



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 **dra hab. Przemysława Stpiczyńskiego**

Lublin 2023

Spis treści

Wstęp	5
1 Wykorzystane narzędzia, technologie i protokoły	7
1.1 Technologia LoRa	7
1.2 Urządzenia wykorzystywane w projekcie	8
1.2.1 ESP32	8
1.2.2 Raspberry Pi Pico	9
1.3 MicroPython dla Raspberry Pi Pico	9
1.4 Protokoły komunikacyjne: MQTT	10
1.5 Bazy danych i pozostałe technologie	10
1.5.1 InfluxDB 2	10
1.5.2 Docker	10
1.5.3 PlatformIO	11
2 Istniejące rozwiązania	13
2.1 LoRaWAN	13
2.2 Biblioteka LoRaMesher	13
2.3 System monitorowania dużej powierzchni z wykorzystaniem LoRa, oparty o architekturę typu mesh	14
2.4 Sieć mesh oparta o LoRa do komunikacji typu <i>peer-to-peer</i>	15
3 Założenia i implementacja	17
3.1 Założenia projektu	17
3.2 Części sieci	17
3.3 Wiadomości	18
3.4 Obsługa protokołu	20
3.5 Implementacja	21
4 Wdrożenie i testy	25
4.1 Wdrożenie	25

4.2	Testy	29
4.2.1	Poprawność działania	29
4.2.2	Wydajność	31
4.3	Zauważone problemy	32
5	Perspektywy rozwoju i dalsze badania	33
5.1	Rozwój projektu	33
5.1.1	Optymalizacja energetyczna węzłów sieci	33
5.1.2	Bezpieczeństwo	34
5.1.3	Komunikacja dwukierunkowa	35
5.1.4	Przygotowanie nowych węzłów sieci	35
5.2	Dalsze badania	35
5.2.1	Badania wydajnościowe dużej sieci	35
5.2.2	Badania zużycia energii	36
5.2.3	Badania zasięgu sieci	36
Podsumowanie		37
Spis listingów		39
Spis rysunków		41
Bibliografia		45

Wstęp

W dzisiejszych czasach ważne jest zbieranie różnorodnych informacji, z różnych, często trudno dostępnych miejsc. W tym celu wykorzystywane są urządzenia, które zbierają informacje o otoczeniu, a następnie przekazują je do systemu, który następnie je przetwarza. Tego rodzaju czujniki mogą być umieszczone w trudno dostępnych miejscach, na przykład na wysokich budynkach czy w głębokich studniach. Nie ma tam możliwości poprowadzenia kabli, dlatego konieczne jest zastosowanie bezprzewodowych technologii komunikacyjnych. Jedną z takich technologii jest LoRa, która pozwala na komunikację na duże odległości, przy niskim poborze mocy.

Celem niniejszej pracy jest stworzenie systemu zbierającego dane z rozproszonych czujników z wykorzystaniem technologii LoRa oraz ich zapisywanie w bazie danych w celu dalszego przetwarzania. W ramach pracy został stworzony system zbierający dane z różnych czujników, takich jak termometry czy higrometry. Informacje są zbierane przez autorskie punkty dostępowe LoRa, które następnie przekazują je do serwera. Serwer przechowuje dane w bazie danych, która została wybrana do zastosowań IoT.

W rozdziale pierwszym zostały opisane technologie, protokoły i urządzenia wykorzystane w projekcie. W kolejnym rozdziale przedstawiono istniejące rozwiązania, zarówno przemysłowych jak i naukowych. Zostały one również porównane z rozwiązaнием zaproponowanym w ramach pracy. W rozdziale trzecim zostały opisane założenia projektowe oraz szczegółowa implementacja systemu. W czwartym rozdziale został opisany proces wdrożenia systemu oraz przeprowadzone testy. W ramach testów została sprawdzona wydajność systemu. Zostały również przedstawione wnioski z testów. W piątym rozdziale zostały przedstawione wnioski z pracy oraz możliwości rozwoju systemu.

Rozdział 1

Wykorzystane narzędzia, technologie i protokoły

W tym rozdziale zostaną opisane narzędzia, technologie i protokoły wykorzystane w projekcie.

1.1 Technologia LoRa

LoRa (ang. *Long Range* — daleki zasięg) to protokół komunikacji bezprzewodowej, stworzony z myślą o urządzeniach IoT (ang. *Internet of Things* — Internet rzeczy). Przeznaczony jest do komunikacji na duże odległości i z niskim zużyciem energii. Został opracowany przez firmę 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 [30].

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

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

Technologia LoRa pozwala również na dostosowywanie parametrów transmisji, takich jak:

- częstotliwość,
- moc nadajnika,
- szybkość transmisji,
- szerokość pasma,
- spreading factor (*SF*) (ang. *współczynnik rozprzestrzeniania*).

Więcej na temat wydajności sieci można przeczytać w artykule [2].

1.2 Urządzenia wykorzystywane w projekcie

W tym podrozdziale zostaną opisane urządzenia wykorzystane w projekcie oraz ich najważniejsze cechy.

1.2.1 ESP32

ESP32 to jednoukładowy mikrokontroler, zaprojektowany i produkowany przez firmę Espressif Systems. Jego najważniejsze cechy to [28]:

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

Powstało wiele wersji tego układu, różniących się m.in. szybkością procesora, wielkością pamięci flash, ilością pinów, interfejsów cyfrowych i analogowych, a także możliwością pracy w trybie bezprzewodowym lub przewodowym [29]. Najczęściej układ te są wykorzystywane w różnych projektach IoT, zarówno jako czujniki, jak i serwery [28]. Na rysunku 1.1 przedstawiono schemat wyprowadzeń płytki LoRa32.

W projekcie ESP32 użyto w dwóch płytach TTGO T3 LoRa32 V1.6.1. Pierwsza z płyt została wykorzystana jako przekaźnik danych pomiędzy siecią LoRa, a siecią WiFi, a druga jako część systemu zbierania danych z wykorzystaniem LoRa.

