Faculdade de Engenharia da Universidade do Porto

SWRS :: Smart Water Reuse System

Matilde Reis da Ribeira

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Luís Miguel Pinho de Almeida

July 26, 2023

Abstract

Addressing the escalating global water scarcity crisis necessitates innovative solutions such as the Smart Water Reuse system, which leverages Internet of Things (IoT) technologies. This dissertation presents a comprehensive investigation of microcontrollers, sensors, wireless technologies, and IoT frameworks, along with the implementation of a prototype designed to alleviate water scarcity challenges.

The proposed Smart Water Reuse system focuses on reutilizing water by diverting it from hand basins or bathtubs to toilet flushing modules, using real-time data gathered through a smart meter system. The system incorporates a battery-powered Arduino Nano 33 IoT board, which commands TDS sensors to measure water quality and water flow sensors to measure quantity. Based on the collected data, the water undergoes filtration before being redirected to the flushing system. Communication between the sensors, boards, and the cloud is facilitated by a Raspberry Pi 4 acting as the gateway. All data is stored in the InfluxDB database and transmitted to the cloud for post-processing, enabling the provision of insightful information through a user-friendly dashboard.

The self-powered nature of the system is achieved by utilizing the output voltage of a water turbine to charge the Arduino board. This sustainability feature ensures continuous operation without relying on external power sources.

This dissertation demonstrates the potential of the Smart Water Reuse system to reutilize up to 30% of water, providing a tangible solution to global water scarcity. By combining IoT technologies, smart meters, and efficient data processing, the system offers real-time insights into water savings and pre- and post-filtration water quality.

Resumo

Para fazer face à crescente crise global de escassez de água, são necessárias soluções inovadoras, como o sistema de reutilização inteligente de água, que tira partido das tecnologias da Internet das Coisas (IoT, do inglês Internet of Things). Esta dissertação apresenta uma investigação abrangente de microcontroladores, sensores, tecnologias sem fios e estruturas IoT, juntamente com a implementação de um protótipo concebido para aliviar os desafios da escassez de água.

O sistema de reutilização inteligente da água proposto centra-se na reutilização desta, desviando-a de lavatórios ou banheiras para módulos de descarga de sanitas, utilizando dados em tempo real recolhidos através de um sistema de *smart meter*. O sistema incorpora uma placa IoT Arduino Nano 33 alimentada por bateria, que comanda sensores de Sólidos Dissolvidos Totais (TDS, do inglês Total Dissolved Solids) para medir a qualidade da água, e sensores de fluxo de água para medir a quantidade. Com base nos dados recolhidos, a água passa por uma filtragem antes de ser redireccionada para o sistema de descarga. A comunicação entre os sensores, as placas e a *cloud* é facilitada por uma Raspberry Pi 4 que actua como gateway. Todos os dados são armazenados na base de dados InfluxDB e transmitidos para a *cloud* para pós-processamento, permitindo o fornecimento de informações relevantes através de um *dashboard*.

A natureza auto-alimentada do sistema é conseguida através da utilização da tensão de saída de uma turbina de água para carregar a placa Arduino. Esta característica de sustentabilidade garante um funcionamento contínuo sem depender de fontes de energia externas.

Esta dissertação demonstra o potencial do sistema Smart Water Reuse para reutilizar até 30% da água, fornecendo uma solução tangível para a escassez global de água. Combinando tecnologias IoT, *smart meters* e processamento eficiente de dados, o sistema oferece informação em tempo real sobre a economia de água e a qualidade da água pré e pós-filtração.

Agradecimentos

Gostaria de agradecer ao meu orientador, o Professor Luís Almeida, pela gentileza e paciência intermináveis, e por ter feito com que enviar emails seja uma experiência menos intimidante. Estou segura de que cresci, tanto a nível académico como pessoal, por ter tido a oportunidade de ter sido acompanhada por si.

Ao Bruno e à Iara, que não me fazem esquecer quem sou e de onde venho.

Ao Armando, ao Campanhã, à Catarina, à Cláudia, ao Filipe, ao Miguel e ao Pedro. Foi uma honra ter-vos ao meu lado nestes últimos anos, e sem as noites de jogos de tabuleiro, os piqueniques no jardim central e as conversas infinitas, esta experiência não teria valido a pena.

A todas as mulheres que lutaram para que eu pudesse estudar - e à minha mãe. A todas as que não puderam estudar - e à minha avó. A todas as que continuam a lutar para que as outras o possam fazer.

À Inês. Aos astros que se alinharam e fizeram com que te conhecesse no primeiro dia de universidade, e que saiamos de mãos dadas no último.

Matilde Ribeira

"Pode ser que para outro mundo eu possa levar o que sonhei.
Mas poderei eu levar para outro mundo o que me esqueci de sonhar?"

Álvaro de Campos

Contents

List of Figures

List of Tables

Abbreviations and symbols

ADC	Analog-to-Digital Converter
AHB	Advanced High-Performance Bus
AI	Artificial Intelligence
AP	Access Point
APB	Advanced Peripheral Bus
AWS	Amazon Web Services
API	Application Programming Interface
BLE	Bluetooth Low Energy
CoAP	Constrained Application Protocol
CPU	Central Processing Unit
CSV	Comma-Separated Values
DHCP	Dynamic Host Configuration Protocol
DTIM	Delivery Traffic Indication Message
EIM	Electronic Interface Module
GCP	Google Cloud Platform
GPIOs	General-purpose input/output
HTTP	Hypertext Transfer Protocol
LCD	Liquid Crystal Display
Li-Po	Lithium Polymer
LPWAN	Low-Power Wide-Area Network
M2M	Machine-to-Machine
MAC	Media Access Control
MQTT	Message Queuing Telemetry Transport
NiCd	Nickel-Cadmium
NiMH	Nickel–Metal Hydride
PM	Power Manager
QoS	Quality of Service
RTC	Real-Time Clock
RTOS	Real-Time Operating System
SBC	Single-Board Computer
SSID	Service Set Identifier
TDS	Total Dissolved Solids
UDP	User Datagram Protocol
ULP	Ultra-Low Processor
WFI	Wait for Interrupt
WDT	Watchdog Timer
WLAN	Wireless Local-Area Network
WPAN	Wireless Personal-Area Network

WSN	Wireless Sensor Network

Chapter 1

Introduction

Nowadays, two billion people, or one out of every four, lack access to potable water at home [4]. By 2025 over half of the world population will reside in water-stressed areas. This issue expands to another one: sanitation. Without neither enough water to support it or the needed infrastructure, two billion people still lack basic sanitation. Water and sanitation access are expected to keep increasing; however, mere access is not enough. If the water cleanliness and safety are not assured, individuals are more prone to diseases like cholera, dysentery, hepatitis A, typhoid, and polio [4]. But whereas water and sanitation access are slowly ascending, another problem comes to the surface: water scarcity. Scientists have found that global warming has not only caused the disintegration of ice sheets into the sea, but also increased drought occurrences, both of which cause a major water necessity. As half of the world's main aquifers are overexploited, the freshwater they hold decreases [5], turning this into a serious threat to global water scarcity.

In conclusion, it is in the best interest of humankind that we find ways to reduce - and reuse - our water consumption before the damage becomes irreparable. In Europe, research has shown that individuals use almost 150 liters of water per day [6], two-thirds of which are freshwater used in residential buildings. Of these, between 25% and 30% is used to flush the toilet (Figure 1.1).

Unlike drinking, cooking, or bathing, flushing does not require freshwater; greywater, which is wastewater from plumbing systems such as showers and hand basins, can be used instead [7]. It presents a lower contaminant load than any other wastewater generated in human activities, and when treated and filtered, this type of water can be reused onsite for toilet flushing, landscape or crop irrigation, and other non-potable uses.

The latest technological advances in the Internet of Things (IoT) allow the creation of a wirelessly connected Smart Meter, capable of reusing flushed water and processing the captured data to increasingly build a more accurate filtering system. The reutilization of water at this scale presents not only a significant environmental impact but also an economic advantage to potential clients since the scarcity of freshwater [8] will shortly begin to increase its price [9]. By developing a smart system capable of reusing greywater

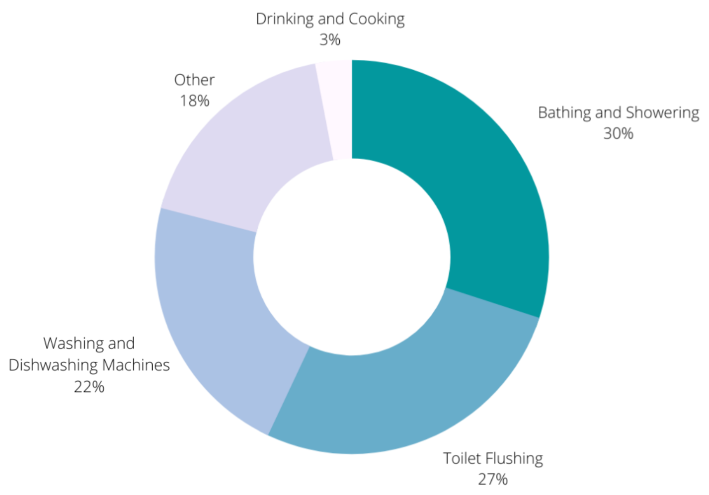


Figure 1.1: Indoors freshwater consumption by sector in an average European household [6]

in toilet flushing, water consumption will decrease, making universal basic sanitation and water access more achievable.

1.1 Project Goals

The purpose of this dissertation is to retrieve and present real-time water flow and quality data through a monitoring dashboard, to automatically control the water circuit, and to implement the necessary sensors, microcontrollers, and actuators to get, process, and use those data, as shown in Figure 1.2.

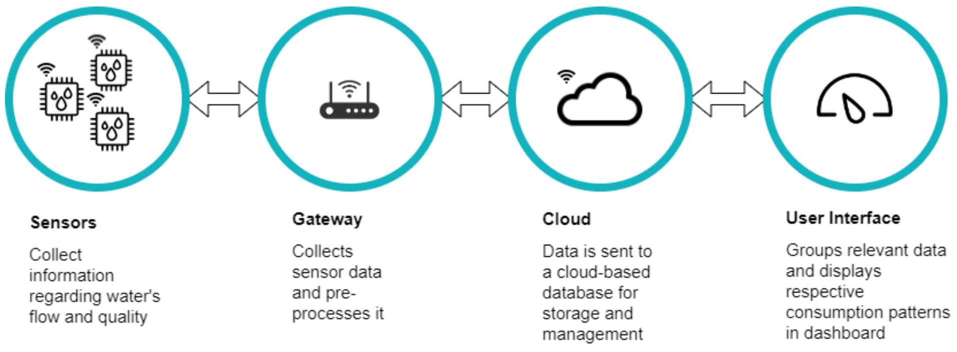

Sensors
Collect information regarding water's flow and quality
Gateway
Collects sensor data and pre-processes it
Cloud
Data is sent to a cloud-based database for storage and management
User Interface
Groups relevant data and displays respective consumption patterns in dashboard

Figure 1.2: Architecture of the proposed IoT system

The company SWRS has previously developed both the mechanical and chemical components for their flagship project, also named SWRS. When a user opens the shower tap or the hand basin tap, the system is activated and separates the water into three categories: ready to use (in which case it is redirected to the flushing system), contaminated but suited for treatment (in which case the water goes through a treatment module), or neither (and it goes into the wastewater stream).

This dissertation focuses on the implementation of water quality sensors, microcontrollers, and data management techniques in water quality sensing systems.

The study involves the implementation of water quality sensors that collect information about water quality and quantity. These sensors are connected to a chosen microcontroller, a battery-powered Arduino Nano 33 IoT, responsible for activating valves and pumps in the existing water filtering system. A Raspberry Pi is configured as a gateway for processing the sensor-collected data in real time. It facilitates communication between the sensors, gateway, and cloud through Wi-Fi. An external database stores all the processed information. Finally, a user dashboard was developed to present real-time data retrieved from the database, allowing users to monitor and manage their water usage effectively.

The power generation of the system, aiming to maximize its self-sufficiency using a water turbine and a battery charger, will be thoroughly discussed. Special emphasis will be placed on understanding the theoretical and experimental aspects related to the selection of microcontrollers. This involves a characterization of various available options and an assessment of their power consumption under different sleep modes.

Considering the scope of the project, it is necessary that this system fits the inside of a drain, that every component is energy efficient, and that it is as low-cost as possible to ensure the more vulnerable communities are given the opportunity to save resources, both natural and financial.

1.2 Document Structure

The dissertation structure comprises several key chapters, each contributing to a comprehensive understanding of the research problem and its proposed solution. Here is an overview of the chapters:

- Chapter 1 presents the context and motivation behind addressing the identified problem. It establishes a thorough understanding of the problem and outlines the objectives of the study.

- Chapter 2 delves into the theoretical background, focusing on essential technologies such as communication protocols, IoT frameworks, sensors, and time series databases.

- Chapter 3 provides a thorough literature review, extensively examining previous studies related to smart meters and recent innovations. This chapter compares these existing techniques with the proposed system architecture, highlighting both similarities and distinctive features.

- Chapter 4 concentrates on the theoretical exploration of three microcontroller options: the Arduino Nano 33 IoT, ESP8266, and ESP32. It delves into their respective frameworks and sleep modes, providing a theoretical foundation for subsequent analyses.
- Chapter 5 outlines the methodology employed in characterizing the aforementioned boards and presents the findings and conclusions derived from the characterization experiments.
- Chapter 6 focuses on describing each component of the system and its integration within the overall architecture of the system. The chapter provides a detailed account of the data flow from the sensor level to the ultimate destination, the user dashboard.
- Chapter 7 concludes the dissertation by summarizing the main findings and drawing conclusions based on the research outcomes. It also discusses potential areas for future work, highlighting topics for further exploration and development.

Chapter 2

Internet of Things Technologies

The concept of having a device reporting its status over a network was first used in the early 1980s by a group of university students that wanted to check if a cold Coca-Cola would be waiting for them when they got to the vending machine. At the end of the 1990s, the expression Internet of Things was created to refer to a system of devices and machines that relate and interact with each other and transfer data between each other through the Internet without the need for human interaction [10]. "Things" can be anything from someone from whom you want to collect biometric data to a place whose weather you want to monitor, as long as they are assigned a unique IP and are able to transfer data. These data are collected through sensors, then sent to a gateway, and processed if need be before being sent to the cloud. As IoT devices grow in popularity, there is one concern that grows with them: these devices hold the user's information, from their names to their address, and the hours they get home. One of the currently most challenging steps is to guarantee the security of the user's information without compromising the efficiency of the system.

