d_id": "ID_e66118c4e336192a"}
[23:18:22.331] DEBUG: TX done.
[23:18:22.331] INFO: Message received: {"d_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity":66,"temperature":17}}, "ttl": 7, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"
[23:18:23.598] INFO: Message received: {"m_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity":66,"temperature":17}}, "ttl": 6, "d_id": "ID_e66118c4e336192a"
[23:18:24.269] INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "ttl": 5}'
[23:18:24.528] DEBUG: TX done.
[23:18:24.880] INFO: Message received: {"d_id": "ID_e66118c4e330192a", "values": {"humidity":66,"temperature":17}}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "ttl": 4
[23:18:24.880] DEBUG: Message forwarded: b'{"ttl": 1, "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:25.140] DEBUG: TX done.
[23:18:26.149] INFO: Message received: {"d_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 1, "m_id": "ID_e66118c4e336192a"}
[23:18:26.819] INFO: Message forwarded: b'{"d_id": "ID_e66118c4e336192a", "values": {"humidity": 66, "temperature": 17}, "ttl": 1, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:27.079] DEBUG: TX done.
[23:18:27.432] INFO: Message received: {"d_id": "ID_e66118c4e330192a", "values": {"humidity":66,"temperature":17}}, "ttl": 10, "d_id": "5432817c-37f6-4957-871a-59545a6b359d"
[23:18:27.432] DEBUG: Message dropped: b'{"d_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 10, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:45.172] INFO: Message received: {"d_id": "ID_44eb125754c", "values": {"random_values": 78, "random_values2": 71}}, "ttl": 9, "d_id": "ID_44eb125754c"
[23:18:45.172] DEBUG: Message forwarded: {"d_id": "ID_44eb125754c", "values": {"random_values": 78, "random_values2": 71}}, "ttl": 9, "m_id": "ID_44eb125754c"
[23:18:45.843] INFO: Message forwarded: b'{"d_id": "ID_44eb125754c", "values": {"random_values": 78, "random_values2": 71}}, "ttl": 9, "m_id": "ID_44eb125754c"
[23:18:46.107] DEBUG: TX done.
[23:18:46.552] INFO: Message received: {"d_id": "ID_44eb125754c", "ttl": 8, "values": {"random_values": 78, "random_values2": 21}}, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"
[23:18:46.552] DEBUG: Message forwarded: b'{"d_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "ttl": 7, "values": {"random_values": 78, "random_values2": 21}}, "d_id": "ID_44eb125754c"
[23:18:47.591] DEBUG: TX done.
[23:18:47.946] INFO: Message received: {"d_id": "ID_44eb125754c", "ttl": 6, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"}, "values": {"random_values": 78, "random_values2": 21}
[23:18:48.617] INFO: Message forwarded: b'{"values": {"random_values": 78, "random_values2": 21}, "d_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "ttl": 5, "m_id": "ID_44eb125754c"}'
[23:18:48.882] DEBUG: TX done.
[23:18:49.343] INFO: Message received: {"d_id": "ID_44eb125754c", "ttl": 4, "values": {"random_values": 78, "random_values2": 21}}, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"
[23:18:49.343] DEBUG: Message forwarded: {"d_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "values": {"random_values": 78, "random_values2": 21}}, "d_id": "ID_44eb125754c"
[23:18:50.276] DEBUG: TX done.