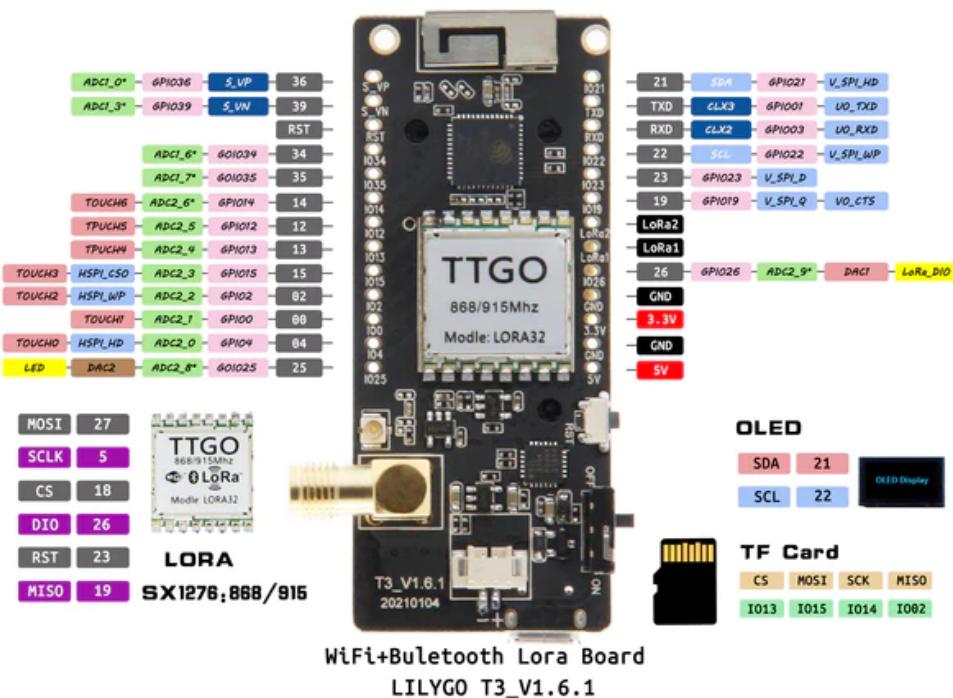
1.2.2 Raspberry Pi Pico

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. Płytnka posiada również wiele interfejsów cyfrowych i analogowych (takich jak: UART, I2C, SPI, I2S, ADC, DAC, PWM, USB 1.1) [19]. Jest ona często wykorzystywana zarówno przez hobbystów do różnych projektów IoT, a także jako sterownik silników, czy kontroler robotów, jak i jest szeroko używana w przemyśle [18]. Na rysunku 1.2 przedstawiono schemat wyprowadzeń płytka Raspberry Pi Pico.

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

1.3 MicroPython dla Raspberry Pi Pico

MicroPython to lekka i wydajna implementacja języka Python 3, która została zaprojektowana z myślą o urządzeniach wbudowanych. Zawiera on okrojoną wersję biblioteki standardowej Pythona. MicroPython jest dostępny na wiele platform, w tym na Raspberry Pi Pico [24].



Rysunek 1.1: Schemat wyprowadzeń TTGO T3 LoRa32 (źródło: <https://www.lilygo.cc/products/lora3>)

1.4 Protokoły komunikacyjne: MQTT

MQTT (ang. *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 [6].

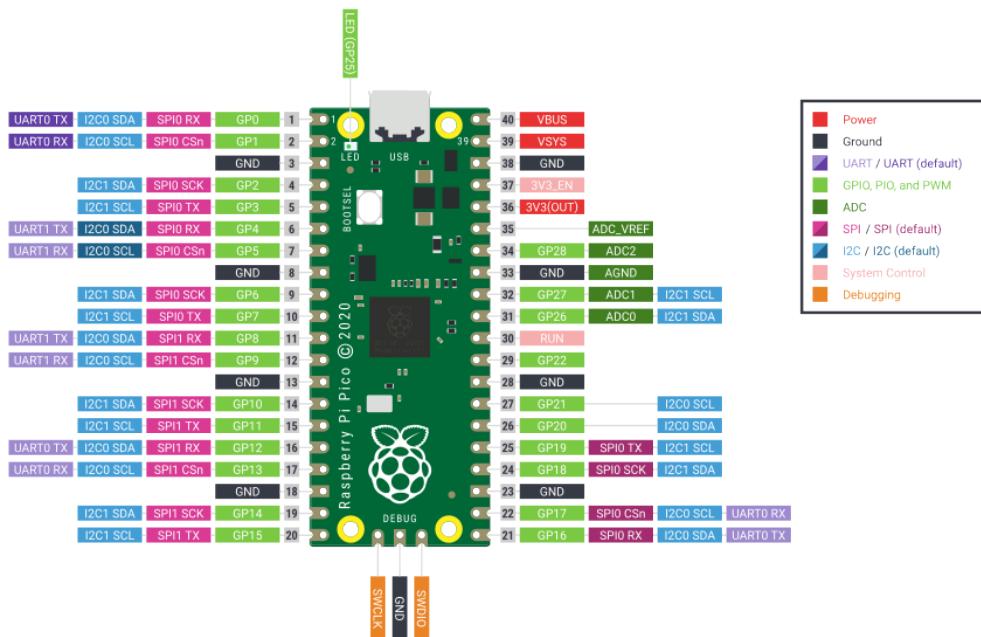
1.5 Bazy danych i pozostałe technologie

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, między innymi z 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