2.1 IoT: Advantages and Uses

The use of IoT is more present in the transportation and manufacturing industries, but has been steadily increasing in the agriculture and home automation industries. IoT can be used to maximize efficiency and thus reduce energy waste and unnecessary costs (for example, by gathering data on weather forecasting that can be used to automate farming techniques) and can even aid the data collection step of developing Artificial Intelligence (AI) algorithms. It doesn't only help companies and industries, but also the end user: they can monitor their own health and habits, leading them to cheaper, more cost-efficient choices. The most obvious application of this is smart homes, which can, for example, be programmed so that the house heating unit only turns on if a user remotely controls it to do so and even adapt its temperature to the number of people in the room. Wearable devices are also increasing in use: by wearing, for example, a bracelet on their wrists, users can keep track of their vital signals and even send them directly to doctors, which can then be notified if anything is alarming. On a larger scale, IoT can be used to program, for example, smart streetlights, which can have a gigantic impact on energy savings.

2.2 IoT Data Protocols

A data protocol is a standardized set of rules for formatting and processing data [11]. They allow electronic devices to connect and communicate with each other. In the Internet of Things, these protocols specifically enable communication between edge devices and the Internet. Three of the most commonly used data protocols for IoT are Message Queuing Telemtry Transport (MQTT), Constrained Application Protocol (CoAP), and Hyper Text Transfer Protocol (HTTP), presented below.

2.2.1 MQTT

MQTT is a bi-directional data protocol that allows clients and servers to share data in a publisher/subscriber mode, and it is compatible with low power consumption. With MQTT, the client can constantly publish its own state, or it can send notifications upon events, e.g. when it disconnects. It can also provide buffering for devices that cannot do it themselves [12]. The message overhead is at least 2 bytes, and messages can weigh up to 256 MB. The use of the publish-subscribe model is reached through an intermediary: the broker, responsible for receiving the messages from the publishers and forwarding them to the respective subscribers (Figure 2.1).

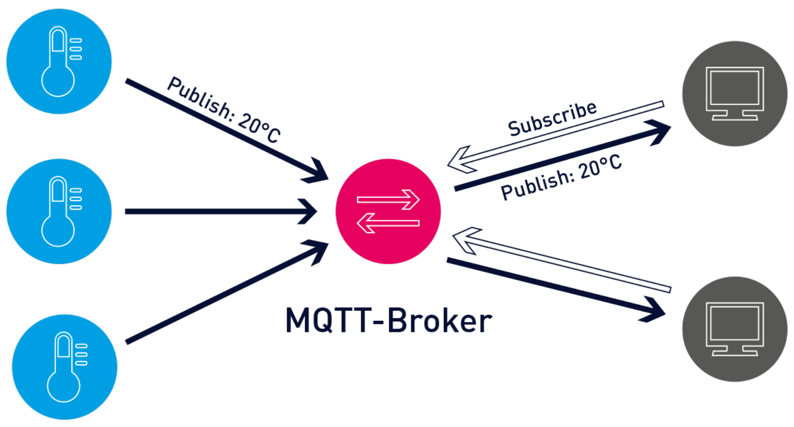


Figure 2.1: MQTT broker receiving sensor data and redirecting it to subscribed clients [13]

There are three levels of quality of service (QoS) in all MQTT connections that are related to the consistency of data delivery:

1. QoS 0. At most once: the message is sent once, independently of its successful arrival. There are no acknowledgments to let the broker know if it arrived, so there might be a data loss (e.g., if a subscriber crashes).

2. QoS 1. At least once: the message is sent as many times as it needs to be until an acknowledgment is received. Arrival is guaranteed, but there may be duplicates.

3. QoS 2. Exactly once: this is the only level that makes sure there is no data loss nor duplicates, by making both the sender and the receiver engage in a two-level handshake.

The higher the QoS level, the higher the bandwidth consumption and computational overhead, thus this choice must be made considering the importance of the reception of each message.

2.2.2 CoAP

CoAP is a low-power device transfer protocol specifically designed for machine-to-machine (M2M) applications like smart home meters. It is optimized to seamlessly integrate with HTTP while meeting requirements like simplicity, low power consumption, and low overhead. CoAP uses User Datagram Protocol (UDP) as its transport layer and can guarantee communication in congested networks [14], unlike MQTT. There are two types of CoAP messages: confirmable and non-confirmable. Confirmable messages guarantee reliability, as the client is certain of the arrival of the message to the server since the message will keep being sent until a Rest message (RST) or an acknowledgment of reception (ACK) is received. Unconfirmable messages do not require an acknowledgment and are used for noncritical messages, like sensor-read values. Another extremely important feature of CoAP is its ability to receive information asynchronously: the client asks the server to get notified only when there are updates to a specific resource, eliminating redundant information. This is considered an extension to the CoAP protocol, providing it with publish-subscribe communication capabilities.

2.2.3 HTTP

Unlike MQTT, HTTP servers respond only to client requests, and each of these requests is handled individually, with its overhead being carried out each time [12]. It is not possible to keep track of the client's state, so if a client unexpectedly disconnects, the process will continue normally. It is more expensive, power-consuming and heavier than MQTT and is recommended for large amounts of data. Although HTTP servers do not feature some important IoT features like queueing and retained messages, they offer several types of request to interact with web resources. These requests are commonly referred to as HTTP methods and include:

- GET: used to retrieve a representation of a specified resource. It is the most commonly used HTTP method and is primarily used for retrieving data.
- POST: used to submit data to be processed by the identified resource. It is commonly used to create new resources on the server or trigger some server-side actions.
- PUT: used to update or replace a specified resource with the request payload. It is often used to update existing resources.
- DELETE: used to delete a specified resource on the server. It instructs the server to remove the identified resource.

2.2.4 Summary

Table 2.1 briefly presents the differences between MQTT, CoAP and HTTP.

Protocol	Pattern	Base Protocol	Reliability
MQTT	Publish-Subscribe	TCP	Delivery not guaranteed Delivery confirmed Delivery double confirmed
CoAP	Request-Response	UDP	Confirmable Non-confirmable ACK RST
HTTP	Request-Response	TCP	N/A

Table 2.1: Summary of the characteristics of the 3 data protocols

2.3 Time Series Databases

Databases are essential in IoT to store, organize and analyze the data generated by IoT devices. A time series database, specifically, specializes in handling time-stamped data. It is optimized for storing, retrieving, and analyzing data points that are associated with specific timestamps, such as sensor data or IoT device readings.

The use of data compression techniques allows to compress large volumes of data, and query mechanisms ensure the efficient retrieval of data. One of the most important concepts of time series databases is downsampling. Downsampling involves summarizing detailed time series data and storing the summarized information in a separate container with a lower resolution. For instance, it might not be particularly useful for an application to store sensor-retrieved data by the minute. A downsampling operation can help retain the mean hourly value of that data instead, allowing the preservation of storage space and enhancing query speed without compromising the quality of the data.

This type of database usually offers built-in functions and tools for aggregating and analyzing data and generates insights based on calculation and statistical analysis. In this section, three of the most widespread alternatives for time series databases are presented.

2.3.1 InfluxDB

InfluxDB is an open-source time series database developed by InfluxData. Among its biggest advantages are the high-availability retrieval of data and the ability to handle high write and query loads (up to millions of writes per second) which make it a good choice for IoT applications.

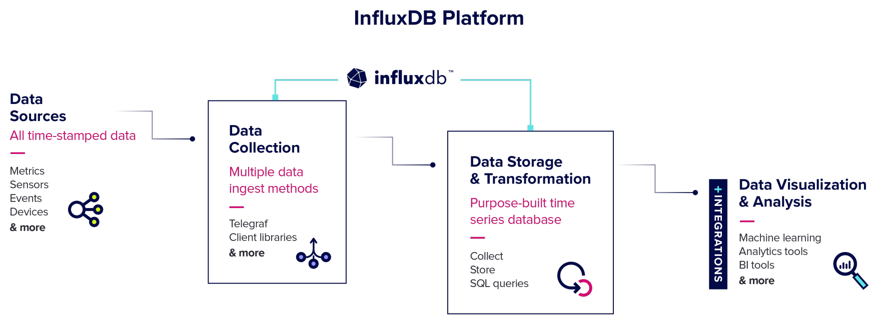


Figure 2.2: InfluxDB in data collection and storage, extracted from [15]

It uses an SQL-like language, InfluxQL, which enables easy filtering of time-series data based on specific criteria. Like most time series databases, it supports compression techniques and downsampling, but unlike its competition, it supports built-in continuous queries which periodically aggregate and downsample data in real-time. Additionally, it uses a tagging system that enables efficient filtering based on metadata. The use of retention policies determines how long data is stored in the database, allowing efficient data management and control over storage resources.

Even though InfluxDB has its own built-in user interface, it does not present the extensive feature of dedicated visualization tools, but it supports integration with various other ones. A popular choice for a visualization tool is Grafana, which offers seamless integration with InfluxDB and creates interactive dashboards and charts, enabling users to monitor trends and detect anomalies. A chart showcasing the role of InfluxDB in data collection and storage is shown in Figure 2.2.

2.3.2 Prometheus

Prometheus is an open-source monitoring and alerting system, originally developed by SoundCloud. It uses a pull-based model (Figure 2.3) for data collection, which allows it to monitor a wide range of applications simultaneously. It features an advanced query language, PromQL, enabling users to perform complex operations, aggregations, and filtering of time-series data.

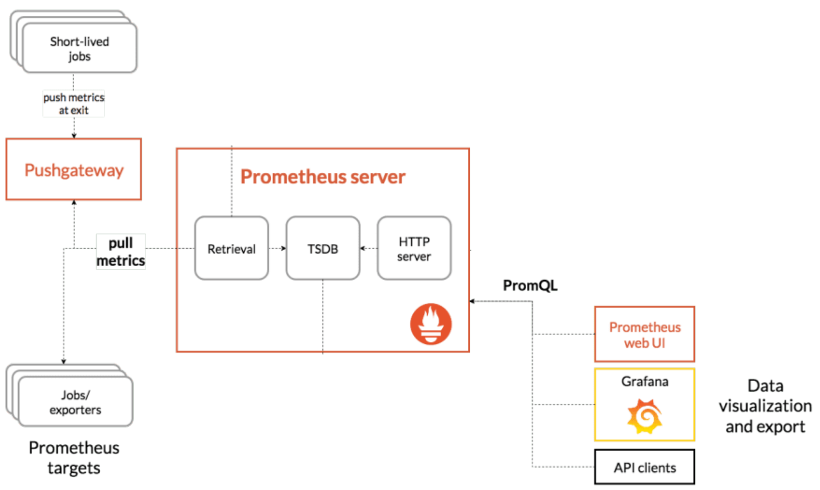


Figure 2.3: Architecture of pull-based system for data collection in Prometheus, adapted from [16]

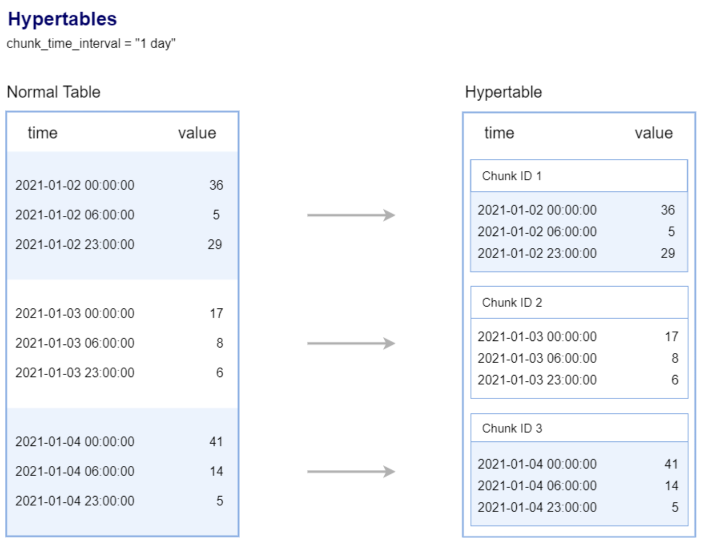
Prometheus distinguishes itself from other time series databases in two primary ways: firstly, by providing a powerful alerting system in which users can define alert rules. When one of the rules is triggered, Prometheus sends alerts to various notification channels (depending on what was set up). Secondly, Prometheus supports a wide range of service discovery mechanisms, which include static configuration files and DNS-based discovery. This enables automatic discovery and monitoring of new instances, which is deemed ideal for dynamic environments.

Like InfluxDB, Prometheus uses a configurable retention period and offers integration with various visualization tools. One disadvantage is that it uses a local on-disk storage engine which provides good performance for short-term storage but might not be the most appropriate choice for high-volume data.

2.3.3 TimescaleDB

TimescaleDB is an open-source time series database that extends PostgreSQL and supports standard SQL queries. For relational data, users can use regular PostgreSQL tables, but TimescaleDB offers the option of using hypertables (Figure 2.4). These tables make it easier to handle time-series data by automatically partitioning the data by time. A hypertable is made up of "chunks", each of them containing data from a different time range. The storage engine manages the chunk lifecycle and data aging, which ensures efficient data management. Another pertinent feature of TimescaleDB is its horizontal scalability through distributed query execution, which can be deployed in a clustered setup across multiple nodes, enabling efficient handling of large datasets and high query workloads.

Similarly to the two time series databases presented above, TimescaleDB integrates well with popular visualization tools and frameworks, allowing seamless integration with



Normal Table

time	value
2021-01-02 00:00:00	36
2021-01-02 06:00:00	5
2021-01-02 23:00:00	29
2021-01-03 00:00:00	17
2021-01-03 06:00:00	8
2021-01-03 23:00:00	6
2021-01-04 00:00:00	41
2021-01-04 06:00:00	14
2021-01-04 23:00:00	5

Hypertable

time	value
Chunk ID 1	
2021-01-02 00:00:00	36
2021-01-02 06:00:00	5
2021-01-02 23:00:00	29
Chunk ID 2	
2021-01-03 00:00:00	17
2021-01-03 06:00:00	8
2021-01-03 23:00:00	6
Chunk ID 3	
2021-01-04 00:00:00	41
2021-01-04 06:00:00	14
2021-01-04 23:00:00	5

Figure 2.4: Structure of a hypertable in TimescaleDB, extracted from [17]

Grafana, Prometheus, and other monitoring and analytics platforms. However, it is not as specialized in advanced monitoring and alerting capabilities or in specific time series-oriented features, e.g. downsampling.