[23:18:50.276] INFO: Message received: {"d_id": "ID_44eb125754c", "ttl": 2, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"}, "values": {"random_values": 78, "random_values2": 21}
[23:18:50.720] INFO: Message received: {"d_id": "ID_44eb125754c", "ttl": 2, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"}, "values": {"random_values": 78, "random_values2": 21}}
```

Rysunek 4.2: Zrzut ekranu zawierający logi z jedno z węzłów sieci(Raspberry Pi Pico)



Rysunek 4.3: Zdjęcie węzła sieci opartego TTGO LoRa32



Rysunek 4.4: Zdjęcie stacji przekaźnikowej opartej o TTGO LoRa32

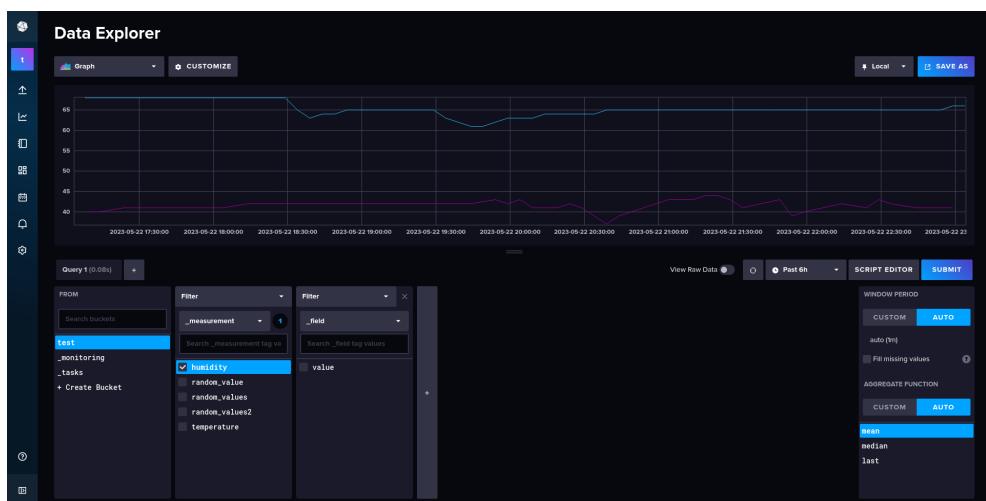
zostały uruchomione w jednej sieci Dockera, w celu zapewnienia komunikacji między nimi.

4.1.4 Broker wiadomości

Broker wiadomości został uruchomiony na wspólnym serwerze wewnętrz budynku. Broker został zabezpieczony hasłem, które zostało zapisane w pliku konfiguracyjnym. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z brokerem za pomocą tego hasła. Niestety, nie zostało użyte szyfrowanie TLS.

4.1.5 Baza danych i program zapisujący dane

Baza danych wraz z programem zapisującym dane zostały uruchomione na wspólnym serwerze wewnętrz budynku. Baza danych została zabezpieczona tokenem dostępu, który został zapisany w pliku konfiguracyjnym. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z bazą danych za pomocą tego tokenu. Niestety, nie zostało użyte szyfrowanie TLS. Zarówno baza danych, jak i program zapisujący dane zostały uruchomione w kontenerach Dockera, w celu zapewnienia izolacji i łatwości konfiguracji. Na rysunku 4.5 został przedstawiony zrzut ekranu interfejsu użytkownika InfluxDB 2. Na rysunku 4.6 został przedstawiony zrzut ekranu zawierający logi programu zapisującego pomiary do bazy InfluxDB 2.



Rysunek 4.5: Zrzut ekranu interfejsu użytkownika InfluxDB 2

4.1.6 Redis

Redis został uruchomiony na wspólnym serwerze wewnątrz budynku. Redis pozostał niezabezpieczony, ponieważ został udostępniony w sieci wewnętrznej. Wszystkie urządzenia zostały skonfigurowane tak, aby połączyć się z Redisem za pomocą tego hasła. Niestety, nie zostało użyte szyfrowanie TLS. Redis został uruchomiony w kontenerze Dockera, w celu zapewnienia izolacji i łatwości konfiguracji.

4.2 Testy

System został przetestowany pod kątem poprawności działania, wydajności oraz niezawodności. Testy zostały przeprowadzone w dniach od 1 do 14 maja 2023 roku.

4.2.1 Poprawność działania

Poprawność działania była weryfikowana na kilka sposobów:

Weryfikacja poprawności działania urządzeń pomiarowych

Weryfikacja poprawności działania urządzeń pomiarowych została przeprowadzona dla odczytów temperatury. Weryfikacja polegała na porównaniu odczytów z dwóch różnych czujników. Porównania dokonano pomiędzy czujnikiem DHT11 a czujnikiem DS18B20. Odczyty zostały zapisane do pliku CSV a następnie porównane. Przykładowe wyniki testów zostały przedstawione na rysunku 4.7.