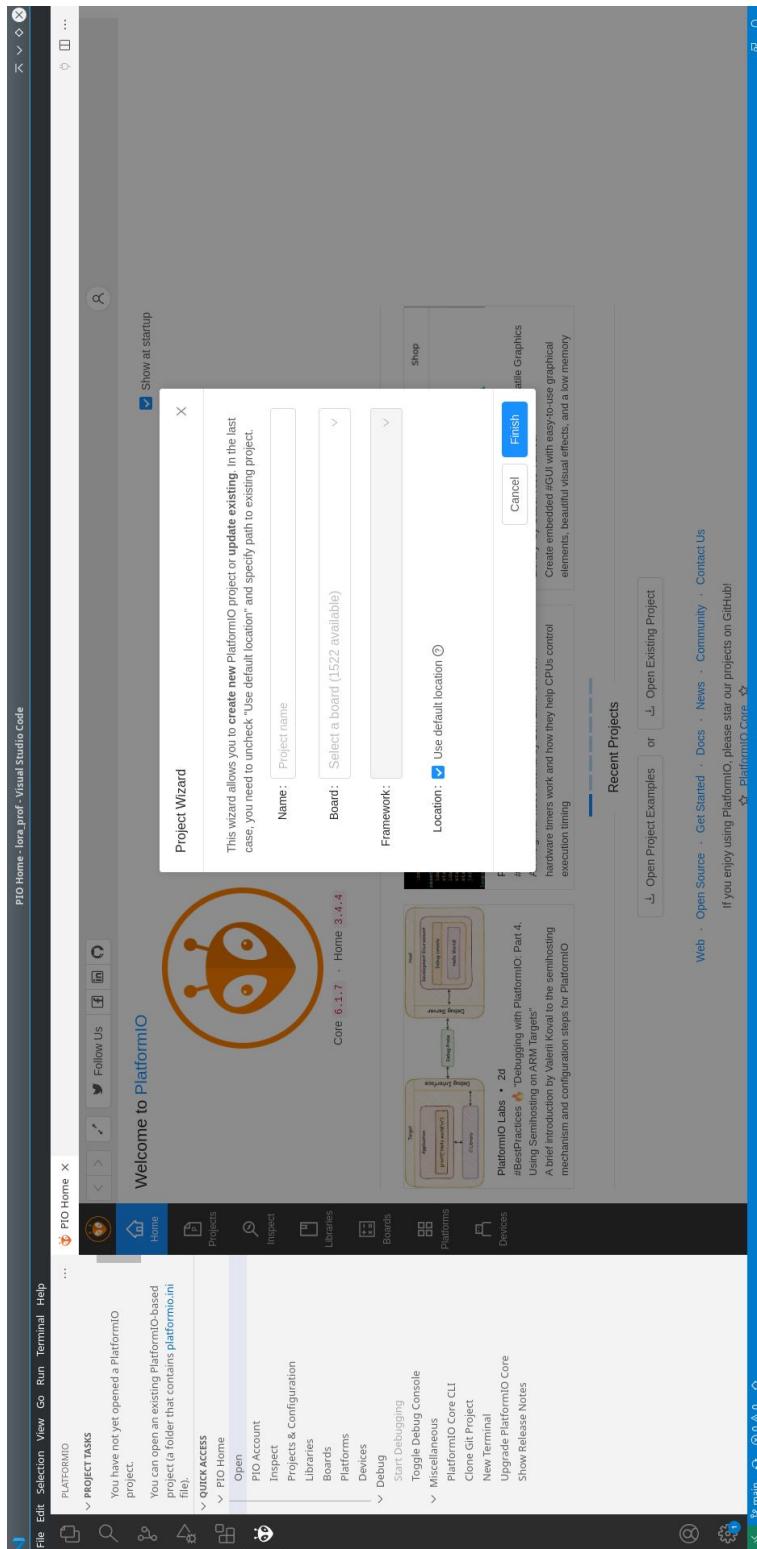


Rysunek 1.2: Schemat wyprowadzeń Raspberry Pi Pico (źródło: <https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html>)

systemu operacyjnego. Kontenery mniej obciążają pamięć i procesor 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 kontrolowania zależności projektu, łatwego dodawania bibliotek, 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 [26]. Na rysunku 1.3 przedstawiono interfejs narzędzia PlatformIO w edytorze Visual Studio Code.



Rysunek 1.3: Interfejs narzędzia PlatformIO w edytorze Visual Studio Code

Rozdział 2

Istniejące rozwiązania

W tym rozdziale zostaną przedstawione istniejące rozwiązania, które zostały zaproponowane w artykułach naukowych i rozwiązaniach komercyjnych, a które mogą być alternatywą dla rozwiązania zaproponowanego w projekcie.

2.1 LoRaWAN

LoRaWAN (ang. *LoRa Wide Area Network*) to protokół komunikacji stworzony przez LoRa Alliance [30]. 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.

Sieć oparta o protokół LoRaWAN jest siecią typu gwiazda. Przez to, że węzły sieci komunikują się tylko z bramami, a nie z innymi węzłami, sieć jest łatwa do zarządzania, ale utrudnia to skalowanie. Jeżeli brama osiągnie maksymalną liczbę węzłów, to aby dodać kolejne węzły, należy dodać kolejną bramę. Dodatkowo, jeżeli brama ulegnie awarii, to wszystkie węzły, które komunikowały się z danym punktem dostępowym, muszą nawiązać połączenie z innym, co może prowadzić do utraty danych. Same bramy są urządzeniami o dużej wydajności, co prowadzi do zwiększenia kosztów projektu.

2.2 Biblioteka LoRaMesher

Badacze w artykule zaproponowali otwartoźródłową implementację autorskiego protokołu LoRaMesher [4] 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 rutingowe 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 (szeregowo). W tym przypadku wskaźnik dostarczenia pakietów wynosił 96%.
- Sieć złożona z 10 węzłów, w tym 5 węzłów działało tylko jako przekaźniki. Architektura sieci symulowała użycie biblioteki w realnym zastosowaniu. W tym przypadku 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 zostało 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 wydłużyć czas 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.

2.3 System monitorowania dużej powierzchni z wykorzystaniem LoRa, oparty o architekturę typu mesh

Badacze w podanym artykule [1] zaproponowali kompletne rozwiązanie umożliwiające monitorowanie dużego obszaru kampusu uniwersyteckiego, w oparciu tylko o protokół LoRa. W odróżnieniu od pracy opisanej w sekcji 2.2 autorzy opisywanego tutaj artykułu 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 jedną stację bazową i 10 węzłów sieci. Dane były pobierane co 60 sekund. Wskaźnik dostarczalności pakietów uplasował się 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 komunikacje. 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 utraty danych.

Drugą znaczącą różnicą jest sposób komunikacji między węzłami sieci. W projekcie zastosowano rozwiązanie, w którym to stacja bazowa odpowiada za pobieranie danych od węzłów sieci. Jednak w takim rozwiązaniu stacja bazowa musi być w stanie obsługiwać wszystkie węzły sieci, co może być problematyczne w przypadku dużych sieci. Jeżeli stacja ulegnie awarii, tracone są dane z wszystkich węzłów sieci.

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 tychże.

2.4 Sieć mesh oparta o LoRa do komunikacji typu *peer-to-peer*

W artykule [3] 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 naszym 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 należałyby 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

Założenia i implementacja

W poniższym rozdziale zaprezentowano założenia, potrzeby i implementację systemu zbierania danych, opartego o technologię LoRa.

3.1 Założenia projektu

System ma kilka podstawowych założeń:

- jeden centralny punkt gromadzenia danych,
- zapisywanie danych do bazy danych szeregow 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,
- możliwie duża dostarczalność pakietów,
- dopuszczalne drobne opóźnienia w dostarczaniu danych.

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 systemu.

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 do bazy danych.

Stacja przekaźnikowa

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

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

Broker wiadomości

Broker wiadomości to program, działający na komputerze mającym dostęp do sieci, umożliwia on wydajną komunikację pomiędzy stacją przekaźnikową a bazą danych.