2.3.4 Summary

This dissertation delved into the significance of databases in the IoT domain, emphasizing the critical role of time series databases for storing and analyzing time-stamped data from IoT devices. In this section we explored three prominent time series databases: InfluxDB, known for its high-availability retrieval and support for high write and query loads; Prometheus, distinguished by its advanced query language and powerful alerting system; and TimescaleDB, extending PostgreSQL with efficient handling of time-series data and horizontal scalability.

2.4 IoT Frameworks

IoT frameworks exist to support the development of the IoT system. What they exactly do can range from an editable structure the developer can work with, to hundreds of pre-developed options for data analysis and control, along with guaranteed connectivity, security, etc. When integrated with time series databases, these frameworks provide a seamless way of managing and processing large volumes of data, and enable efficient storage, retrieval, and analysis of the data. The integration also enhances data integrity, reliability, and security, ensuring that the time series data remains accessible, accurate, and protected.

After researching the most adequate frameworks for IoT [18], three of them seem to stand out: Microsoft Azure, Google Cloud Platform (GCP), and Amazon Web Services

(AWS). The three offer similar features like processing, storage, databases, and networking, but minor differences distinguish each option. Table 2.2 summarizes these differences.

2.4.1 Amazon Web Services

AWS offers scalability, superior privacy and security measures, and the broadest service catalog, with more than 200 services available. An important feature of this service is its pay-per-use payment model, meaning the user only pays for what they need, as long as they need it. AWS cloud storage price can be chosen according to each individual need, and varies according to the chosen frequency and speed of data retrieval, making it a highly customizable service. Two of AWS's key tools are Gluon, a deep-learning library that helps to build neural networks, and SageMaker used to train and deploy machine learning models. Recommended applications are character recognition, image identification, and object identification. When it comes to ease of use, AWS features a complete and efficient dashboard, and extensive documentation, making it arguably the easiest platform to learn.

AWS can be integrated with various time series databases, including the three presented above. InfluxDB can be deployed on AWS using virtual servers with Amazon EC2 or container services like Amazon EKS. It supports long-term storage on Amazon S3 or persistent storage using Amazon EBS. Prometheus can run on AWS using EC2 instances or container services like Amazon EKS or Amazon ECS. AWS also offers integrations with services like Amazon CloudWatch for collecting and pushing metrics into Prometheus. TimescaleDB can be deployed on AWS using EC2 instances or Amazon RDS for PostgreSQL.

2.4.2 Google Cloud Platform

Apart from the aforementioned basic features, GCP includes advanced data analytics and development tools. Besides being a frontrunner in the AI domain, the use of Google Application Programming Interfaces (APIs) is made easy through Google Cloud, a cloud platform designed for that specific purpose, and some of the available APIs are natural language, speech, and translation. When considering pricing, Google Cloud Platform (GCP) unquestionably stands out as the favorable choice, offering a wide range of cost-saving benefits, including diverse discounts and incentives, coupled with a customer-centric pricing framework. As for cloud storage, Google uses a scalable system like the one used by Google Drive, guaranteeing high speeds no matter the amount of data and making it the perfect solution for massive amounts of data, e.g. media. Just like AWS, GCP has a good deal of available documentation and a good-enough dashboard.

Time series databases can be easily deployed on GCP using Compute Engine instances or container services like Google Kubernetes Engine (GKE). GCP provides robust storage solutions, including options like Google Cloud Storage and Cloud SQL for PostgreSQL. Integration with services like Google Cloud Monitoring and Stackdriver Logging enhances metrics collection and log analysis capabilities. Additionally, GCP's services like BigQuery enable advanced analytics and querying of time series data.

2.4.3 Microsoft Azure

Azure is recognized for its robust security features, including the Cloud Defender tool, an AI-powered solution designed to fortify users' cloud environments against cyber threats. With Cloud Defender, Azure provides an advanced system for discovering and mitigating vulnerabilities, making it easier for users to protect their data and applications. Another advantage is the possibility of moving data between sites without leaving the Azure network, leading to higher security, lower costs, and higher speeds. Azure leads the way when it comes to cognitive services with Face API, Computer Face API, and many others. Regarding cloud storage, Azure offers various storage services, as well as site recovery, backup, import/export, etc. The AI, Machine Learning (ML), and analytics services are first-class. Reportedly the lack of documentation for Azure makes it harder to implement and use than the two alternatives presented above.

InfluxDB can be deployed on Azure using Virtual Machines or Azure Kubernetes Service (AKS) for containers, with Azure Blob Storage for long-term storage. Prometheus runs on Azure using Virtual Machines or AKS, integrating with Azure Monitor and Log Analytics. TimescaleDB can be deployed on Azure using Virtual Machines or Azure Database for PostgreSQL, with Azure Data Explorer (ADX) for analytics.

2.4.4 Final Comparison

Table 2.2 presents the most important differences between each of the three frameworks. The smallest instance, in the second-to-last line, includes 2 CPUs and 8 GB of RAM.

Framework	AWS	Azure	GCP
No. of Services	200	100	60
Geographical Regions	22	54	34
Edge Locations	114+	116	200+
Pricing	Per Second	Per Minute	Per Minute
Smallest Instance	\$69/month	\$70/month	\$52/month
Key Tool	Gluon & Sage Maker	Cognitive Services	AI and APIs

Table 2.2: Differences between different IoT frameworks

2.5 Wireless Communication Technologies

At the beginning of 2022, it was predicted that by the end of the year, over 7 billion devices would be wirelessly connected [19], making wireless communication technologies a pillar of the IoT. We rely on wireless technologies to become faster and more efficient while having power consumption preoccupations in mind. From energy efficiency and quick transmission to extended coverage range, various wireless technologies are well-suited to meet the diverse demands of IoT. In this section, we present several communication technologies that are particularly pertinent to a Smart Meter system.

2.5.1 LPWAN

Low-power wide-area networks (LPWAN) are a type of technology that interconnects low bandwidth devices with a low transmission rate across long distances and long periods of time. Due to its low-power characteristics, LPWANs are particularly suitable for apps that require short and occasional messages, and years can go by without ever needing to replace the battery of their devices [20].

2.5.1.1 Sigfox

The Sigfox protocol is characterized by extremely low energy consumption. It offers easy installation without the need for configuration or connection. Sigfox is designed to handle small messages up to 12 bytes and it is resilient to the coexistence with other kinds of networks such as Wi-Fi [21]. It provides an impressive range, capable of traveling up to 10 km in urban areas and 40 km in rural areas. Sigfox operates on a one-way communication model, relying on proactive retransmission to minimize losses. However, it's important to note that messages can still be lost and are not encrypted. Additionally, it's worth noting that Sigfox operates on a subscription-based model, with its functionality and service availability dependent on the chosen provider.

2.5.1.2 LoRa

LoRa is an LPWAN technology that is not proprietary to any specific operators. It presents an extremely low power consumption and is suited for long distances, reaching up to 10 km in rural zones and up to 5 km in urban areas [21]. Because it functions with two-way communication, it is a good solution for critical applications that cannot afford to lose messages during the communication process. It can be configured as a private network and can achieve data rates between 0.3 kbit/s and 50 kbit/s, with a maximum payload of 243 bytes. Utilizing LoRa for the physical layer, in conjunction with LoRaWAN, a protocol designed to define the upper layers, presents a compelling choice for implementing IoT applications.

2.5.2 WPAN

WPAN, or Wireless Personal Area Network, is a type of network that connects communication devices in a relatively short (personal) area to reduce transmission power, data rate, and thus, consumed energy [22]. WPANs first appeared because of the need to connect sensors and peripherals to personal computers. These networks are known to be particularly cheap, allowing for a wide range of devices to connect with low power consumption. Among its advantages, security, and portability stand out the most [23].

2.5.2.1 Bluetooth

Bluetooth, standardized as IEEE 802.15.1, can be used as an alternative to wired communications with computer peripherals. Relying on short-range (typically up to 10 m) radio frequency, it exchanges data between two devices using key authentication and encryption, and it works in the 2.4 GHz frequency band. It is mainly used to either exchange files between two nearby devices or connect cell phones or desktops to wireless devices (e.g. wearables or gadgets that monitor physical activity or blood pressure, to connect PC peripherals without the need for wires, etc).

Bluetooth 4.0, also known as Bluetooth Low Energy (BLE), implements the same pairing modules, encryption, and authentication protocols, but it has a data rate of 1 Mbps, and the maximum transmit power is 10 mW. This version is widely regarded as a good IoT option because of its low power consumption and years of battery life and is ideally suited for indoor positioning and location services: the use of beacon-based navigation has oftentimes proven to be a worthy complement to the traditional GPS system, and it is mainly used to help people get to a location, indoor item finding, or get Points of Interest (PoI) information. Using Bluetooth's wireless sensor networks (WSN), one can monitor various environmental parameters such as light, temperature, or water. This has contributed to the significant increase in the adoption of Bluetooth in home applications, with annual shipments rising from 29 million in 2017 to over 80 million in 2022.

BLE operates in the same frequency band but ensures smoother transitions between data pulses. It achieves this by utilizing forty 2-MHz channels and employing Gaussian frequency shift modulation for data transmission [24]. This approach reduces interference problems compared to traditional Bluetooth, which uses a frequency-hopping spread spectrum (FHSS).

As mentioned above, BLE presents a much lower power consumption. This is due to it being in "sleep mode" until a connection starts, with every connection lasting only a few seconds, a peak current consumption of about 15 mA, and an average of 1 uA. As a consequence, small devices have a battery life of up to 10 years and save up to ten times the energy of Bluetooth. There is, of course, a trade-off: its range is considerably shorter than Bluetooth's, as well as the throughput - about 100-250 Kbps for BLE versus 2 Mbps for Bluetooth, up to ten times lower. It is also frequently discarded when Wi-Fi is an option: most organizations have the necessary Wi-Fi infrastructure, but if they were to implement a BLE system, this would require BLE beacons (responsible for powering the transmission of broadcast signals and enabling every BLE device's location), sensors, and more. Altogether, if one is willing to endure the range and throughput-related disadvantages, BLE is a cost-efficient option for occasional low-power short-range connections.

2.5.2.2 Zigbee

Zigbee is a communication protocol built upon the IEEE 802.15.4 standard that is mostly used for low-bandwidth applications, like home automation or data collection [25]. It was designed to be a simpler, cheaper alternative to other wireless networks, which is possible due to its low power and short-distance data transfer.

The data transmission rate can go up to 250 kbps, and the range goes anywhere from 10 to 100 m, depending on the environment and the presence of obstacles. Zigbee operates in a mesh network, meaning that if a connection fails between a master node and another node, the master can still reestablish the connection using another path, and no communication will get lost.

Similarly to BLE, Zigbee is well-suited for IoT applications that require low power consumption and support for wireless sensor networks, allowing for the monitoring of environmental parameters like temperature and humidity. Both protocols are commonly used in home automation systems to control smart devices, such as lights, thermostats, and security systems.

2.5.3 WLAN

WLANs (Wireless Local-Area Networks) are a group of devices that communicate via a wireless medium, normally Radio Frequency (RF). Data is sent in packets containing labels and instructions with a unique Media Access Control (MAC) address assigned to endpoints. There are two ways to set up a WLAN, illustrated in Figure 2.5: in infrastructure mode, all the endpoints are connected to each other through a wireless router, also called Access Point (AP), that provides connection to other segments of the network (possibly to the Internet). In the *Ad hoc* mode, the endpoints are connected directly, without the router. To ensure the security of an otherwise vulnerable network, encryption is normally used.

Today, WiFi is the de facto standard for WLAN protocols. It is based on the standard family IEEE 802.11, was created as an alternative to the wired Ethernet standard, and it works by using radio waves to exchange data with nearby devices [26]. Considered the most widely used computer network in the world, it can connect almost all types of devices to a router, which connects them to the Internet and to each other. The most common frequencies in which Wi-Fi operates are the 2.4 GHz and the 5.0 GHz radio bands. Wi-Fi has a short range of about 90 meters outdoors and 45 meters indoors, and any type of obstacle, e.g. a wall, can impact it even more, although a few overlapping APs can somewhat fix this issue. Recently, Wi-Fi has achieved speeds of up to 9.6 Gbps [27]. The

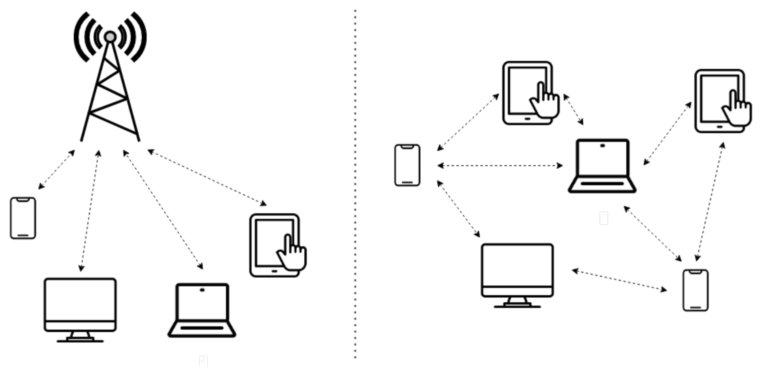


Figure 2.5: WLAN network configured in Infrastructure mode (on the left) and on *Ad hoc* mode (on the right)

typical data rate can also be fairly high (about 200 Mbps, for a strong signal), which helps explain its popularity: as of 2019, more than three billion Wi-Fi-enabled devices were annually shipped and ready to use.

Out of all the Wi-Fi standard versions on the market, three stand out: 802.11ac (branded as Wi-Fi 5) and 802.11ax (branded as Wi-Fi 6), launched respectively in 2014 and 2021, and Wi-Fi 7, whose release date is expected somewhere along 2024. Briefly, here are the differences between the three:

- Wi-Fi 5: Works exclusively on 5 GHz frequency and offers a maximum speed of up to 1 Gbps. It supports 8 channels, each of which presents a bandwidth of 80 MHz. This version greatly increased the use of beamforming: by detecting the approximate location of a device, it channels the Wi-Fi signal to that device, instead of sending the radio waves in every direction and resulting in better signal reception. The at-the-time modern feature of multi-user Multiple-Input Multiple-Output (MIMO) is particularly valuable when multiple users are online at the same time. Other advantages in comparison to older versions were the higher throughput as a consequence of higher bandwidths, the higher number of spatial streams (8), and the higher number of modulation schemes. The maximum range also increased by about 10 meters in comparison with the previous version, Wi-Fi 4.