Jak widać na załączonym rysunku, wyniki odczytów są bardzo zbliżone(maksymalna różnica pomiarów to 1°C), co świadczy o poprawności działania urządzeń pomiarowych.

```

2023-05-22 21:08:45 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 9, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:46 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 8, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:46 INFO Message already processed
2023-05-22 21:08:47 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 7, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:47 INFO Message already processed
2023-05-22 21:08:47 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 6, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:47 INFO Message already processed
2023-05-22 21:08:48 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 5, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:48 INFO Message already processed
2023-05-22 21:08:49 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 4, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:49 INFO Message already processed
2023-05-22 21:08:50 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 3, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:50 INFO Message already processed
2023-05-22 21:08:51 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 2, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:51 INFO Message already processed
2023-05-22 21:08:51 INFO Received message: {"values": {"random_values2": 92, "random_values": 53}, "ttl": 1, "m_id": "9b85e246-919f-4b08-9008-b514d43b9d97", "d_id": "ID44eab125754c"}
2023-05-22 21:08:51 INFO Message already processed
2023-05-22 21:08:53 WARNING Received message: N<-R;Y: u6&Mq;Y JR
+61Krh30VfkWgrJp (N;r)YC
dr_0a-$`Y
y-OSBUV/*uK/\eje0e\n2
ULV'ID7YY|i5|i?"M=-$~-f
>zx |4
2023-05-22 21:08:53 ERROR Invalid json message
2023-05-22 21:08:54 INFO Received message: {"values": {"humidity": 40, "temperature": 22}, "m_id": "146a7a28-aec6-451d-b2b0-97a33015f6fc", "d_id": "ID_e6614184038c5938"}
2023-05-22 21:08:55 ERROR Invalid json message
t", "values": {"humidity": 40, "temperature": 22}

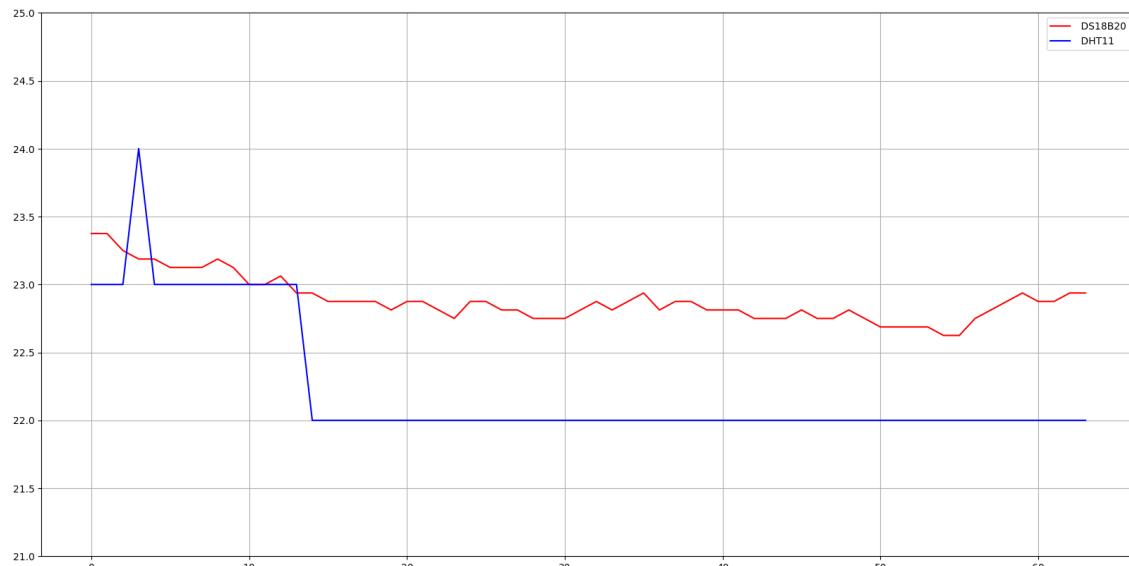
```

Rysunek 4.6: Zrzut ekranu zawierający logi programu zapisującego pomiary do bazy InfluxDB 2

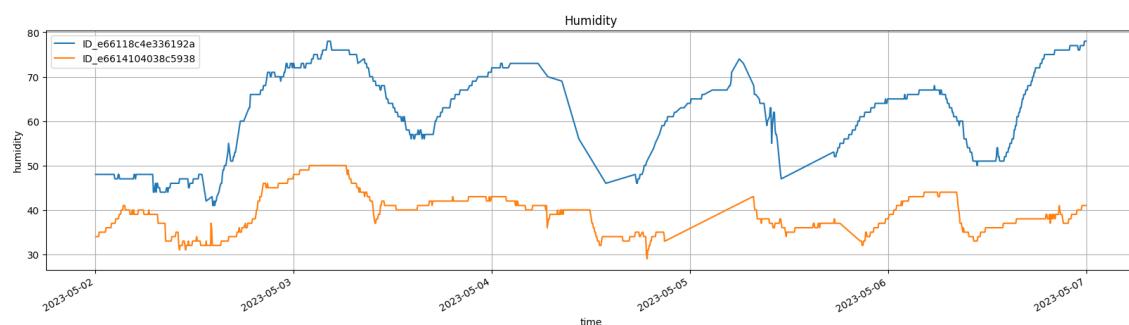