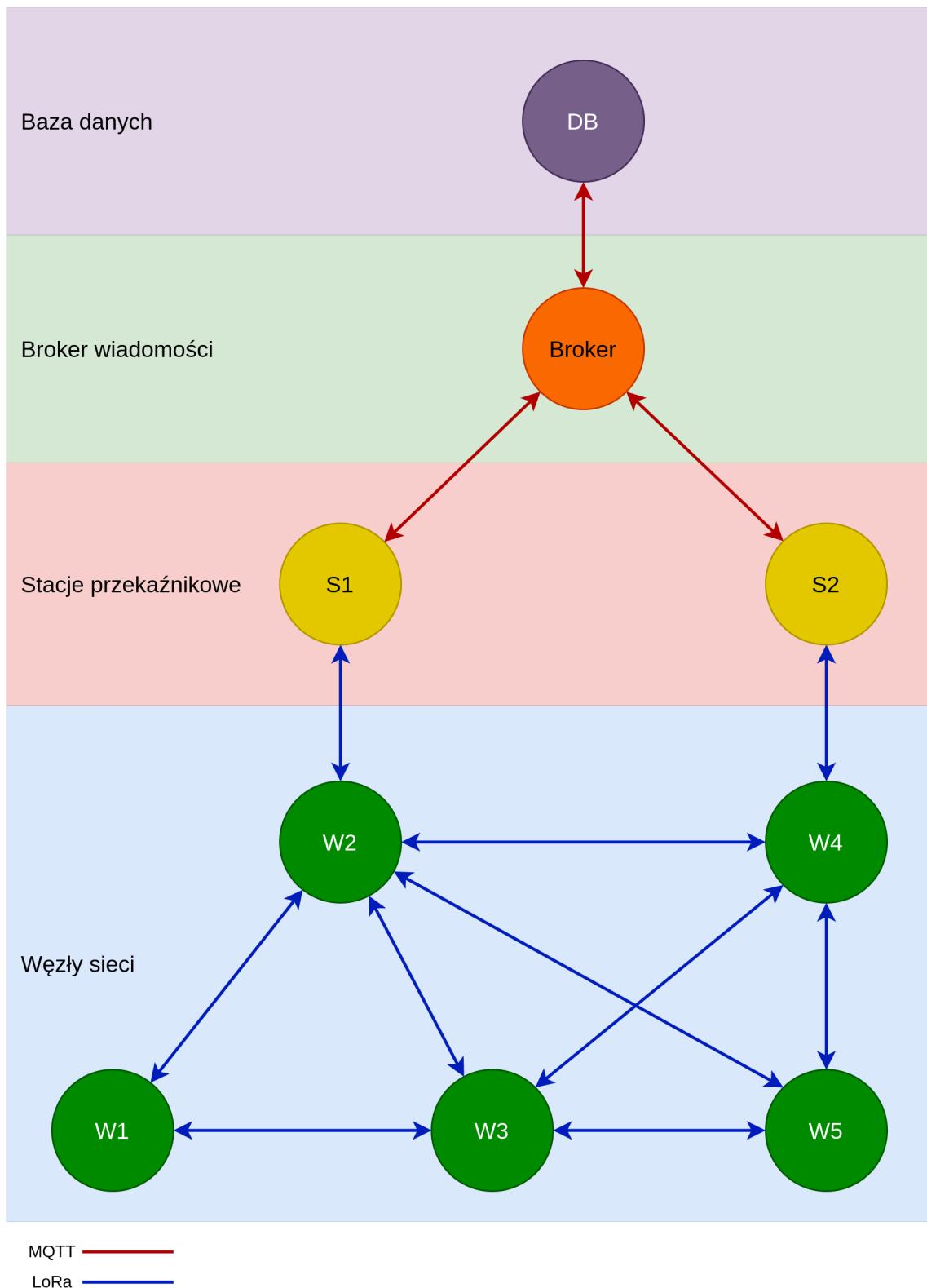
Baza danych

Baza danych umożliwia zapisywanie sporej ilości danych, uwzględniając również ich czas (baza danych szeregów czasowych). O zapis 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.

3.3 Wiadomości

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

Zawiera ona pola:



Rysunek 3.1: Diagram prezentujący schemat ideowy systemu

- **ttl** — (ang. *time to live* — czas życia) wartość określająca maksymalną liczbę skoków pomiędzy węzłami sieci. (domyślnie wynosi 10, może zostać wydłużona w zależności od wielkości planowanej sieci);
- **m_id** — UUID [5] 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      "ttl": 10,
3      "d_id": "id_233",
4      "m_id": "eaa17a7b-9388-43b6-9310-731c942fc6b9"
5      "values": {
6          "temp": 21,
7          "hum": 50,
8          "press": 1000,
9          "light": 100,
10         "co2": 1000,
11         "pm25": 10,
12         "pm10": 20
13     },
14 }
```

Listing 3.1: Przykładowa wiadomość przesyłana przez system

3.4 Obsługa protokołu

Działanie węzłów

Wiadomości generowane są przez węzły sieci, zawierając wszystkie niezbędne pola (wymienione wyżej) i odczyty z czujników zamieszczonych w węźle. Następnie zostaje ona rozesłana do wszystkich węzłów w zasięgu (ang. *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 w celu wyeliminowania wiadomości niepoprawnych z sieci.

Działanie stacji przekaźnikowej

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

Działanie bazy danych i programu zapisującego dane

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

1. sprawdzenie poprawności wiadomości;
2. upewnienie się, 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 `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

Implementacja węzłów sieci

W wyniku pracy nad systemem zbierania danych przygotowano implementację węzłów sieci z wykorzystaniem dwóch 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 [7] (z wykorzystaniem płytka rozwojowej Waveshare SX1262 LoRa Node Module [8]) oraz czujnik temperatury i wilgotności DHT11. Urządzenie zostało zaprogramowane w języku MicroPython z wykorzystaniem biblioteki `micropySX126X` [23]. 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 T3 LoRa32

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [10]. Zostały one zaprogramowane w języku C++ z użyciem PlatformIO — ekosystemu do programowania urządzeń IoT. [26]. W programie zostały wykorzystane biblioteki:

- `arduino-LoRa` — biblioteka do obsługi modułu LoRa [12];
- `ArduinoJson` — biblioteka do obsługi formatu JSON [13];
- `Adafruit-SSD1306` — biblioteka do obsługi wyświetlacza OLED [11];
- `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.

Implementacja stacji przekaźnikowej

W wyniku pracy nad systemem zostało przygotowane jedno urządzenie oparte o płytę TTGO LoRa32. Urządzenie zostało wyposażone w moduł LoRa SX1276 [10] oraz moduł WiFi ESP32 [28]. Urządzenie zostało zaprogramowane w języku C++ z wykorzystaniem PlatformIO [26]. W programie zostały 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 [9].

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.

Implementacja brokera wiadomości

Aby zapewnić niezawodność i wysoką wydajność komunikacji zdecydowano użyć otwartoźródłowego brokera wiadomości Mosquitto [25]. Jest to sprawdzony, wydajny i niezawodny program, który jest szeroko stosowany w systemach IoT, również w zastosowaniach profesjonalnych [25].

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].

Redis

W celu przechowywania identyfikatorów wiadomości, które zostały już sprawdzone, zdecydowano użyć bazy danych Redis. Redis jest szybką, otwartoźródłową, niereadyjną bazą danych typu *klucz-wartość*, która jest jedną z najczęściej stosowanych w systemach informatycznych [22].

Program zapisujący dane

W celu zapisywania danych zdecydowano użyć programu napisanego w języku Python. Program został napisany 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 [27].

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

Rozdział 4

Wdrożenie i testy

W tym rozdziale zostanie opisane wdrożenie systemu oraz przeprowadzone testy.

4.1 Wdrożenie

System został wdrożony 1 maja 2023 roku. Wdrożenie polegało na uruchomieniu serwera z bazą danych, jednej stacji przekaźnikowej oraz uruchomieniu trzech urządzeń pomiarowych.

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 (Raspberry Pi Pico). Urządzenia zostały połączone z czujnikami za pomocą zwykłych kabli prototypowych, jak na rysunku 4.1. Przykład urządzenia wykorzystującego wbudowane czujniki został zaprezentowany na rysunku 4.3.

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 rysunku 4.4.

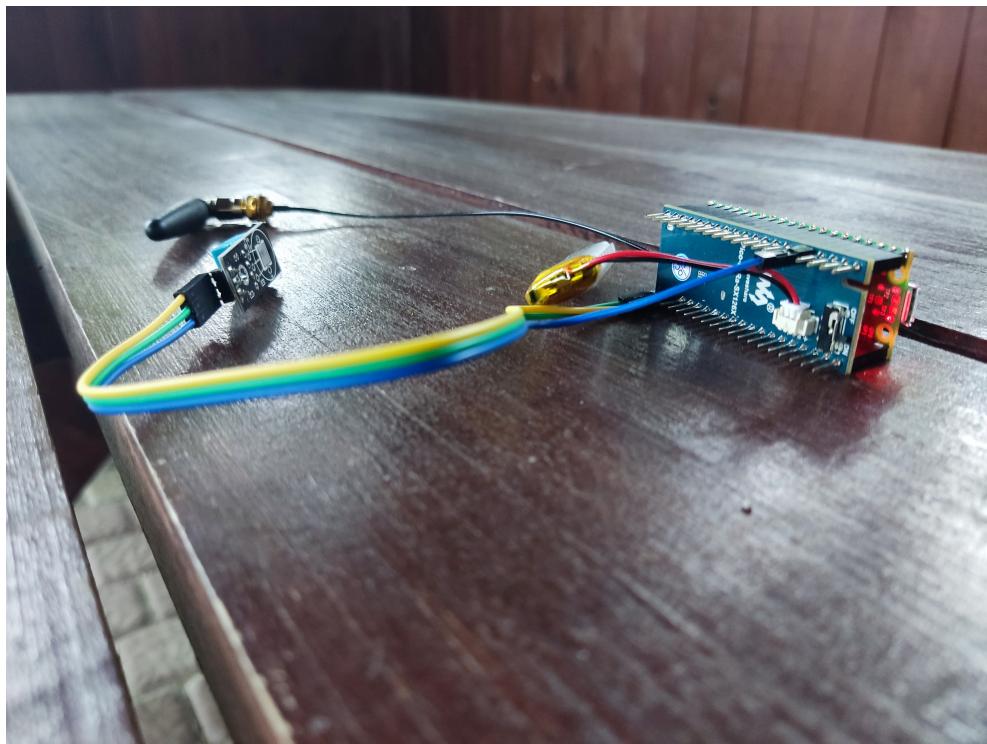
Serwer testowy

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

- Mosquitto (patrz str. 23)

- InfluxDB 2 (patrz str. 23)
- Program zapisujący do InfluxDB 2 (patrz str. 23)
- Redis (patrz str. 23)

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



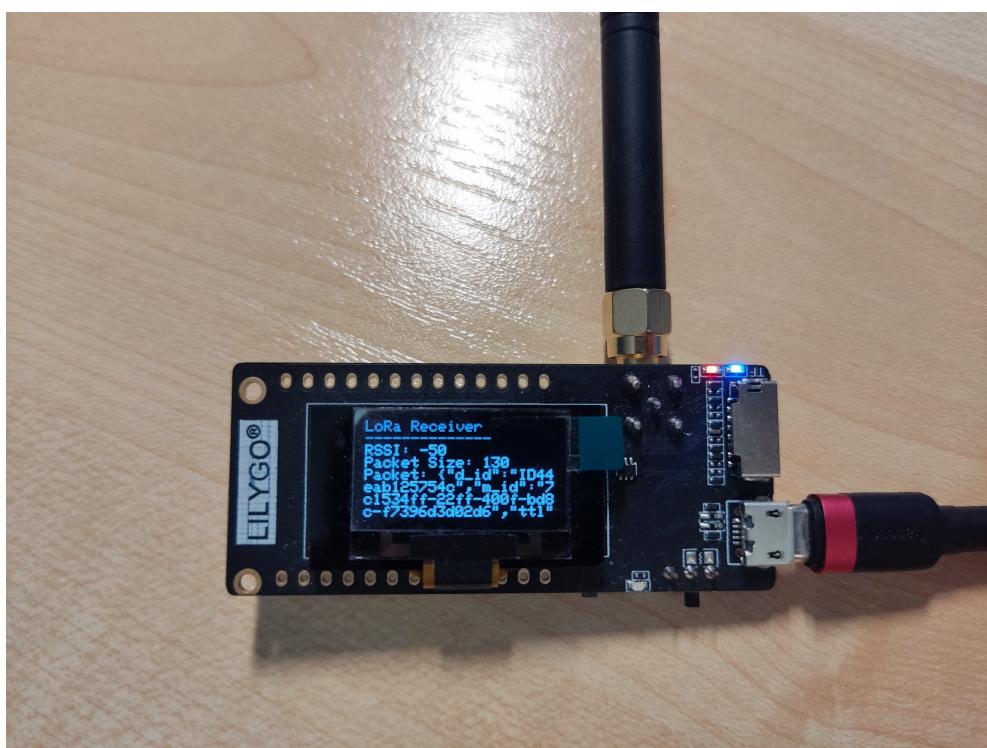
Rysunek 4.1: Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią

```
[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"}
[23:13:54.836] DEBUG: Message dropped: b'{"d_id": "ID_e6618c4e336192a"}'
[23:18:21.036] INFO: Message received: {"ttl": 10, "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "d_id": "ID_e6618c4e336192a"}, TX done.
[23:18:21.797] INFO: Message forwarded: b'{"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "ttl": 9, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:21.996] DEBUG: TX done.
[23:18:22.331] INFO: Message received: {"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "ttl": 8, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}
[23:18:22.334] DEBUG: Message dropped: b'{"m_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 7, "d_id": "ID_e6618c4e336192a"}'
[23:18:22.340] DEBUG: TX done.
[23:18:22.349] DEBUG: TX done.
[23:18:23.588] INFO: Message received: {"m_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 16, "d_id": "ID_e6618c4e336192a"}, TX done.
[23:18:24.269] INFO: Message forwarded: b'{"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "ttl": 5}'
[23:18:24.528] DEBUG: TX done.
[23:18:24.889] INFO: Message received: {"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "ttl": 14}
[23:18:25.049] DEBUG: Message dropped: b'{"ttl": 14, "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:25.810] DEBUG: TX done.
[23:18:26.149] INFO: Message received: {"ttl": 12, "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "d_id": "ID_e6618c4e336192a"}, TX done.
[23:18:26.819] INFO: Message forwarded: b'{"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "ttl": 11, "m_id": "5432817c-37f6-4957-871a-59545a6b359d"}'
[23:18:27.079] DEBUG: TX done.
[23:18:27.434] INFO: Message received: {"d_id": "ID_e6618c4e336192a", "values": {"humidity": 66, "temperature": 17}, "m_id": "5432817c-37f6-4957-871a-59545a6b359d", "ttl": 10}
[23:18:27.544] WARNING: Message dropped: b'{"m_id": "5432817c-37f6-4957-871a-59545a6b359d", "values": {"humidity": 66, "temperature": 17}, "ttl": 9}' (TTL is 0)
[23:18:45.172] INFO: Message received: {"d_id": "ID44ebab125754c", "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "ttl": 10, "values": {"random_values": 78, "random_values2": 21})
[23:18:45.843] INFO: Message forwarded: b'{"values": {"random_values": 21, "random_values2": 78}, "ttl": 9, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "d_id": "ID44ebab125754c"}'
[23:18:46.197] DEBUG: TX done.
[23:18:46.552] INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 8, "values": {"random_values": 78, "random_values2": 21}, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"}, TX done.
[23:18:47.111] INFO: Message forwarded: b'{"m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "values": {"random_values": 78, "random_values2": 21}"}, TX done.
[23:18:47.501] DEBUG: TX done.
[23:18:47.946] INFO: Message received: {"d_id": "ID44ebab125754c", "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": 21, "random_values2": 78}, "ttl": 5, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "d_id": "ID44ebab125754c"}'
[23:18:48.882] DEBUG: TX done.
[23:18:49.031] INFO: Message received: {"d_id": "ID44ebab125754c", "ttl": 4, "values": {"random_values": 78, "random_values2": 21}, "m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93"}, TX done.
[23:18:49.034] INFO: Message forwarded: b'{"m_id": "e3852dc8-3e91-448e-8ca0-60552462cc93", "values": {"random_values": 21, "random_values2": 78}, "d_id": "ID44ebab125754c"}'
[23:18:59.276] DEBUG: TX done.
[23:18:59.720] INFO: Message received: {"d_id": "ID44ebab125754c", "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 T3 LoRa32