- Wi-Fi 6: The biggest difference regarding the previous version is arguably the maximum speed; 9.6 Gbps [27]. Unlike Wi-Fi 5 but similarly to most other versions, Wi-Fi 6 supports dual frequency bands, and a subversion known as Wi-Fi 6e will also work in 6 GHz. This version introduced the Orthogonal Frequency-Division Multiple Access (OFDMA) concept, in both uplink and downlink directions, and once again, the number of streams increased, this time to 12. This version is also considered optimized for IoT. Each IoT-connected device can take a toll on a Wi-Fi network, but Wi-Fi 6 has been designed to be able to handle multiple devices while not compromising the speed, providing a much smoother continuous experience between different IoT applications.

- Wi-Fi 7: Like Wi-Fi 6e, it operates in three bands, but it is expected to more than quadruple speed up to 46 Gbps. Its worst-case scenario latency is one hundred times better than the previous one, and it provides up to five times more network capacity.

2.5.3.1 Final Comparison

Below, table 2.3 presents the main characteristics of different technologies. One for each family was chosen: LoRa, Wi-Fi 6, and Bluetooth BLE.

Specification	LoRa	Wi-Fi 6	BLE
Range (outdoors)	10 km	90 m	100 m
Data Rate	22 kbps	9.6 Gbps	1 Mbps
Frequency	868 MHz (Europe)	2.4 GHz	2.4 GHz
Power Consumption		5 ~20 W	0.01 ~ 0.5 W
Topology	Star	Star, Mesh	P2P, Star, Mesh, Broadcast
Module Cost	10€	5€	< 5€

Table 2.3: Comparison between LoRa, Wi-Fi 6 and BLE

2.6 Different Types of IoT Sensors

A sensor is an instrument designed to detect specific inputs from the physical environment. It then generates an output, which can be in the form of a readable display or an electronic signal. Sensors can be classified into numerous categories, with analog vs. digital sensors and active vs. passive sensors being two significant classifications worth considering.

Analog and digital sensors (Figure 2.6) differ in how they produce and represent measured data. Analog sensors generate an output signal that is continuously variable and proportional to the measured quantity. While they allow for precise measurements, they may be susceptible to noise and require additional analog-to-digital conversion for further processing and analysis. On the other hand, digital sensors use analog-to-digital converters (ADCs) to produce a discrete output signal that can be easily processed, stored, and transmitted.

Regarding active and passive sensors, the distinction lies in whether the sensor requires an external power source to operate (Figure 2.7). Active sensors require an internal power source, such as a battery or external power supply, to function. They generate and emit a signal to measure the desired parameter and detect changes in the environment, and provide enhanced accuracy and sensitivity, but limited battery life. Passive sensors do not require an internal power source: they respond to changes in the physical conditions or stimuli without actively emitting any energy themselves, making them simpler, smaller,

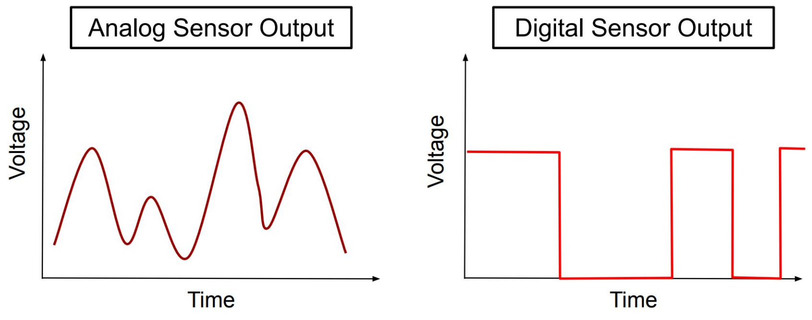


Figure 2.6: Analog sensor and digital sensor output, extracted from [28]

and more cost-effective. They are often used in applications where power constraints or size limitations are critical factors.

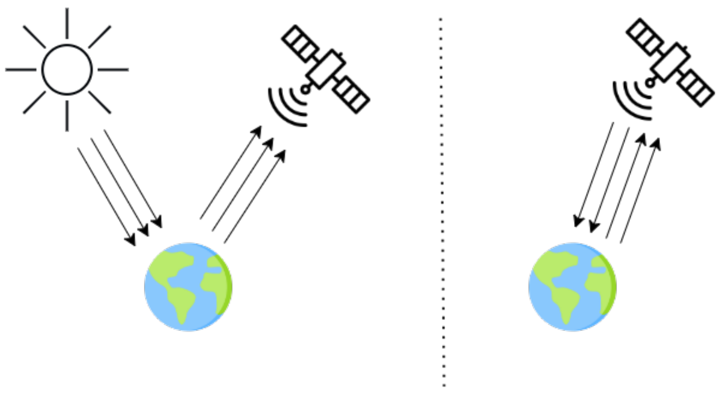


Figure 2.7: Passive sensing (on the left) and active sensing (on the right), adapted from [29]

In IoT, sensors act as the bridge between the physical and the digital world, measuring several physical parameters, capturing them and converting them into electronic signals, and optimizing systems across diverse domains such as smart cities, environmental monitoring, or healthcare. Below is a non-exhaustive list of some of the principal sensors in the IoT field:

1. Environmental Sensors

    (a) Temperature sensors work by converting temperature changes into electrical signals. They are typically composed of two metals that produce a voltage or resistance variation when exposed to temperature fluctuations. For a diode-based temperature sensor, the forward voltage (VBE) across the diode is negatively proportional to increasing temperature, resulting in a negative temperature coefficient (–mV/C°). This linear relationship between forward voltage and temperature allows diodes to be used as temperature measurement devices.

These sensors are used in environmental monitoring, medical devices, consumer electronics, etc.

(b) Humidity sensors detect moisture changes and can be resistive or capacitive. Capacitive sensors measure capacitance variations caused by moisture absorption or desorption on a sensitive layer between electrodes. Resistive sensors rely on a moisture-absorbing substrate that alters electrical resistance with humidity. These sensors convert physical changes into electrical signals, enabling accurate humidity measurements. Widely used in agriculture, meteorology, data centers, and HVAC systems, they ensure precise humidity monitoring and control.

(c) Pressure sensors convert mechanical pressure in gases or liquids into an electrical output signal. They typically employ a pressure-sensitive element, such as a strain gauge, that changes its electrical resistance in response to applied pressure. This change in resistance is then measured and converted into an electrical signal, providing an accurate representation of the pressure being measured. Pressure sensors are used extensively in machinery, automobiles, and aircraft.

(d) Water quality sensors acess and monitor different parameters, among which are pH, conductivity and dissolved solids. For instance, pH sensors utilize electrodes to measure the acidity or alkalinity of water, while some dissolved solids sensors work based on electrical conductivity principles, taking advantage of the fact that certain dissolved solids can conduct electricity. Additionally, there are other types of water quality sensors, such as turbidity sensors, which operate on different principles to assess water clarity. Water quality sensors enable accurate and real-time monitoring of water conditions and are used for a multitude of domains from environmental monitoring to drinking water management and wastewater treatment.

2. Imaging and Optical Sensors

(a) Image sensors consist of an array of light-sensitive pixels that detect and convert incoming photons into electrical charges, which are then transformed into a digital representation of the captured image. Different types of image sensors, such as CCD (Charge-Coupled Device) and CMOS (Complementary Metal-Oxide-Semiconductor), utilize distinct technologies to achieve this conversion process. Image sensors play a critical role in capturing visual information, enabling advancements in photography, computer vision, etc.

(b) Optical sensors respond to light of various wavelengths. Depending on the type of sensor, they work by either detecting the interruption in a beam of light or its reflection caused by the presence of the object. They can include photodiodes or phototransistors that convert light intensity into electrical signals and are widely used in fiber optics, laser measurement, and other fields where precise detection of light properties is required.

   (c) Proximity sensors detect the presence or absence of an object within a certain distance and work on various technologies, among which are infrared, capacitive, magnetic, or ultrasonic. Infrared sensors emit infrared light and measure the reflection or interruption of light to detect presence, while magnetic sensors use magnetic fields to sense the presence of ferrous objects. They play a crucial role in object detection and automation and can be found in elevators, parking lots, automobiles, and numerous others.

3. Position Sensors

   (a) Gyroscope sensors measure the rotation of an object and determine its angular velocity and orientation around a 3-axis system. They typically consist of a sensing element, such as a vibrating or oscillating mass, which experiences a deflection or displacement when subjected to angular motion. This deflection is then converted into electrical signals, providing data about the rotational movement. Gyroscope sensors are widely used in navigation systems, motion-tracking devices, and virtual reality.

   (b) Acceleration sensors provide information about the rate of change of velocity in a specific direction. Accelerometers work based on various sensing principles, such as piezoelectric or capacitive systems. Piezoelectric accelerometers utilize crystals that generate an electrical charge when subjected to acceleration forces. Capacitive accelerometers measure changes in capacitance between two plates due to acceleration. Accelerometers find applications in areas such as motion sensing, aerospace and aviation, and consumer electronics.

Chapter 3 will delve into a detailed exploration of some of these sensors in specific IoT applications.

2.7 Summary

In this chapter, an introduction to the concept of the Internet of Things was provided. The discussion encompassed key IoT protocols, including MQTT, CoAP, and HTTP, along with an examination of three prominent time series databases. Furthermore, a comprehensive analysis of diverse IoT frameworks was conducted, highlighting their distinctions. The chapter also presented a review of essential wireless communication technologies. Finally, an overview of IoT sensors was presented, offering insights into their significance and applications.

Chapter 3

Smart Metering Systems

While smart meters have gained significant attention in recent years, their application in the water reuse field remains largely unexplored, presenting us with a pioneering opportunity. In the section that follows, we will delve into the state of the art in smart metering, highlighting relevant examples from related fields in order to provide context and enhance this system design.

3.1 Smart water metering

With the growing popularity of smart metering, there is a shift towards empowering consumers to reduce unnecessary waste and optimize their water consumption, rather than solely relying on water utility companies. In 2016, the Water Research Foundation (WRF) published an updated and expanded assessment of water use [30], building upon their initial report in 1999. The report reveals a significant decline of approximately 22% in water use across the residential sector compared to 1999, despite increasing populations. However, even with this decrease, leakage remains a concern, accounting for 13% of indoor household water use, which translates to an average daily absolute value of about 67.4 liters per household.

Several studies highlight the importance of detecting leaks [31], [32]. Various techniques have been proposed to identify and address leaks. For instance, Hu et al. [33] propose the DBSCAN-MFCN (Density-Based Spatial Clustering of Applications with Noise and Multiscale Fully Convolutional Networks) approach, which combines density-based spatial clustering (DBSCAN) and multiscale fully convolutional networks (MFCN) to detect leaks. Liu et al. [32] utilize WSNs installed on pipelines, collecting and transmitting data through 4G, and detecting leaks using signal features constructed from intrinsic mode function, approximate entropy, and principal component analysis. Smart meters, however, are one of the most commonly used methods for leakage detection.

Amir et al. [34] presented a solution to Indonesia's current mechanical and labor-intensive, as well as often inaccurate, method of using smart meters: the use of a flow sensor. This flow sensor is not only used to measure and record the volume of water consumed but also allows the customer to determine the desired amount of water, with the smart meter closing the valve once the chosen amount is reached. Amir et al.'s proposal is implemented using an Arduino microcontroller, and the data is sent wirelessly to an Android device. The wireless transceiver module integrated with a microcontroller using NodeMCU is compatible with smartphones and uses UART. The data is accessible via a specific IP address that the consumer can access.

Tasic et al. take Amir et al.'s approach one step further by proposing a self-powered water meter [35]. However, their system is limited to small pipes for indoor use. On the contrary, in [36], an electromagnetic and piezoelectric energy harvester-powered smart meter is presented, but it is designed in a way that is not suitable for indoor use.

Li et al. [37] propose a system that combines the advantages of both previous approaches, making it suitable for indoor use while being scalable to larger systems. Their system uses a water turbine generator as both a flow measurement sensor and a power generator, eliminating the need for regular battery replacement and turning the system into a self-powered one. The Water Turbine Generator (WTG) generates a voltage signal when water flows through it. This voltage signal is then rectified and connected to a programmable single-pole double-throw (SPDT) switch. The microcontroller receives the voltage signal for sampling and flow rate estimation, while in other cases, the signal is directed to the voltage regulator, charging circuit, and rechargeable batteries (Figure 3.1). The microcontroller is powered by these batteries, and once fully charged, they are disconnected from the charging circuit. Additionally, this system can be applied to other fluids as long as the fluid's viscosity is not so high that it impairs the measurements.

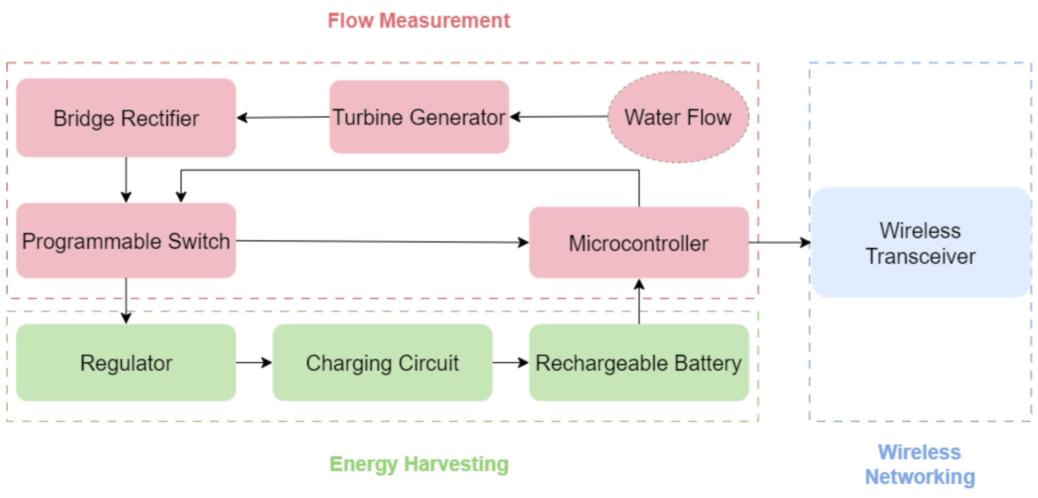


Figure 3.1: Block diagram adapted from [37]

In addition to addressing the critical issue of leakages, water metering encompasses a broader range of concerns. Consumers not only seek to detect and mitigate leakages but also express a growing interest in gaining insights into their water consumption patterns

[38] and water quality [39], optimizing their water usage efficiency [40] and exploring opportunities for water reuse. By delving beyond leak detection, water metering endeavors to cater to consumer needs for enhanced understanding, sustainable practices, and the effective management of this vital resource.