Weryfikacja poprawności działania całego systemu

Weryfikacja poprawności działania całego systemu została przeprowadzona poprzez porównywanie pakietów wysyłanych przez urządzenia pomiarowe z pakietami zapisanymi w bazie danych. Weryfikacja polegała na porównaniu wszystkich pól pakietów. Porównanie to odbywało się ręcznie i dotyczyło tylko kilku pakietów. Dodatkowo weryfikowano poprawność działania poprzez sprawdzenie zmian we wskazaniach czujników. Przejścia dla temperatury i wilgotności były bardzo płynne,(co widać na rysunku) co świadczy o poprawności działania systemu. Przejścia dla liczb losowych były bardzo chaotyczne, co również świadczy o poprawności działania systemu. Przykładowe wyniki testów zostały przedstawione na rysunku 4.8.

Dzięki wyżej wymienionym testom, udało się stwierdzić, że system działa poprawnie.



Rysunek 4.7: Wykres temperatury od czasu dla czujników DHT11 i DS18B20



Rysunek 4.8: Wykres wilgotności od czasu(w okresie od 2 Maja 2023 do 7 Maja 2023)

4.2.2 Wydajność

Testy wydajności zostały przeprowadzone w celu sprawdzenia, czy system jest w stanie obsłużyć wszystkie pakiety wysyłane przez urządzenia pomiarowe. Testy zostały przeprowadzone w dwóch etapach:

- Testy wydajności stacji przekaźnikowej
 - Testy wydajności węzłów sieci

Testy wydajności stacji przekaźnikowej

Testy zostały przeprowadzone w następujący sposób: z 3 węzłów sieci były emitowane pakiety w bardzo krótkich odstępach czasu (co 10 sekund). Wszystkie pakiety były wysyłane do stacji przekaźnikowej. Następnie przekazywała ona te pakiety do brokera wiadomości. Wszystkie pakiety były zapisywane w bazie danych.

W wyniku testów zostało stwierdzone, że stacja przekaźnikowa jest w stanie obsługiwać wszystkie pakiety wysyłane przez węzły sieci poprawnie.

Testy wydajności węzłów sieci

Testy zostały przeprowadzone w następujący sposób: z 3 węzłów sieci były emitowane pakiety w bardzo krótkich odstępach czasu(10 s). Weryfikacji podlegało to, czy wszystkie pakiety zostały wysłane ponownie rozgłoszone przez węzły sieci. Wszystkie pakiety były odbierane równie przez stację przekaźnikową.

W wyniku testów zostało stwierdzone, że węzły sieci są w stanie obsłużyć wszystkie pakiety wysyłane przez sieci poprawnie. Przykładowe wyniki testów zostały przedstawione na rysunku 4.9.