Rysunek 4.4: Zdjęcie stacji przekaźnikowej opartej o TTGO T3 LoRa32

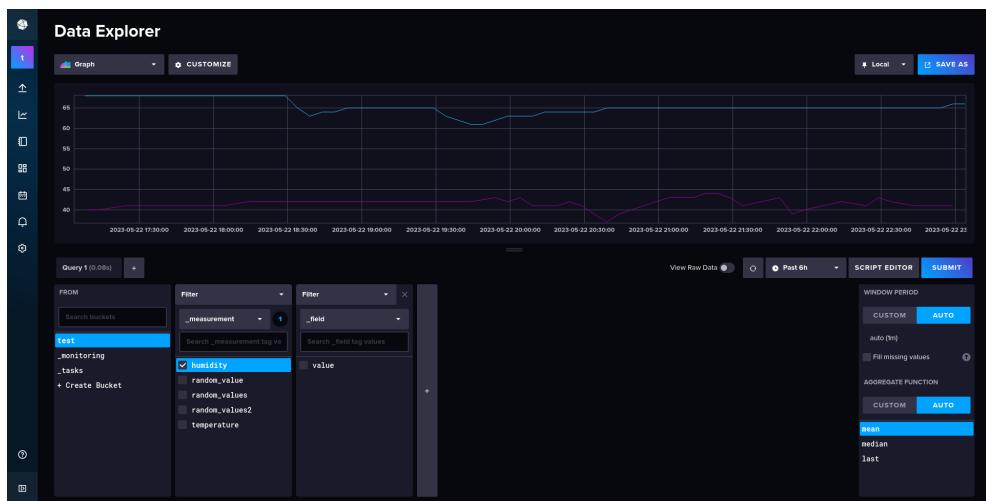
GB. Wszystkie usługi zostały uruchomione w kontenerach Dockera [14], w jednej sieci wirtualnej, w celu zapewnienia komunikacji między nimi.

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. Nie zostało użyte szyfrowanie TLS, ponieważ sieć wewnętrzna została uznana za bezpieczną, a użycie szyfrowania zwiększyło by skomplikowanie systemu.

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. Program zapisujący dane został skonfigurowany tak, aby połączyć się z bazą danych za pomocą tego tokenu. 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

Redis

Redis został uruchomiony na wspólnym serwerze wewnętrz budynku. Redis pozostał niezabezpieczony, ponieważ został udostępniony w sieci wewnętrznej. Tutaj również nie zostało użyte szyfrowanie. 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, następujących sposobów:

Weryfikacja poprawności działania urządzeń pomiarowych

Weryfikacja poprawności działania urządzeń pomiarowych została przeprowadzona dla odczytów temperatury. Test ten polegał na porównaniu odczytów z dwóch różnych czujników temperatury w tym sam czasie. 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 (mak-symalna 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-b514d43b9d97"}, {"d_id": "ID44eb125754c"}]
2023-05-22 21:08:46 INFO Received message: {"d_id": "ID44eb125754c", "ttl": 8, "values": [{"random_values": 53, "random_values2": 92}], "m_id": "9b85e246-919f-4b08-b514d43b9d97"}
2023-05-22 21:08:46 INFO Message already processed
2023-05-22 21:08:47 INFO Received message: {"m_id": "9b85e246-919f-4b08-b514d43b9d97", "ttl": 7, "values": [{"random_values2": 92, "random_values": 53}, {"d_id": "ID44eb125754c"}]}
2023-05-22 21:08:47 INFO Message already processed
2023-05-22 21:08:47 INFO Received message: {"d_id": "ID44eb125754c", "ttl": 6, "m_id": "9b85e246-919f-4b08-b514d43b9d97"}, {"values": [{"random_values": 53, "random_values2": 92}]}
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-b514d43b9d97"}, {"d_id": "ID44eb125754c"}]}
2023-05-22 21:08:48 INFO Message already processed
2023-05-22 21:08:49 INFO Received message: {"d_id": "ID44eb125754c", "ttl": 4, "values": [{"random_values": 53, "random_values2": 92}], "m_id": "9b85e246-919f-4b08-b514d43b9d97"}
2023-05-22 21:08:49 INFO Message already processed
2023-05-22 21:08:50 INFO Received message: {"d_id": "ID44eb125754c", "ttl": 2, "m_id": "9b85e246-919f-4b08-b514d43b9d97"}, {"values": [{"random_values": 53, "random_values2": 92}], "m_id": "9b85e246-919f-4b08-b514d43b9d97", "d_id": "ID44eb125754c"}
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": 1, "m_id": "9b85e246-919f-4b08-b514d43b9d97"}, {"d_id": "ID44eb125754c"}]}
2023-05-22 21:08:51 INFO Message already processed
2023-05-22 21:08:51 INFO Received message: {"d_id": "ID44eb125754c", "ttl": 0, "values": [{"random_values": 53, "random_values2": 92}], "m_id": "9b85e246-919f-4b08-b514d43b9d97", "r": 0}
2023-05-22 21:08:53 WARNING Received message: Non-R.Y.: u65bmQ.y JR
<-61KH30VtKw6rp NgYrYC
|_0a:$`Y
|_QSBUV/*uK/Eei@e\n2
ULV'ID7YY|i5|i:T|M=-~f
>zx |4
2023-05-22 21:08:53 ERROR Invalid json message
2023-05-22 21:08:54 INFO Received message: {"ttl": 10, "values": [{"humidity": 40, "t": 31.61, "sDM": 22}, {"m_id": "146a7a28-aec6-451d-b2b0-97a33015f6fc"}, {"d_id": "ID_e6614104038c5938"}]
2023-05-22 21:08:55 ERROR Invalid json message||t=9, "values": [{"humidity": 40, "tempe": 0}