In [38], recurrent routine behaviors are automatically detected in smart water meter data, specifically focusing on short subsequences of variable length, unlike previous motif discovery algorithms that primarily focused on exact matches. By capturing and analyzing these recurrent routine behaviors, the algorithm provides insights into regular water use activities, enabling experts to gain a deeper understanding of water consumption patterns.

Mezzera et al. [39] propose a WSN designed for the management of drinking water. The key innovation lies in the compact electronic board, which enables analog conditioning, acquisition, and cloud-based processing of various water parameters, e.g. pH, temperature, flow, and deposit thickness, in real-time. Additionally, the system addresses practical concerns such as battery recharge, energy harvesting from water velocity (at 15 mA), data logging, and transmission through the Long-Term Evolution (LTE) network.

3.2 Data acquisition, storage and visualization

Most smart meters try to tackle leaking issues by detecting normal consumption usage, extrapolating trends and patterns, and comparing real-time consumption with expected values to detect possible leaks. Pereira et al. [41] utilized the SustDataED2 dataset, which includes smart meter data from a single household, along with aggregated and individual appliance consumption, current and voltage waveforms, and ON-OFF transition timestamps, to develop a Machine Learning algorithm that identifies and disaggregates consumption of different appliances.

In IoT applications, the MQTT protocol is commonly used for data communication [42]. Recently, there has been a focus on ensuring secure communications in MQTT [43]. For example, in the study by Rachmad Atmoko et al., [44], MQTT was employed as the communication protocol between temperature and humidity sensors and a MySQL database. However, the choice of communication protocol depends on the specific project requirements. HTTP and CoAP are also frequently used in IoT applications [43] [45].

Sayed et al. [46] implemented a smart meter system with the goal of measuring and displaying energy consumption metrics. In order to do so, an Arduino Uno was used, as well as an ESP8266 board operating as a Wi-Fi module, programmed to send the data to the router that, in turn, sends it to a server to ultimately store it and display it. The Arduino is powered through a mobile battery charger which converts 240V AC into 5V DC, with a 1A output current. The calibration curves of the inputs and outputs of the voltage and current transducers are obtained using a simple linear interpolation method. This smart meter was then tested with various loads, among which were a light bulb and a fridge. After acquiring the signals acquired from voltage and current transducers, these were processed and converted back to analog values in order to calculate all the relevant variables, before being displayed on the LCD (Liquid Crystal Display) and sent through Wi-Fi to the EmomCMS IP for remote monitoring. The flowchart in Figure 3.2 illustrates how this process is carried out.

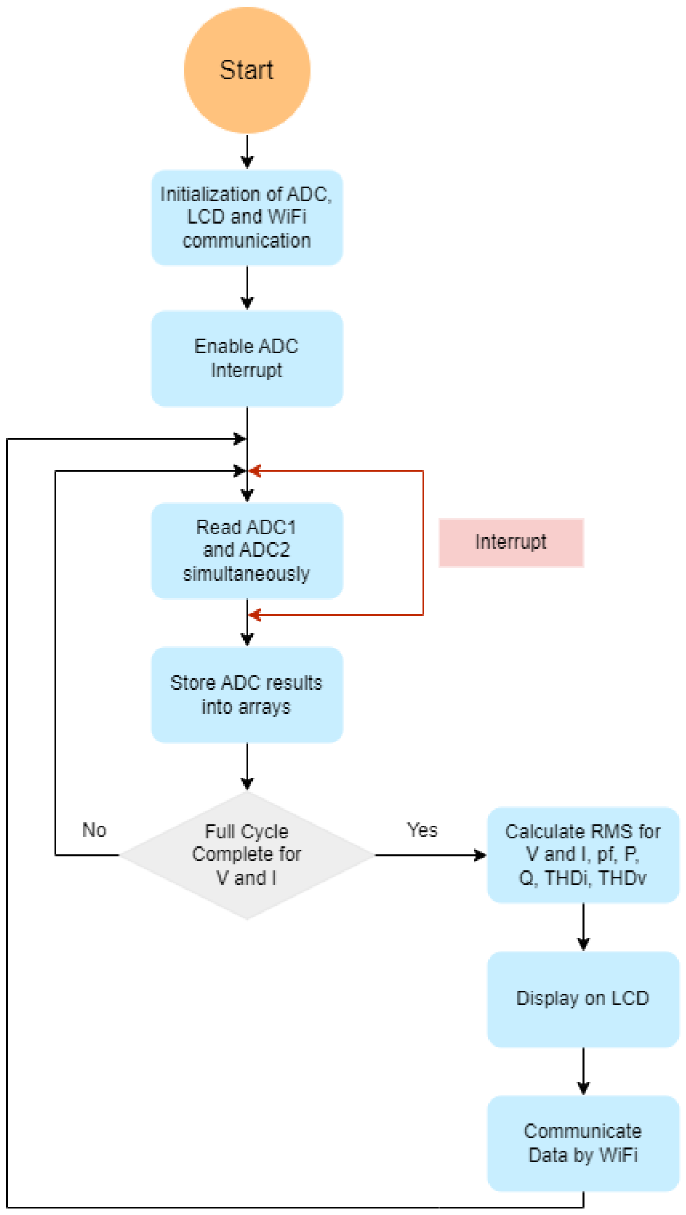

Start
Initialization of ADC, LCD and WiFi communication
Enable ADC Interrupt
Read ADC1 and ADC2 simultaneously
Interrupt
Store ADC results into arrays
Full Cycle Complete for V and I
No
Yes
Calculate RMS for V and I, pf, P, Q, THDi, THDv
Display on LCD
Communicate Data by WiFi

Figure 3.2: Flowchart for signal acquisition, processing and communication adapted from [46]

Sayed et al. report that a slight delay between the LCD and the cloud's display was noticed, a consequence of the time it took for data to be transmitted through the Internet. This has a negative impact on the retrieval of real-time results.

Suresh et al. [47] present a novel approach to Water-Meter reading based on IoT that, together with a smartphone app, helps improve water consumption management and allows for the detection of leakages and water tampering, as well as individual consumption tracking. Unlike commercial methodologies, their system uses low-cost IoT hardware and regular smartphones, optimized for areas with poor or unreliable mobile networks and limited investment in infrastructure.

The key hardware component, the Electronic Interface Module (EIM), is designed to be cost-effective and efficient. It includes a low-cost CPU, a real-time clock, flash memory for data logging, an Ethernet stack and port for communication over TCP/IP network, and a built-in battery, which will require periodic charging, as well as the correspondent charging adapter. The app was developed with the goal of letting the user keep updated consumption records regarding both real-time and past data, gaining insights into their consumption pattern, and securely updating the data to the utility's central database after login authentication.

This system helps both utilities and consumers: for utilities, it reduces the overhead costs associated with manual meter readings and leads to faster response times for incidents such as pipeline leakages. For consumers, it helps them take proactive measures to conserve water and understand consumption trends. Up to 14% of water was saved merely by corrective actions and replacements. The water consumption across multiple locations can be seen in Figure 3.3.

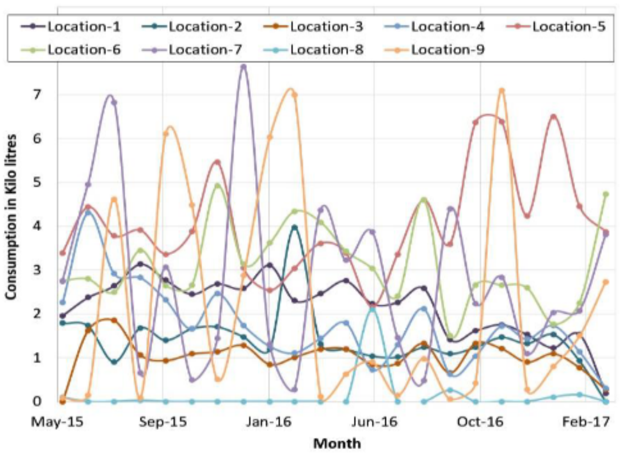


Figure 3.3: Daily water consumption for each month, extracted from [47]

Modi et al. [48] bring forward a cost-effective, open-source smart meter monitoring and controlling system that measures power consumption using MQTT. The process begins when the smart meter connects to the available local network using service set identifier (SSID) and API keys. The power values are sent to the server, processed, and stored. An Android app retrieves the data and can be used by the user both for monitoring and billing information and communicates with the smart meter through HTTP requests and APIs. The values in the server are continually updated until there is suspicion of meter tampering, in which case a flag, provided by power authorities, is activated and alerts the user, and adequate measures begin to take place. An illustrative block diagram can be seen in Figure 3.4.

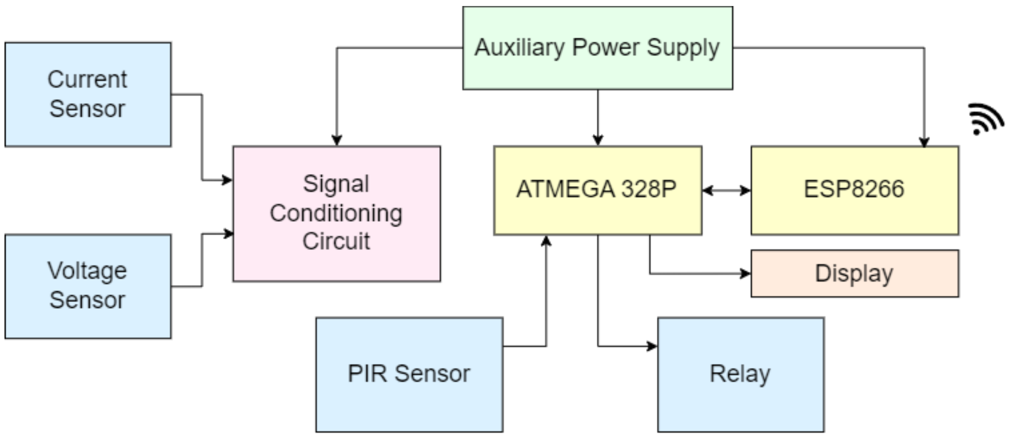


Figure 3.4: Proposed Block Diagram adapted from [48]

The chosen hardware includes an Arduino Uno Wi-Fi microcontroller, which offers an ADC sampling rate of 9615 Hz that can calculate precise values, the aforementioned LEM sensors (one for voltage measurements and one for current measurements, from which the power is calculated), and an integrated ESP32 with a built-in antenna, which is used to communicate via Wi-Fi with the router and sending data to the server.

One other area which relies in abundance on smart meters and IoT, outside of smart meters, is health care, with several examples of sensors being used in health monitoring applications [49], [50]. Kadir et al. [51] focus on a different area - not consumption monitoring, but health monitoring. Their system comprises edge devices that record patients' data, e.g. temperature and heart rate, and is dispatched wirelessly to a Raspberry Pi. The Raspberry Pi acts as an intermediary between the microcontroller and InfluxDB to provide real-time health monitoring and analysis.

Grafana is then integrated, as mentioned in Chapter 2, for visualization purposes. Additionally, a mobile app was developed so that users can access their data on the go, and receives alerts when it detects anomalies.

3.3 Proposed System

The proposed system exhibits similarities to several articles cited in this dissertation. For instance, Sayed et al.'s solution [46] also relies on wireless communication for data transmission through an Arduino family board. However, unlike Sayed et al.'s approach, the proposed system does not require an additional ESP8266 module since the Nano 33 IoT board has built-in Wi-Fi capability.

In contrast to Suresh et al.'s proposal [47], which directly sends data from the EIM to a smartphone app, this dissertation aims to store the data in an external database before presenting it to the user. This approach allows for data cleaning and processing, enhancing the overall data quality.

The project described in Modi et al.'s work [48] shares many similarities with the system outlined in this dissertation. Although the data-gathering methods may differ due to monitoring different variables, the overall system closely aligns with the anticipated implementation described in this document. Instead of using a system shutdown approach when a value exceeds a threshold, the proposed system will notify the user of unusually high or low water consumption levels.

Additionally, the system shares Mezzera et al.'s [39] features of cloud-based processing of water parameters in real-time and battery recharge concerns, and it aligns with Li et al.'s system [37] by proposing the use of a water turbine as a power generator.

In essence, the proposed system consists of two sensors - a Total Dissolves Solids (TDS) sensor that measures water quality and a flow sensor that measures flow quantity, connected to an Arduino Nano 33 IoT. The microcontroller communicates with a Raspberry Pi via Wi-Fi, and when the Raspberry Pi receives the sensor-gathered data, it sends it over to InfluxDB, which displays it in Grafana and stores it in AWS.

As for the power generation, a turbine will be connected to the system, and when water passes through it (as long as it surpasses a certain threshold), it outputs 5V DC, which are then used as the input for a battery charger which feeds the battery responsible for powering the Arduino. Figure 3.5 provides a visual representation of the system's block diagram.