Rysunek 4.9: Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci

4.2.3 Zauważone problemy

W trakcie testów zostały zauważone następujące problemy:

Błędne pakiety

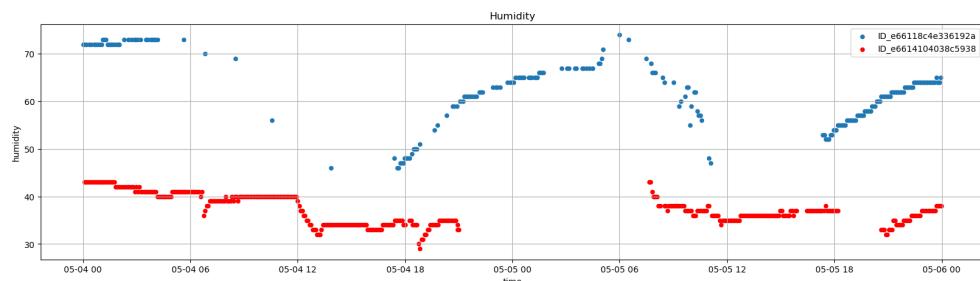
- Węzły sieci czasami odbierały błędne pakiety, jak na rysunku 4.10. Problem ten został rozwiązywany przez sprawdzanie czy odebrany pakiet jest poprawny, a w przypadku błędu, pakiet jest odrzucany.
- Przekaźnik sieci czasami przekazuje błędne pakiety do brokera(jak na rysunku 4.10). Problem ten został rozwiązywany przez sprawdzanie czy odebrany pakiet jest poprawny na poziomie programu zapisującego do bazy danych, a w przypadku błędu, pakiet jest odrzucany. Można byłoby sprawdzać poprawność pakietu na poziomie przekaźnika, ale obniżyłoby to wydajność systemu.

Błąd ludzki

Jak łatwo zauważać na wykresie 4.11 niestety w bazie danych pojawiają się luki w zapisanych danych. Takie luki spowodowane są brakami zasilania spowodowane przez osoby obsługujące ten system w czasie testów. Niektóre czujniki traciły zasilanie przez nieuwazne zmiany kabli zasilających czy nie zmienianie baterii na czas. Jednym z możliwych rozwiązań jest zastosowanie większej baterii w każdym węźle sieci.

```
2023-05-06 11:45:25 INFO    Message already processed
2023-05-06 11:46:45 WARNING  Received message: 23RXF2W>kuxkU>RZ#
2023-05-06 11:46:45 ERROR   Invalid json message
```

Rysunek 4.10: Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci



Rysunek 4.11: Szczegółowych diagram danych zapisanych w bazie między 4 Maja, a 6 Maja 2023

Rozdział 5

Perspektywy rozwoju i dalsze badania

5.1 Rozwój projektu

W trakcie pracy nad projektem udało się zrealizować wszystkie założenia projektowe. Wszystkie urządzenia zostały zaprogramowane i skonfigurowane, a system zbierania danych został zaimplementowany. Wszystkie urządzenia zostały przetestowane i działają poprawnie. Jednak w przyszłości można rozwinąć projekt o kilka funkcjonalności i usprawnień, które mogą zwiększyć użyteczność systemu.

5.1.1 Optymalizacja energetyczna węzłów sieci

Zmniejszenie częstotliwości wysyłania pakietów

Implementacja węzłów sieci w obecnej wersji skupia się na stabilności i poprawności działania. Jednak w przyszłości można rozwinąć projekt o funkcjonalność optymalizacji energetycznej węzłów sieci. W obecnej wersji węzły sieci wysyłają pakiety w stałych odstępach czasu. Jednak w przypadku, gdy węzły sieci nie wykryją żadnych zmian w otoczeniu, nie ma potrzeby wysyłania pakietów. W takiej sytuacji można zastosować algorytm, który będzie sprawdzał, czy w otoczeniu węzła sieci występują jakieś zmiany. Jeśli nie, to węzeł sieci będzie wysyłał pakiety w dłuższych odstępach czasu. Dzięki temu można zaoszczędzić energię węzłów sieci.