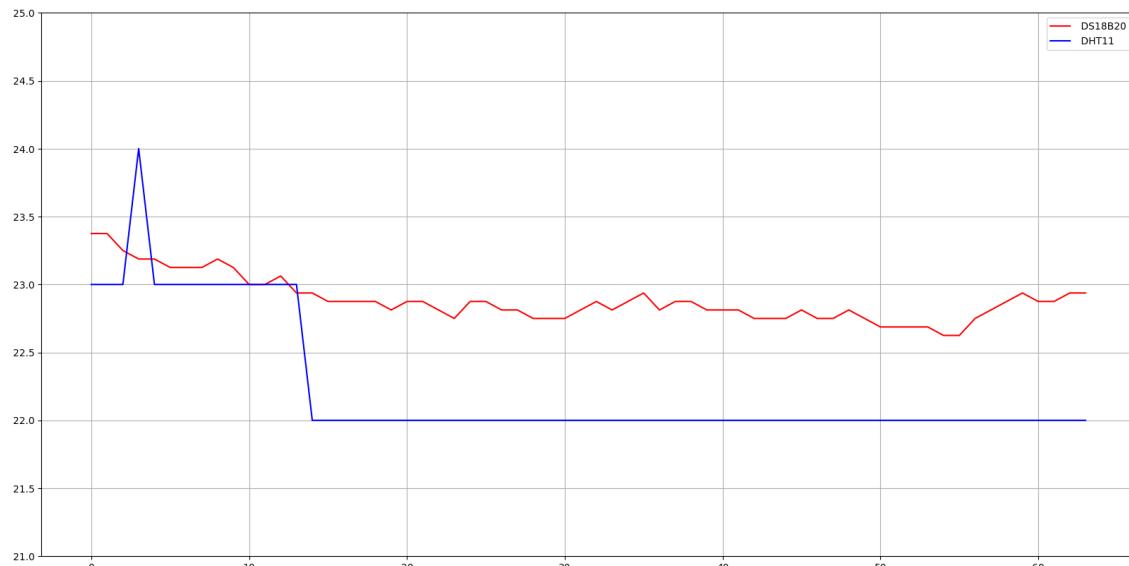
```

Rysunek 4.6: Zrzut ekranu zawierający logi programu zapisującego pomiary do bazy Influ-xDB 2

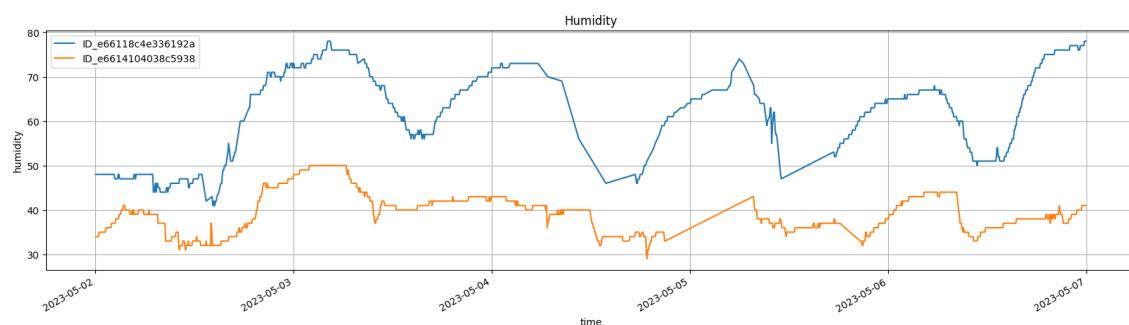
Weryfikacja poprawności działania całego systemu

Weryfikacja poprawności działania całego systemu została przeprowadzona poprzez porównywanie pakietów wysyłanych przez urządzenia pomiarowe z pakietami zapisanymi w bazie danych. Próba ta 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 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 trzech 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 brokeru 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 trzech węzłów sieci były emitowane pakiety w bardzo krótkich odstępach czasu (co 10 sekund). Weryfikacji podlegało to, czy wszystkie pakiety zostały wysłane ponownie rozgłoszone przez węzły sieci. Wszystkie pakiety były odbierane również przez stację przekaźnikową.

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