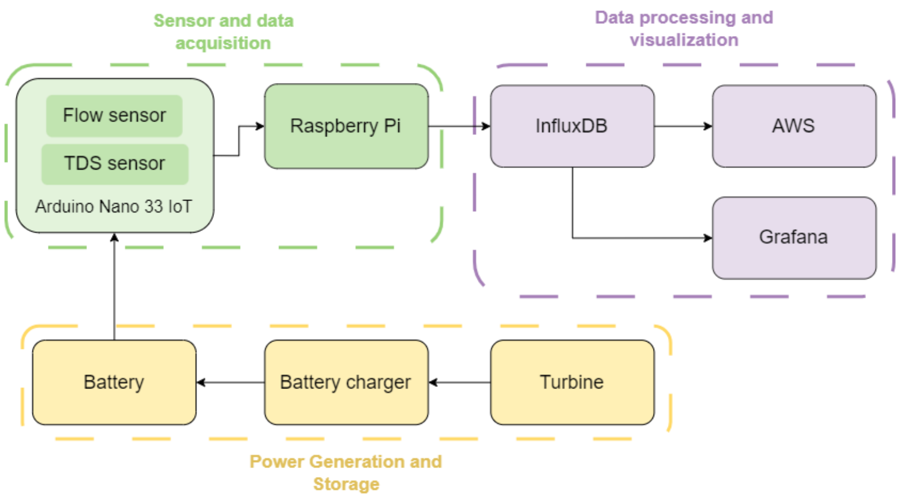


Figure 3.5: System Functional Architecture Block Diagram

The system becomes active upon the detection of water flow by the water flow sensor. Once triggered, the Arduino wakes up from sleep mode, initializing the entire system. Immediately, the Arduino attempts to establish a Wi-Fi connection to the same network as the Raspberry Pi. If the connection is successful, a TCP socket is opened, and the Raspberry Pi awaits the reception of TDS and flow values from the Arduino. These values are stored in a database, with separate columns for each type of measurement. Concurrently, Grafana constantly updates itself and retrieves new data from the database. When more than 120s pass without water being detected, it is considered the event is closed, and the Arduino goes back to sleep until a new water detection occurs. This process is depicted in Figure 3.6.

Simultaneously, the turbine, which is also exposed to the flowing water, generates 5V DC. This electrical output is directed to a battery charger, which charges the battery responsible for powering the Arduino.

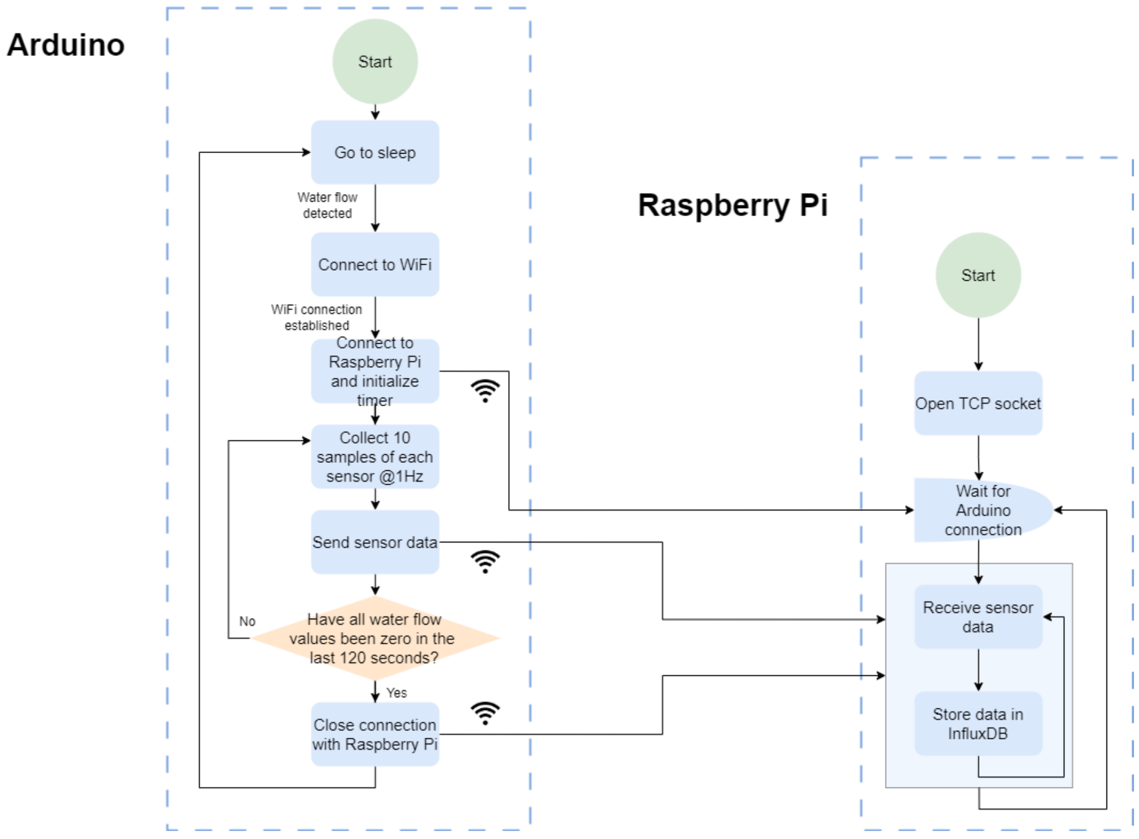

Arduino
Start
Go to sleep
Water flow detected
Connect to WiFi
WiFi connection established
Connect to Raspberry Pi and initialize timer
Collect 10 samples of each sensor @1Hz
Send sensor data
Have all water flow values been zero in the last 120 seconds?
No
Yes
Close connection with Raspberry Pi
Raspberry Pi
Start
Open TCP socket
Wait for Arduino connection
Receive sensor data
Store data in InfluxDB

Figure 3.6: System Architecture Flowchart

3.4 Summary

This chapter explored the state of the art in smart metering systems, specifically focusing on smart water metering. The use of smart meters for leak detection was discussed, along with examples of innovative approaches such as flow sensors and self-powered water meters. The chapter also discussed the broader concerns of water metering, including consumer demand for insights into water consumption patterns, water quality monitoring, and opportunities for water reuse. It explored studies that utilize smart water meter data to detect recurrent routine behaviors and gain a deeper understanding of water consumption patterns.

Several examples of smart meter systems using Arduino, ESP8266, and Raspberry Pi were provided, highlighting their capabilities in measuring and displaying energy consumption metrics. The proposed system architecture incorporates elements from existing literature, such as wireless communication, data storage in an external database, and the use of a water turbine for power generation.

Chapter 4

Comparative Study Between Different Microcontroller Options

A microcontroller is an integrated circuit device that controls other components of a system. In this case, based on the values retrieved by the sensors, the microcontroller commands the consequent actions that take place. Three relevant microcontrollers for a smart metering system are the Arduino Nano 33 IoT, the ESP32, and the ESP8266, presented below. These microcontrollers have built-in Wi-Fi capabilities, which are oftentimes crucial in IoT devices, and are three of the most cost-effective options.

4.1 Arduino Nano 33 IoT

Arduino is both an open-source cross-development environment and a physical circuit board. The cross-development environment allows editing C or C++ programs, compiling and downloading them to the board through a USB cable. Due to Arduino's simplicity, it is used by professionals and amateurs alike, advertising itself as a low-cost platform that allows the development of non-complex projects using sensors and actuators.

The boards are equipped with digital and analog I/O pins and their functionalities can be expanded through "shields" (e.g. for Wi-Fi connection or LCDs).

Arduino Nano 33 IoT, seen in Figure 4.1, is a type of Arduino specifically designed for IoT applications and M2M communication.

These are some of its most relevant characteristics in the context of IoT:

- Small size of 45mm x 18mm and form factor enable seamless integration into embedded systems
- Powerful processor 32-bit SAMD21 Cortex®-M0+ with a clock speed of 48MHz.
- Integrated Wi-Fi (u-blox NINA-W102) and BLE connectivity, facilitating wireless communication

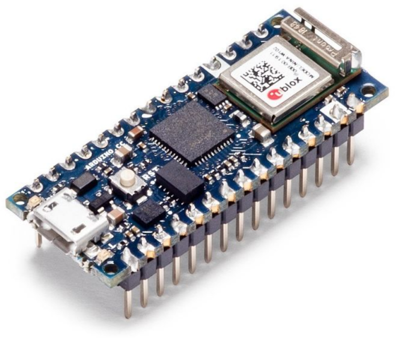


Figure 4.1: Arduino Nano 33 IoT Board [52]

- Integrated Real-Time Clock (RTC) for timekeeping purposes
- Full TCP/IP stack, allowing integration into IP-based networks
- 256KB of Flash Memory and 32KB of SRAM
- Low power consumption modes ensuring extended battery life
- Built-in cryptographic module that supports AES and SHA-256, used for encryption and hashing purposes
- Consequently to the last point, WPA and WPA2 support, ensuring secure communication over Wi-Fi networks
- Diverse set of built-in sensors, e.g. accelerometers and gyroscopes and 22 I/O pins (14 digital I/O pins and 8 analog input pins) for connecting external devices

Figure 4.2 shows the Arduino Nano 33 IoT's pinout.

4.1.1 Development Frameworks

While Arduino IDE offers a straightforward platform to work with, there are several advantages in choosing a real-time operating system (RTOS) specifically designed for IoT applications. RTOSs offer multitasking support, allowing the user to run tasks concurrently and attribute a higher priority to time-critical operations. They provide mechanisms for efficient resource management such as task scheduling and memory allocation. Their modularity and scalability mean that only the necessary components will be used, offering the ability to add or remove features as needed. This flexibility ensures efficient resource usage, reduces overhead, and enables the development of sophisticated and robust IoT applications. Below, Arduino IDE is compared with two of the most popular RTOS frameworks.

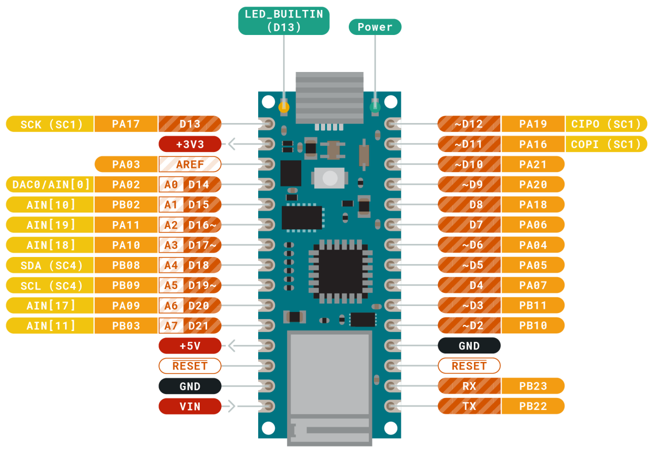


Figure 4.2: Arduino Nano 33 IoT Pinout [52]

- Arduino IDE: The Arduino IDE (Integrated Development Environment) is the official software provided by Arduino for programming their boards. It offers a beginner-friendly interface and a simplified programming language based on C/C++. The extensive community support and rich library ecosystem make it a popular choice among Arduino users.

- Zephyr: Zephyr is an open-source RTOS designed for resource-constrained devices. It supports a wide range of microcontrollers, including the SAMD21 used in the Arduino Nano 33 IoT. Zephyr offers a scalable and modular architecture and emphasizes security and low power consumption.

- RIOT: RIOT is an open-source RTOS for IoT and offers features such as networking protocols, low-power operation, and real-time capabilities, and focuses on portability and interoperability.

4.1.2 Power Consumption Management

The Arduino Nano 33 IoT board offers inherent low-power features that position it as a suitable choice for energy-efficient systems. Notably, its 3.3V operating voltage, in contrast to the 5V typically found in other Arduino boards, facilitates reduced power dissipation by minimizing voltage drop across components and mitigating heat generation. However, if further power reduction is desired, additional low-power solutions can be employed. While lowering the clock speed or eliminating unnecessary hardware components are two feasible options, leveraging sleep modes stands out as a straightforward and highly efficient approach to achieve significant power savings.

The Power Manager (PM) of the SAM D21 microcontroller comprises three modules: the synchronous clock controller, sleep controller, and reset controller (Figure 4.3). These modules handle clock distribution, low-power mode supervision, and reset management.

The PM receives clocks from GCLK, which can be divided down for power savings, and distributes them to the CPU, high-speed bus, and peripheral bus. The Advanced High-Performance Bus (AHB) clock remains synchronized with the CPU, while peripheral bus clocks are typically pre-scaled for lower-frequency operation. The PM also allows selective distribution of AHB and Advanced Peripheral Bus (APB) clocks to chosen peripherals, further reducing power consumption. Additionally, the PM enables runtime switching of clock frequencies and oversees lower-power STANDBY and IDLE modes (Table 4.1).

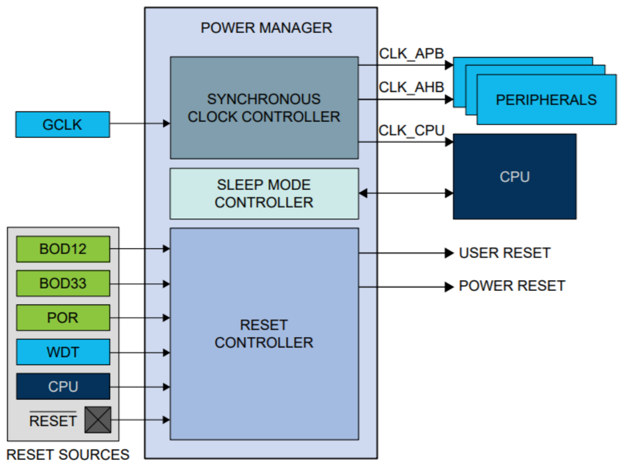


Figure 4.3: Power Manager Architecture in a Arm® Cortex®- M0+ MCU [1]

STANDBY mode is the lowest power consumption mode available. It disables all system clocks and sets the voltage regulators to low power mode. The device can be configured to stop oscillators completely, keep them running, or activate them based on peripheral requests. The device enters STANDBY mode when a Wait For Interrupt (WFI) instruction is executed with the appropriate settings. An interrupt or the Watchdog Timer (WDT) can wake up the device from STANDBY mode. Before entering Standby mode, it is important to disable several clocks and peripherals to prevent overloading the voltage regulator. To avoid exceeding its capacity, it is advisable to configure the peripherals in a way that keeps the total power consumption supplied by the internal regulator below 50µA in low power mode.

IDLE mode, on the other hand, stops the CPU clock while keeping the peripherals running, thus its fastest wake-up time. There are three levels of IDLE mode, each selectively turning off unnecessary clocks to save power. The device enters IDLE mode after executing a WFI instruction with the suitable settings. An interrupt or a WDT timeout can bring the device back to active mode from IDLE mode.

To better understand Table 4.1 it is important to distinguish each clock.

- The CPU clock is specifically routed to the CPU itself. Halting its stops the execution of instructions, resulting in a paused or idle state of the CPU. The CPU clock's frequency follows the equation $f_{CPU} = \frac{f_{main}}{2^{CPUDIV}}$.