Uśpienie węzłów sieci

Dodatkowo wiele mikrokontrolerów posiada funkcjonalność głębokiego uśpienia. W takim trybie mikrokontroler zużywa bardzo mało energii. Jednak w takim trybie mikrokontroler nie jest w stanie wykonywać żadnych operacji. Należałyby rozpoznać czy tryby uśpienia zaproponowanych przez producentów mikrokontrolerów są wystarczające do zastosowania w projekcie. Jeśli tak, to można zastosować tryby uśpienia

w węzłach sieci, które nie będą wysyłały pakietów przez dłuższy czas.

Większa bateria

By wydłużyć czas działania węzłów sieci można również zastosować większe baterie. Wydłużyło by to czas pracy na baterii.

Zastosowanie innych technologii

W celu optymalizacji energetycznej węzłów sieci można również zastosować inne technologie do oprogramowania węzłów sieci. W obecnej wersji węzły sieci niektóre węzły sieci zostały zaprogramowane w języku MicroPython, a inne w języku C++. W przyszłości można przepisać wszystkie węzły sieci na jeden język programowania. W takim przypadku można zastosować język C++, który jest bardziej wydajny od języka MicroPython. Dzięki temu można zwiększyć wydajność węzłów sieci przy jednoczesnej zwiększonej wydajności energetycznej.

5.1.2 Bezpieczeństwo

Autoryzacja węzłów sieci

W obecnej wersji systemu węzły sieci nie są autoryzowane. W takiej sytuacji każdy węzeł sieci może dołączyć do sieci i wysyłać pakiety. Jednak w przyszłości można rozwinać projekt o funkcjonalność autoryzacji węzłów sieci. W takiej sytuacji węzły sieci będą musiały autoryzować się przed dołączeniem do sieci. Dzięki temu można zwiększyć bezpieczeństwo systemu.

Szyfrowanie wiadomości

W obecnej wersji systemu wiadomości wysyłane przez węzły sieci nie są szyfrowane. Biorąc pod uwagę specyfikę protokołu LoRa wiadomości są rozgłaszcane w formie broadcastu, więc mogą zostać odebrane przez dowolne urządzenie. Jednak w przyszłości można rozwinać projekt o funkcjonalność szyfrowania wiadomości. W takiej sytuacji węzły sieci będą szyfrowały wiadomości przed wysłaniem ich do stacji przekaźnikowej. Następnie stacja przekaźnikowa będzie odszyfrowywała wiadomości przed przekazaniem ich do brokeru wiadomości. Dzięki temu można zwiększyć bezpieczeństwo systemu.

Należałyby rozpoznać, czy szyfrowanie wiadomości nie spowoduje zbyt dużego spadku wydajności systemu. W takim przypadku można zastosować szyfrowanie wiadomości tylko w sytuacji, gdy w wiadomości znajdują się dane wrażliwe.

Należały również sprawdzić jaka forma szyfrowania jest najbardziej odpowiednia do zastosowania w projekcie. W tym celu można przeprowadzić badania porównujące różne formy i algorytmy szyfrowania pod względem wydajności i bezpieczeństwa.

Jeżeli zdecydowano by się na implementację szyfrowania z wykorzystaniem klucza, to należały rozpoznać, jak klucz będzie przekazywany do węzłów sieci. W tym celu można zastosować algorytm wymiany klucza Diffiego-Hellmana. Dzięki temu klucz szyfrowania będzie przekazywany bezpiecznie.

W celu szyfrowania wiadomości należały również rozpoznać, jakie algorytmy szyfrowania są dostępne w mikrokontrolerach. W tym celu można przeprowadzić badania porównujące różne mikrokontrolery pod względem dostępnych algorytmów szyfrowania.

5.1.3 Komunikacja dwukierunkowa

W obecnej wersji systemu komunikacja między węzłami sieci a stacją przekaźnikową jest jednokierunkowa. Jednak w przyszłości można rozwinać projekt o funkcjonalność komunikacji dwukierunkowej. W takiej sytuacji węzły sieci będą mogły odbierać wiadomości od stacji przekaźnikowej. Dzięki temu można zwiększyć użyteczność systemu, chociażby poprzez możliwość zdalnego sterowania węzłami sieci.