```

INFO: Message received: {"ttl":9,"values":{"humidity":79,"temperature":13}, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "d_id": "ID_e66118c4e336192a"}
INFO: Message forwarded: b'("d_id": "ID_e66118c4e336192a", "values": {"humidity": 79, "temperature": 13}, "ttl": 8, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID_e66118c4e336192a", "values": {"humidity":79,"temperature":13}, "ttl":7, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"}
INFO: Message forwarded: b'("m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "values": {"humidity": 79, "temperature": 13}, "ttl": 6, "d_id": "ID_e66118c4e336192a")
DEBUG: TX done.
INFO: Message received: {"m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402"}, "values": {"humidity":79,"temperature":13}, "ttl":5, "d_id": "ID_e66118c4e336192a"}
INFO: Message forwarded: b'("m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "values": {"humidity": 79, "temperature": 13}, "ttl": 4}
DEBUG: TX done.
INFO: Message received: {"d_id": "ID_e66118c4e336192a", "values": {"humidity":79,"temperature":13}, "ttl":4, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "d_id": "ID_e66118c4e336192a"}
INFO: Message forwarded: b'("ttl": 2, "values": {"humidity": 79, "temperature": 13}, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "d_id": "ID_e66118c4e336192a")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID_e66118c4e336192a", "values": {"humidity":79,"temperature":13}, "ttl":3, "m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "d_id": "ID_e66118c4e336192a"}
INFO: Message forwarded: b'("m_id": "1710aa99-16ac-4783-84cb-9c0a9e01f402", "values": {"humidity": 79, "temperature": 13}, "ttl": 2)
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "m_id": "60de50d6-5086-4d83-820f-955c41877562", "ttl":10, "values": {"random_values":32, "random_values2":68}}
INFO: Message forwarded: b'("values": 68, "random_values": 32, "random_values2": 68, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "ttl": 8, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c"}
INFO: Message forwarded: b'("m_id": "60de50d6-5086-4d83-820f-955c41877562", "ttl": 7, "values": {"random_values": 32, "random_values2": 68}, "d_id": "ID44eab125754c")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "ttl": 6, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "values": {"random_values": 32, "random_values2": 68}}
INFO: Message forwarded: b'("values": 68, "random_values": 32, "random_values2": 68, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "ttl": 4, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c"}
INFO: Message forwarded: b'("m_id": "60de50d6-5086-4d83-820f-955c41877562", "ttl": 3, "values": {"random_values": 32, "random_values2": 68}, "d_id": "ID44eab125754c")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "ttl": 2, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "values": {"random_values": 32, "random_values2": 68}}
INFO: Message forwarded: b'("values": 68, "random_values": 32, "random_values2": 68, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c")
DEBUG: TX done.
INFO: Message received: {"d_id": "ID44eab125754c", "ttl": 0, "values": {"random_values": 32, "random_values2": 68}, "m_id": "60de50d6-5086-4d83-820f-955c41877562", "d_id": "ID44eab125754c"}, TTL is 0
WARNING: Message dropped: {'m_id': '60de50d6-5086-4d83-820f-955c41877562'}, TTL is 0

```

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

4.3 Zauważone problemy

W trakcie testów zostały zauważone następujące problemy:

Błędne pakiety

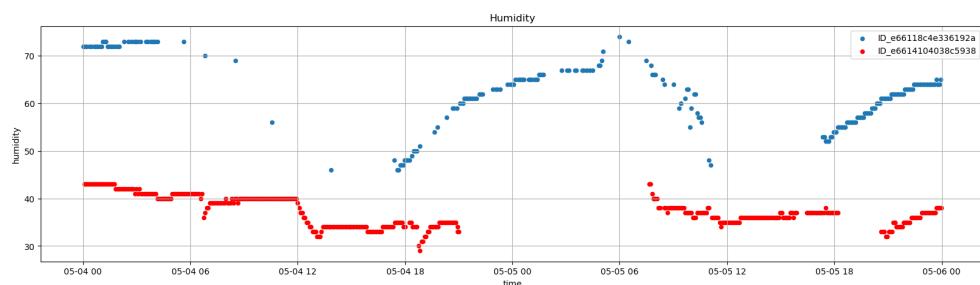
- Węzły sieci czasami odbierały błędne pakiety, jak na rysunku 4.10. Problem ten został rozwiązywany przez sprawdzanie czy odebrany pakiet jest poprawny, a w przypadku błędu, pakiet był odrzucany.
- Przekaźnik sieci czasami przekazuje błędne pakiety do brokera (jak na rysunku 4.10). Problem ten został 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.

Błędy pochodzenia zewnętrznego

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 nieuwążne zmiany kabli zasilających czy wyczerpanie się baterii. Jednym z możliwych rozwiązań jest baterii o większej pojemności 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 zbiernania danych został zaimplementowany. Wszystkie urządzenia zostały przetestowane i działają poprawnie. Jednak w przyszłości można rozwinać 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 rozwinać 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.

Usypanie węzłów sieci

Wiele mikrokontrolerów posiada funkcjonalność głębokiego uśpienia. W takim trybie mikrokontroler zużywa bardzo mało energii, jednak wtedy 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

W celu wydłużenia czasu działania węzłów sieci można również zastosować większe baterie. Wydłużyłoby to nieprzerwany 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. Węzły sieci będą musiały autoryzować się przed dołączeniem do sieci. Dzięki temu można zwiększyć wtedy bezpieczeństwo systemu.

Szyfrowanie wiadomości

W obecnej wersji systemu wiadomości wysyłane przez węzły sieci nie są szyfrowane. Biorąc pod uwagę specyfikę protokołu LoRa wiadomości są rozgłaszczone w formie broadcastu, więc mogą zostać odebrane przez dowolne urządzenie. 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 brokera wiadomości. Dzięki temu można zwiększyć bezpieczeństwo systemu.

Należałoby 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 poufne.

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. 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. Podczas prac nad następną iteracją projekt można rozwinać 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 bardzo małej sieci (trzy węzły). Powinny jednak 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 zużycia energii

W trakcie przyszłych prac nad projektem należałyby przeprowadzić badania konsumpcji energii węzłów sieci. W takiej sytuacji można sprawdzić, jak długo węzły sieci mogą działać na baterii. Należałyby 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 należałyby 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

W pracy został przedstawiony autorski projekt systemu zbierania rozproszonych danych z wykorzystaniem sieci LoRa, który został też zaimplementowany i wdrożony. System spełnił wszystkie założenia projektowe, to jest:

- jeden centralny punkt gromadzenia danych,
- zapisywanie danych do bazy danych szeregow 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,
- możliwie duża dostarczalność pakietów,
- dopuszczalne drobne opóźnienia w dostarczaniu danych.

Został on przetestowany w praktyce — choć w bardzo niewielkiej skali. Testy te pokazały, że system działa poprawnie i może być użyty w rzeczywistych zastosowaniach. Przedstawiono również dalsze możliwości rozwoju systemu.

Spis listingów

3.1 Przykładowa wiadomość przesyłana przez system	20
---	----

Spis rysunków

1.1	Schemat wyprowadzeń TTGO T3 LoRa32 (źródło: https://www.lilygo.cc/products/lora3)	9
1.2	Schemat wyprowadzeń Raspberry Pi Pico (źródło: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html)	10
1.3	Interfejs narzędzia PlatformIO w edytorze Visual Studio Code	12
3.1	Diagram prezentujący schemat ideowy systemu	19
4.1	Zdjęcie węzła sieci opartego o Raspberry Pi Pico wraz z baterią	26
4.2	Zrzut ekranu zawierający logi z jedno z węzłów sieci(Raspberry Pi Pico)	26
4.3	Zdjęcie węzła sieci opartego TTGO T3 LoRa32	27
4.4	Zdjęcie stacji przekaźnikowej opartej o TTGO T3 LoRa32	27
4.5	Zrzut ekranu interfejsu użytkownika InfluxDB 2	28
4.6	Zrzut ekranu zawierający logi programu zapisującego pomiary do bazy InfluxDB 2	29
4.7	Wykres temperatury od czasu dla czujników DHT11 i DS18B20	30
4.8	Wykres wilgotności od czasu(w okresie od 2 Maja 2023 do 7 Maja 2023)	30
4.9	Przykładowy pakiet, odebrany przez jeden z węzłów sieci	31
4.10	Przykładowy błędny pakiet odebrany przez jeden z węzłów sieci	32
4.11	Szczegółowych diagram danych zapisanych w bazie między 4 Maja, a 6 Maja 2023	32

Bibliografia

Artykuły

- [1] 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.
- [2] 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>.
- [3] 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>.
- [4] 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.

Odnosniki w sieci

- [5] ietf. *A Universally Unique IDentifier (UUID) URN Namespace*. 2005. URL: <https://datatracker.ietf.org/doc/html/rfc4122> (term. wiz. 23.04.2023).
- [6] mqtt.org. *MQTT*. 2019. URL: <https://mqtt.org/mqtt-specification/> (term. wiz. 23.04.2023).
- [7] 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).
- [8] Waveshare. *Waveshare SX1262 LoRa Node Module 868MHz*. 2019. URL: <https://www.waveshare.com/wiki/Pico-LoRa-SX1262> (term. wiz. 16.04.2023).

- [9] *pubsubclient*. 2020. URL: <https://github.com/knolleary/pubsubclient> (term. wiz. 16.04.2023).
- [10] Semtech. *SX1276/77/78/79 - 137 MHz to 1020 MHz Low Power Long Range Transceiver*. 2020. URL: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R0000001Rbr/6EfVZUorrpoKFFvaF_Fkpgp5kzjiNyiaBqcpqh9qSjE (term. wiz. 16.04.2023).
- [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] Redis Ltd. *Redis Documentation*. 2023. URL: <https://redis.io/docs/> (term. wiz. 23.04.2023).
- [23] *micropySX126X*. 2023. URL: <https://github.com/ehong-tl/micropySX126X> (term. wiz. 16.04.2023).

- [24] *MicroPython*. 2023. URL: <https://micropython.org/> (term. wiz. 16.04.2023).
- [25] Eclipse Mosquitto. *Eclipse Mosquitto*. 2023. URL: <https://mosquitto.org/> (term. wiz. 23.04.2023).
- [26] PlatformIO. *PlatformIO Documentation*. 2023. URL: <https://docs.platformio.org/en/latest/> (term. wiz. 23.04.2023).
- [27] Redis. *redis-py*. 2023. URL: <https://redis.io/docs/clients/python/> (term. wiz. 23.04.2023).
- [28] Espressif Systems. *ESP32 Datasheet*. 2023. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf (term. wiz. 16.04.2023).
- [29] Espressif Systems. *ESP32 SoCs*. 2023. URL: <https://www.espressif.com/en/products/socs> (term. wiz. 16.04.2023).
- [30] LoRa. *LoRa Documentation*. URL: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan> (term. wiz. 23.04.2023).