Sleep Mode	**CPU Clock**	**AHB Clock**	**APB Clock**	**Oscillators**				**Main Clock**	**Reg. Mode**	**RAM Mode**
				ONDEMAND = 0		**ONDEMAND**				
				RUN STDBY = 0	**RUN STDBY = 1**	**RUN STDBY = 0**	**RUN STDBY = 1**			
Idle 0	Stop	Run	Run	Run	Run	Run on request	Run on request	Run	Normal	Normal
Idle 1	Stop	Stop	Run	Run	Run	Run on request	Run on request	Run	Normal	Normal
Idle 2	Stop	Stop	Stop	Run	Run	Run on request	Run on request	Run	Normal	Normal
Standby	Stop	Stop	Stop	Stop	Run	Stop	Run on request	Stop	Low power	Low power

Table 4.1: Sleep mode differences in the SAM D21 family [1]

- The AHB clock shares its frequency with the CPU clock, but can run even if the CPU clock is turned off. This way, the smooth functioning of the peripherals that rely on it is ensured.

- The APB clock is the PM's bus clock and its state can be controlled by the PM. If it is disabled it can only be re-enabled by performing a reset. Various peripheral modules are connected to the APB bus, including the ADC clock, the Pulse Width Modulation (PWM) module and the I/O pins of the microcontroller.

- The main clock serves as the primary source for synchronous clocks for the CPU, AHB, and APB modules. Its frequency is determined by an 8-bit prescaler, which allows the synchronization across different modules: these can run from any tapping off this prescaler as long as the condition $f_{CPU} \geq f_{APB_X}$ is met.

SAM D21's datasheet provides power consumption values for a multitude of scenarios and variants, summarized in Table 4.2. Arduino Nano 33 IoT's microcontroller is a SAMD21G18A, which means that it belongs to variant A. These measurements are valid for an input voltage of 3.3 V_{DDIN}, with the XOSC (crystal oscillator) stopped, the XOSC32K (32kHz crystal oscillator) running with an external crystal, and the DFLL48M using the XOSC32K as a reference and running at 48MHz. As for the clocks, the main clock source is the DFLL48M, and the CPU and AHB clocks are undivided. Some specific AHB module clocks and peripheral clocks (PM, SYSCTRL, and RTC) are running, while all other clocks are stopped. The definition of wake-up time considered was the time from the edge of the wake-up signal to the execution of the first instruction fetched in flash.

Overall, power consumption gradually decreases as we move from active mode to each idle mode and, finally, the standby mode. However, it is important to consider the trade-off between power savings and wake-up time, as the wake-up time increases with each idle mode. Standby mode offers the most significant power savings, especially when XOSC32K

Mode		Ta	Min.	Typ.	Max.	Wake-up time
Idle 0		25ºC	3.11 mA	3.37 mA	3.64 mA	-
		85ºC	3.24 mA	3.48 mA	3.75 mA	-
Idle 0		25ºC	1.89 mA	2.04 mA	2.20 mA	4.0μs
		85ºC	1.98 mA	2.14 mA	2.33 mA	4.0μs
Idle 1		25ºC	1.34 mA	1.46 mA	1.58 mA	12.1μs
		85ºC	1.41 mA	1.55 mA	1.71 mA	13.6μs
Idle 2		25ºC	1.07 mA	1.17 mA	1.28 mA	13.0μs
		85ºC	1.13 mA	1.27 mA	1.40 mA	14.5μs
Standby	XOSC32K running RTC running at 1kHz	25ºC	-	4.06 μA	12.8 μA	19.6μs
		85ºC	-	55.2 μA	100 μA	19.7μs
	XOSC32K and RTC stopped	25ºC	-	2.70 μA	12.2 μA	-
		85ºC	-	55.3 μA	100 μA	-

Table 4.2: Current consumption and wake-up time in each low-power mode of the SAM21 processor in the Arduino Nano 33 IoT board (extracted from the datasheet, [1])

and RTC are stopped, making it suitable for applications requiring long periods of inactivity.

4.2 ESP8266

The ESP8266, presented in Figure 4.4, is a low-cost microcontroller developed by Espressif Systems that allows direct connection to Wi-Fi. Similarly to the previous case, this system includes wireless networking, allowing the user to quickly control and monitor devices remotely via Wi-Fi.

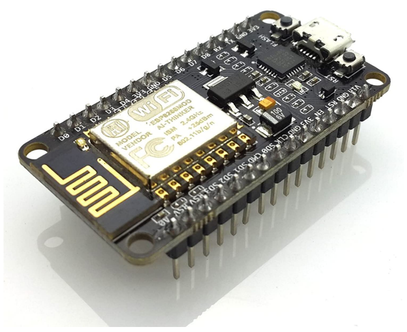


Figure 4.4: ESP8266 NodeMCU Board [53]

The onboard processing allows this microcontroller to be integrated with general purpose inputs/outputs (GPIOs) with a low loading time, and it was designed to occupy the least possible amount of space. It is suggested that the ESP8266 ESP-12E NodeMCU Kit development board is used to facilitate using the system. Some of the most relevant characteristics of the NodeMCU-ESP8266 when it comes to its use in the domain of IoT are the following:

- Small size of 49mm x 24.5mm, ensuring it is suitable for applications with size constraints;
- 32-bit microcontroller: Tensilica RISC CPU Xtensa LX106, running at 80 MHz with various power-saving modes;
- Wi-Fi connectivity with Station mode, SoftAP (access point) mode, and Statio+SoftAP mode, allowing flexible Wi-Fi connectivity options
- Full TCP/IP stack, enabling seamless and reliable integration into IP-based networks;
- 17 GPIOs that can be used to interface with sensors, actuators, and displays and provide flexibility for integrating the ESP8266 with other devices;

- Diverse set of peripherals, including ADC, Serial communication (e.g., UART, SPI, I2C), PWM output for, e.g., generating analog-like signals, and I2S, commonly used for audio applications;
- 4 MB of flash memory for data storage;
- Arduino Integration and the possibility of programming ESP8266 using the Arduino IDE, as well as all the dedicated libraries;
- WPA and WPA2 support, ensuring secure communication over Wi-Fi networks;
- Cost-effectiveness.

The ESP8266 series of modules are powered by the ESP8266EX chip, which contains all the necessary components for Wi-Fi connectivity and microcontroller functionality. The ESP8266 series is usually made up of the ESO8266EX chip along with additional components such as voltage regulators, antennas, and flash memory. Figure 4.5 shows the functional block diagram of this chip.

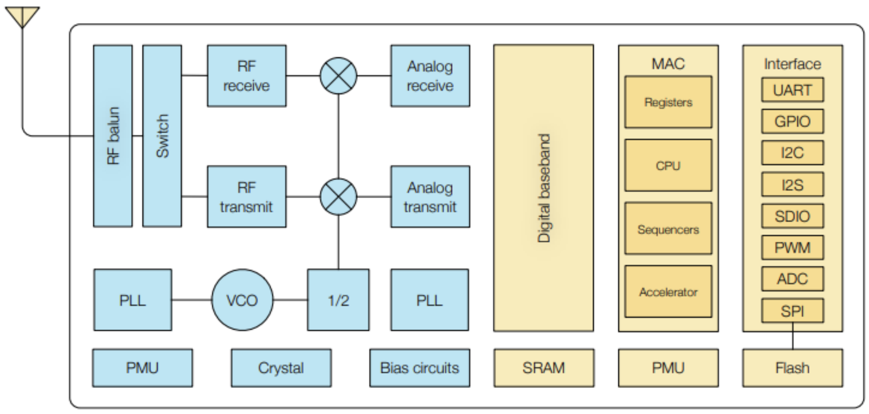


Figure 4.5: Functional Block Diagram of ESP8266EX [2]

4.2.1 Development Frameworks

The increasing popularity of the ESP8266 module in the IoT community led to the development of various frameworks that simplify its programming. Two of the most widely known frameworks are Arduino IDE, mentioned in Section 4.1, and NodeMCU. Arduino IDE can be integrated with FreeRTOS for smoother functioning.

- Arduino IDE + FreeRTOS: FreeRTOS is an open-source RTOS that provides multitasking capabilities. It facilitates task managing, scheduling, and application synchronization, similarly to Zephyr and RIOT (Section 4.1). By integrating FreeRTOS with the Arduino IDE one can create and define multiple concurrent tasks using a priority system and execution schedule, which is useful to handle time-critical operations such as sensor data acquisition. These tasks can communicate with each other using synchronization mechanisms provided by FreeRTOS, such as queues, semaphores, and mutexes.

- NodeMCU is an open-source development kit based on the programming language Lua. It offers features such as GPIO control, PWM, I2C, SPI, and Wi-Fi connectivity, all accessible through the Lua programming language. One of the significant advantages of NodeMCU is its ability to quickly prototype and test ideas using its interactive shell, which allows you to execute Lua code directly on the ESP8266. A popular choice for an IDE is ESPlorer, specifically designed for NodeMCU development.

4.2.2 Power Consumption Management

The ESP8266 includes three types of sleep modes [54], suited for power-consumption conscious designs:

1. Modem Sleep: this is the default sleep mode. The CPU is operational, as well as the low-power peripherals, and the board is active when connected to an AP. It is mostly used in applications that require the CPU to be working such as in PWM or I2S applications. In DTIM3, for instance, it maintains a sleep of 300 ms with a wakeup of 3 ms cycle to receive AP's beacon packages at interval and it consumes about 15 mA. This sleep mode is divided into two categories: Minimum Modem, where the device wakes up every Delivery Traffic Indication Message (DTIM) to receive the beacon and has a lower power saving capacity; and Maximum Modem, which wakes up at every listen interval but might lose data in the process.

2. Light Sleep: similar to Modem Sleep, but it additionally turns off the system clock and suspends the CPU, which can then wake up with a signal sent to a predefined code. It keeps the RTC memory and peripherals running and consumes about 0.4 mA. During light sleep, since the CPU is suspended, it will not respond to signals and interrupts from peripheral hardware interfaces, thus the only wakeup mechanism is via an external preconfigured GPIO which can only be enabled by level triggers. The external wakeup function takes as parameters the IO serial number and the trigger mode (LOW level or HIGH level).

3. Deep Sleep: everything apart from the RTC is shut down, including Wi-Fi and the CPU. There are two ways to activate it, either after a predefined period of time or when the RST button is pressed. It consumes about 20µA. It is generally used by applications with long time lags between data transmission, e.g. a sensor that measures data every 100s and sleeps for 300s. The function receives the *time_in_us*

parameter which makes the chip wake up at regular intervals when non null. Independently of the value passed to this parameter, the chip can be woken up and initialized by a low-level pulse generated via an external IO.

It is important to notice that there is a discrepancy of the nomenclature between the different states in Espressif's ESP8266 Low Power Solutions [54] and in ESP8266EX's datasheet [2]. In reality, the "sleep mode" mentioned in ESP8266EX's datasheet corresponds to two modes, "modem sleep" and "light sleep".

Item		Modem-Sleep	Light-sleep	Deep-sleep
Wi-Fi		OFF	OFF	OFF
System clock		ON	OFF	OFF
RTC		ON	ON	ON
CPU		ON	Pending	OFF
Susbtrate current		15 mA	0.4 mA	~ 20 μA
Average Current	DTIM = 1	16.2 mA	1.8 mA	-
	DTIM = 3	15.4 mA	0.9 mA	
	DTIM = 10	15.2 mA	0.55 mA	

Table 4.3: Differences between ESP8266 Sleep Modes [2]

Here is how each step is carried out:

1. In the Off state, when CHIP_PU is low and the RTC is disabled, all registers are cleared. In deep-sleep mode, only the RTC remains powered on while the rest of the chip is powered off. The RTC's recovery memory can store essential Wi-Fi connection information.

2. During Sleep mode, only the RTC operates, and the crystal oscillator is disabled. Any wake-up events such as MAC, host, RTC timer, or external interrupts can trigger the chip to enter the wakeup mode.

3. In the Wakeup state, the system transitions from the sleep states to the PWR mode. The crystal oscillator and the phase-locked loops (PLLs) are enabled.

4. During CPU On, the high-speed clock becomes operational and is distributed to each enabled block according to the clock control register. The system implements lower-level clock gating at the block level, including the CPU, which can be turned off using the WAITI instruction while the system remains powered on.

Figure 4.6 illustrates ESP8266EX's sleep modes and the transitions between them.

The DTIM interval determines how often the chip wakes up to receive multicast or broadcast messages. It is expressed as an integer number of beacon intervals. This means

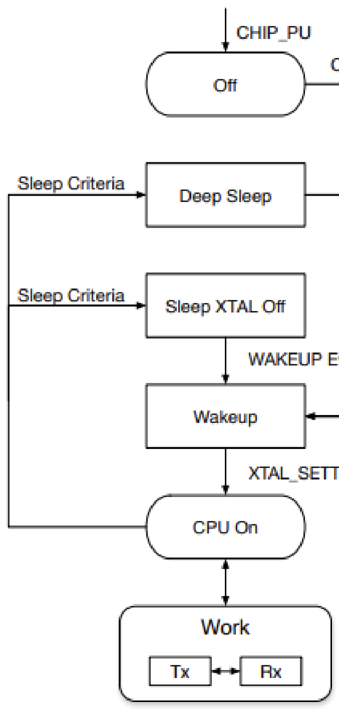


Figure 4.6: Power Management Flowchart in the ESP8266EX [2]

that, every x beacon intervals, the node wakes up for the beacon and stays awake to receive possible broadcast or multicast frames queued in the AP. A lower DTIM means the chip wakes up more frequently. Similarly, the beacon interval determines the frequency at which the Access Point sends beacon frames to announce its presence. . All nodes connected to an AP need to wake up regularly to receive the beacon, synchronize with it, maintain the connection, and check whether they should stay awake, in case there are more frames to receive, or return to sleep state, if activated. The power consumption when in Light Sleep and Modem Sleep depends on the set DTIM and the beacon interval, namely, the lower these intervals are, the higher the power consumption. Deep Sleep, on the other hand, completely turns off the Wi-Fi communication, which implies breaking the connection to the AP, and is therefore not dependent on these two factors. Its power consumption equation is given by:

$$I = I_{deep\_sleep} + \frac{20mA \times 100ms_{wakeup} + I_{RX} \times 1s + I_{TX} \times 0.1s}{300s} \tag{4.1}$$

$$= 10\mu A + \frac{20mA \times 100ms_{wakeup} + 70mA \times 1s + 200mA \times 0.1s}{300s} \approx 0.3mA$$

4.3 ESP32

A successor of the ESP8266 chip, also developed by Espressif System, is the ESP32 (Figure 4.7) module, which features a dual-core processor and supports Bluetooth communications. Here are some of its most relevant characteristics:

Figure 4.7: ESP-WROOM-32 Development Board [55]

- Dual-core processor Xtensa LX6 running up to 240 MHZ, making it particularly well suited for task handling;
- Built-in connectivity with Bluetooth and Wi-Fi, and support for multiple Wi-Fi modes, including Station, SoftAP, and Station+SoftAP;
- Support with WPA, WPA2 and WPA3, ensuring secure communications over Wi-Fi networks;
- Built-in connectivity with BLE, enabling energy-efficient Bluetooth communication;
- Robust TCP/IP stack, facilitating efficient communication protocols;
- 520KB of SRAM for data storage;
- 34 programmable GPIO pins and various peripherals such as SPI, I2C, UART and PWM;
- Incorporation of power optimization features and sleep modes;
- Compatibility with Arduino IDE, ESP-IDF, and native support of FreeRTOS.