5.1.4 Przygotowanie nowych węzłów sieci

W obecnej wersji systemu została przygotowana implementacja węzłów sieci, która jest kompatybilna z platformami opartymi o Arduino lub MicroPython. Jednak w przyszłości można rozwinać projekt o funkcjonalność przygotowania nowych węzłów sieci. W takiej sytuacji można przygotować implementację węzłów sieci, która będzie kompatybilna z innymi platformami (takimi jak STM32). Dzięki temu można zwiększyć uniwersalność systemu.

5.2 Dalsze badania

5.2.1 Badania wydajnościowe dużej sieci

W trakcie pracy nad projektem zostały przeprowadzone testy wydajnościowe małej sieci (3 węzły). Jednak powinny zostać przeprowadzone badania wydajnościowe dużej sieci. W takiej sytuacji można przetestować wydajność systemu w sytuacji, gdy w sieci znajduje się dużo węzłów sieci na znacznej powierzchni. Dzięki temu można sprawdzić, czy system jest w stanie obsłużyć dużą liczbę węzłów.

5.2.2 Badania konsumpcji energii

W trakcie przyszłych prac nad projektem należałoby przeprowadzić badania konsumpcji energii węzłów sieci. W takiej sytuacji można sprawdzić, jak długo węzły sieci mogą działać na baterii. Należałoby również zoptymalizować zużycie energii węzłów sieci. Dzięki temu można zwiększyć czas działania węzłów sieci na zasilaniu baterijnym.

5.2.3 Badania zasięgu sieci

W trakcie przyszłych prac nad projektem należałoby przeprowadzić badania zasięgu sieci. W takiej sytuacji można sprawdzić, jak daleko od siebie mogą znajdować się węzły sieci. Dzięki temu można sprawdzić, jakie są ograniczenia zasięgu sieci.

Podsumowanie

Spis listingów

3.1 Przykładowa wiadomość przesyłana przez system	24
---	----

Spis rysunków

1.1	Interfejs narzędzia PlatformIO w edytorze Visual Studio Code	15
3.1	Diagram prezentujący schemat ideowy systemu	23
4.1	Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią	30
4.2	Zrzut ekranu zawierający logi z jedno z węzłów sieci(Raspberry Pi Pico)	30
4.3	Zdjęcie węzła sieci opartego TTGO LoRa32	31
4.4	Zdjęcie stacji przekaźnikowej opartej o TTGO LoRa32	31
4.5	Zrzut ekranu interfejsu użytkownika InfluxDB 2	32
4.6	Zrzut ekranu zawierający logi programu zapisującego pomiary do bazy InfluxDB 2	33
4.7	Wykres temperatury od czasu dla czujników DHT11 i DS18B20	34
4.8	Wykres wilgotności od czasu(w okresie od 2 Maja 2023 do 7 Maja 2023)	34
4.9	Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci	35
4.10	Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci	36
4.11	Szczegółowych diagram danych zapisanych w bazie między 4 Maja, a 6 Maja 2023	36

Bibliografia

- [1] ietf. *A Universally Unique IDentifier (UUID) URN Namespace*. 2005. (Term. wiz. 23.04.2023).
- [2] Huang-Chen Lee i Kai-Hsiang Ke. “Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation”. W: *IEEE Transactions on Instrumentation and Measurement* 67.9 (2018), s. 2177–2187. DOI: 10.1109/TIM.2018.2814082.
- [3] Ramon Sanchez-Iborra i in. “Performance Evaluation of LoRa Considering Scenario Conditions”. W: *Sensors* 18.3 (2018). ISSN: 1424-8220. DOI: 10.3390/s18030772. URL: <https://www.mdpi.com/1424-8220/18/3/772>.
- [4] mqtt.org. *MQTT*. 2019. URL: <https://mqtt.org/mqtt-specification/> (term. wiz. 23.04.2023).
- [5] Semtech. *SX1261/2 Low Power Long Range Transceiver*. 2019. URL: https://www.waveshare.com/w/upload/e/e1/DS_SX1261-2_V1.2.pdf (term. wiz. 16.04.2023).
- [6] Waveshare. *Waveshare SX1262 LoRa Node Module 868MHz*. 2019. URL: <https://www.waveshare.com/wiki/Pico-LoRa-SX1262> (term. wiz. 16.04.2023).
- [7] knolleary. *pubsubclient*. 2020. URL: <https://github.com/knolleary/pubsubclient> (term. wiz. 16.04.2023).
- [8] Semtech. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver*. 2020. URL: https://semtech.my.salesforce.com/sfc/p/#E0000000Je1G/a/2R0000001Rbr/6EfVZUorrpoKFfvaF_Fkpgp5kzjiNyiaBqcpqh9qSjE (term. wiz. 16.04.2023).
- [9] Riccardo Berto, Paolo Napoletano i Marco Savi. “A LoRa-Based Mesh Network for Peer-to-Peer Long-Range Communication”. W: *Sensors* 21.13 (2021). ISSN: 1424-8220. DOI: 10.3390/s21134314. URL: <https://www.mdpi.com/1424-8220/21/13/4314>.
- [10] Joan Miquel Solé i in. “Implementation of a LoRa Mesh Library”. W: *IEEE Access* 10 (2022), s. 113158–113171. DOI: 10.1109/ACCESS.2022.3217215.

- [11] adafruit. *Adafruit_SSD1306*. 2023. URL: https://github.com/adafruit/Adafruit_SSD1306 (term. wiz. 16.04.2023).
- [12] arduino-LoRa. 2023. URL: <https://github.com/sandeepmistry/arduino-LoRa> (term. wiz. 16.04.2023).
- [13] ArduinoJson. 2023. URL: <https://github.com/bblanchon/ArduinoJson> (term. wiz. 16.04.2023).
- [14] Docker. *Docker Documentation*. 2023. URL: <https://docs.docker.com/> (term. wiz. 23.04.2023).
- [15] Eclipse. *Paho MQTT Python Client*. 2023. URL: <https://www.eclipse.org/paho/> (term. wiz. 23.04.2023).
- [16] ESPRandom. 2023. URL: <https://github.com/moritz89/ESPRandom> (term. wiz. 16.04.2023).
- [17] espressif. *arduino-esp32*. 2023. URL: <https://github.com/espressif/arduino-esp32> (term. wiz. 16.04.2023).
- [18] Raspberry Pi Foundation. *Raspberry Pi Documentation*. 2023. URL: <https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html> (term. wiz. 16.04.2023).
- [19] Raspberry Pi Foundation. *Raspberry Pi Pico Datasheet*. 2023. URL: <https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf> (term. wiz. 16.04.2023).
- [20] InfluxData. *InfluxDB Documentation*. 2023. URL: <https://docs.influxdata.com/influxdb/> (term. wiz. 23.04.2023).
- [21] influxdata. *InfluxDB Python client library*. 2023. URL: <https://docs.influxdata.com/influxdb/v2.7/api-guide/client-libraries/python/> (term. wiz. 23.04.2023).
- [22] micropySX126X. 2023. URL: <https://github.com/ehong-tl/micropySX126X> (term. wiz. 16.04.2023).
- [23] MicroPython. 2023. URL: <https://micropython.org/> (term. wiz. 16.04.2023).
- [24] Eclipse Mosquitto. *Eclipse Mosquitto*. 2023. URL: <https://mosquitto.org/> (term. wiz. 23.04.2023).
- [25] PlatformIO. *PlatformIO Documentation*. 2023. URL: <https://docs.platformio.org/en/latest/> (term. wiz. 23.04.2023).
- [26] Redis. *redis-py*. 2023. URL: <https://redis.io/docs/clients/python/> (term. wiz. 23.04.2023).

- [27] Espressif Systems. *ESP32 Datasheet*. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (term. wiz. 16.04.2023).
- [28] Espressif Systems. *ESP32 SoCs*. 2023. URL: <https://www.espressif.com/en/products/socs> (term. wiz. 16.04.2023).
- [29] LoRa. *LoRa Documentation*. URL: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan> (term. wiz. 23.04.2023).