ESP32's block diagram is presented in Figure 4.8.

4.3.1 Development Frameworks

Like its predecessor ESP8266, ESP32 is supported by Arduino IDE. Another common choice is ESP-IDF (Espressif IoT Development Framework), the official development framework for ESP32. It offers a comprehensive set of libraries, APIs, and tools for building IoT applications with advanced features and fine-grained control over the hardware.

ESP-IDF is built on top of FreeRTOS, unlike what happens with ESP8266. This ensures seamless integration and optimization, and added support since Espressif Systems can provide updates, bug fixes, and enhancements to FreeRTOS that are optimized for the ESP32's architecture.

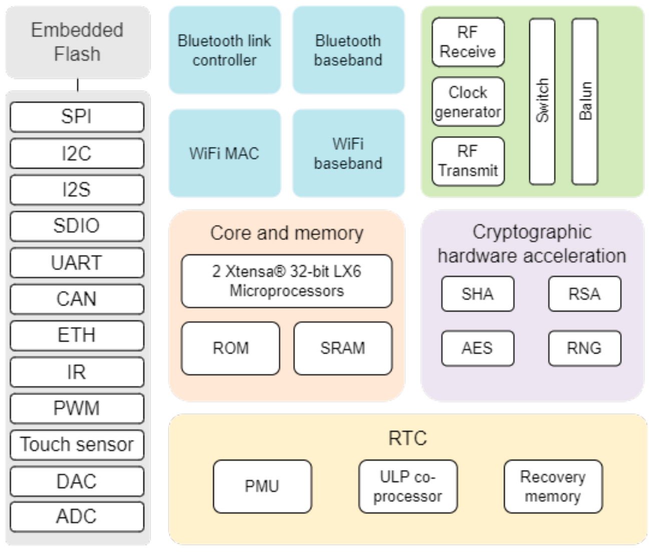


Figure 4.8: ESP32 Block Diagram [3]

Additionally, ESP-IDF provides a modular and flexible architecture that allows for the customization of various components, e.g. Wi-Fi, Bluetooth and peripherals. Furthermore, it offers a comprehensive collection of APIs for various functionalities, e.g. Transport Layer Security (TLS) for implementing secure communication protocols, and its TCP/IP stack supports IPv4, IPv6, TCP, UDP, HTTP, MQTT and more. Finally, it includes a command-line interface (CLI), a robust debugger, and an extensive set of documentation.

4.3.2 Power Management

The ESP32 can switch between four different sleep modes [3], summarized in the table below:

- Modem-sleep mode: the CPU is active, and its clock can be configured. However, the Wi-Fi/Bluetooth baseband and radio are disabled, conserving power while allowing the CPU to operate. When Wi-Fi is enabled, the chip switches between Active mode and sleep mode.

- Light-sleep mode: pauses the CPU, but certain components remain active. The RTC memory and peripherals continue running, along with the Ultra-Low Power (ULP) coprocessor. The microcontroller can be awakened by various events such as MAC activities, host events, RTC timer triggers, or external interrupts.

- Deep-sleep mode: only the RTC memory and RTC peripherals are powered on. The Wi-Fi and Bluetooth connections' data is stored in the RTC memory, and the ULP

Power mode	Description		Power Consumption
Modem-sleep	The CPU is powered on	240 MHz	30 mA ~ 68 mA
		160 MHz	27 mA ~ 44 mA
		Normal speed: 80 MHz	20 mA ~ 31 mA
Light-sleep	-		0.8 mA
Deep-sleep	The ULP coprocessor is powered on.		150μA
	ULP sensor-monitored pattern		100μA @1% duty
	RTC timer + RTC memory		10μA
Hibernation	RTC timer only		5μA
Power off	CHIP_PU is set to low level, the chip is powered off.		1μA

Table 4.4: ESP32 power consumption across different sleep modes (data extracted from the processor datasheet, [3])

coprocessor remains functional. This mode helps minimize power consumption while retaining essential functionalities.

- Hibernation mode: disables the internal 8MHz oscillator and ULP coprocessor. The RTC recovery memory is powered down as well. Only one RTC timer operating on the slow clock and certain RTC GPIO remain active. The RTC GPIOs can trigger the microcontroller to wake up from the Hibernation mode.

ESP32 offers more complex sleep mode options when compared with the ESP8266. While both boards can wake up from a sleep mode either after a predetermined period of time or an external interrupt, the ESP32 offers a more direct way of implementing the latter options, with built-in functions for external wake-up calls. Additionally, the ESP32 generally offers more advanced features and capabilities compared to the ESP8266, which can include enhanced Wi-Fi performance, Bluetooth connectivity and more GPIO pins.

4.4 Summary

The Arduino Nano 33 IoT, ESP32, and ESP8266 microcontrollers offer distinct features and capabilities, making them suitable for different project requirements.

The Arduino Nano 33 IoT provides a compact and energy-efficient solution with limited connectivity options but is well-suited for simple IoT applications. On the other hand, the ESP32 offers a more powerful dual-core processor, ample memory, and extensive connectivity options, including BLE support, making it a versatile choice for projects

Feature	Arduino Nano 33 IoT	ESP8266	ESP32
Clock speed	Up to 48Mhz	Up to 80 MHz	Up to 240 MHz
SRAM	32 KB	36 KB	Up to 520 KB
Flash Memory	256 KB	4 MB	4 MB
Connectivity	Wi-Fi, Bluetooth	Wi-Fi	Wi-Fi, Bluetooth
BLE	Yes	No	Yes
Security Protocols	WPA2	WPA2	WPA2, WPA3
I/O Pins	22	Up to 17	Up to 34
Sleep Modes	Idle [0-2], standby	Modem-sleep, light-sleep, deep-sleep	Modem-sleep, light-sleep, deep-sleep, hibernation
RTOS native support	No	No	Yes

Table 4.5: Comparison of Key Features and Specifications between different Microcontrollers

that require higher processing capabilities and advanced connectivity features. Meanwhile, the ESP8266, though less powerful than the ESP32, offers a cost-effective solution with Wi-Fi connectivity. Table 4.5 summarizes the main differences between the three microcontrollers.

Chapter 5

Experimental Characterization of Different Microcontroller Options

As previously stated, achieving low energy consumption is a crucial requirement in the field of smart meters, where long battery life and efficient power management are essential. To better understand the energy consumption characteristics of different hardware options, we conducted consumption and latency experiments using the Arduino Nano 33 IoT board, ESP32, and ESP8266 modules.

5.1 Methodology and Tools

Before the deployment of the sensors, a series of experiments were conducted. A simple push button was connected to the board to simulate the water flow sensor. Pressing the button would trigger the wake-up of the device from its sleep state. The latency, that is, the time delay between pressing the button and the device actually waking up, was measured for each board.

To gain a better understanding of the power consumption and current drawn during the cycle of putting the device to sleep and waking it up, Monsoon Solutions Inc.'s Power Monitor was employed. The Power Monitor is a specialized instrument designed to measure and analyze power consumption in various devices. It offers precise and high-resolution measurements of voltage, current, and power. The device supports voltage levels between 2.1V and 4.5V, with a maximum current of 3A. It operates at a sampling frequency of 5kHz.

The PowerTool software provided by Monsoon Solutions Inc. was used to retrieve and export the data related to the different variables. To accommodate small voltage fluctuations relative to the 3.3V input voltage of the Arduino Nano 33 IoT and the two ESP modules, the output voltage of the Power Monitor was set to 3.35V. The data was exported in CSV format to facilitate further data processing.

5.2 Arduino Nano 33 IoT

The red probe of the Power Monitor was connected to the 3.3V pin, and the black probe was connected to the ground, as can be seen in Figure 5.1. Even though the SAMD21 microcontroller provides the Idle and Standby sleep modes seen in Chapter 4, there are no direct ways of implementing them without resorting to changing the registers. It was chosen to use external libraries, which offer more flexibility, simplify the development process and enable the selection of specific wake-up sources based on the system requirements.

The device was put to sleep using two sleep functions from the *ArduinoLowPower.h* library: *LowPower.idle*() and *LowPower.sleep*().

- *LowPower.idle*() puts the microcontroller into idle mode, where the CPU is halted but other peripherals remain active. The microcontroller can wake up by triggering an interrupt or using a watchdog timer.

- *LowPower.sleep*() ensures the CPU is halted as well as most peripherals. Only some low-power peripherals, like the RTC, can wake up the microcontroller.

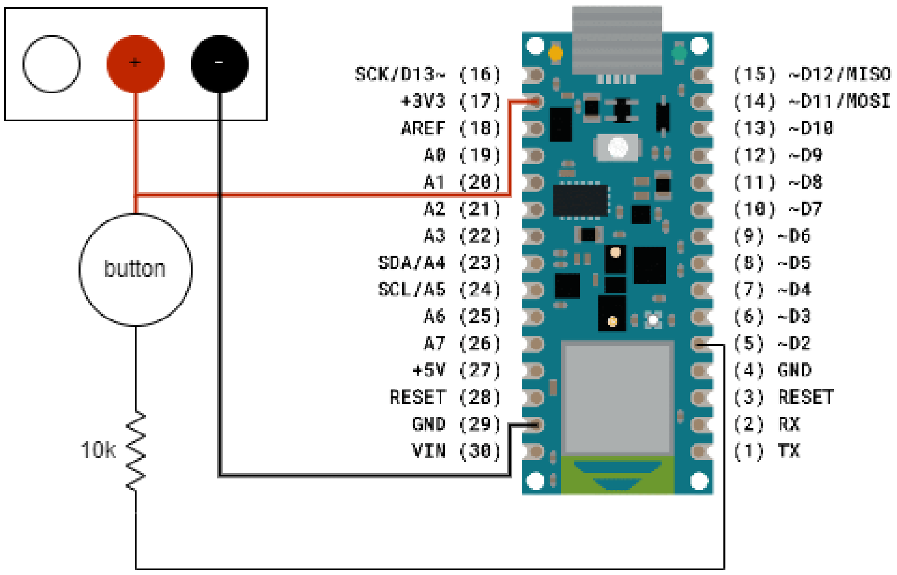


Figure 5.1: Schematic of the experimental setup for current measurement

The results of the conducted experiences are displayed in Figures 5.2 and 5.3. In Figure 5.2, no Wi-Fi connection was established, hence the periodic wakeup every ten seconds with relatively low values of current: around 13.65 mA during sleep mode and around 31.42 mA while awake. For these Figures, and all other Figures obtained using the power monitor, the orange line corresponds to current, the purple line to voltage, and the blue line to power.

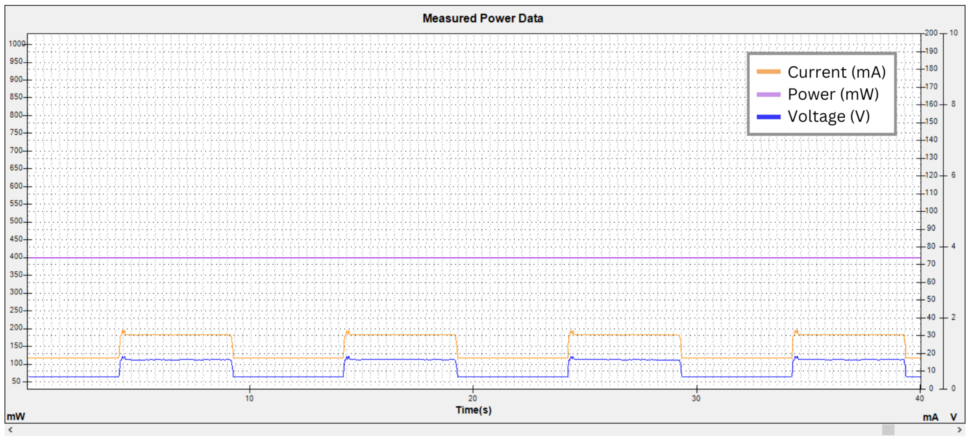


Figure 5.2: Arduino power consumption without Wi-Fi connection

Using the exact same function but now with a Wi-Fi connection, set up during the active periods and broken while sleeping, the power consumption increases significantly (Figure 5.3).

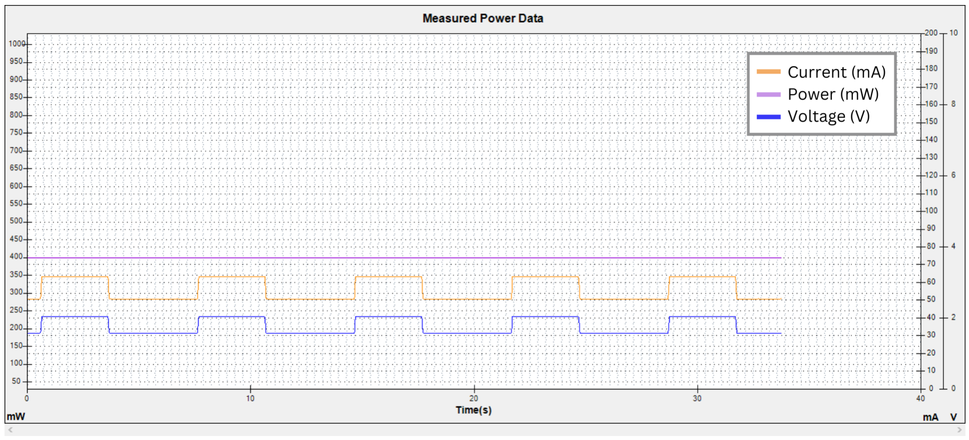


Figure 5.3: Arduino power consumption with Wi-Fi connection each time the board wakes up

In Table 5.1, the main differences between the two programs are summarized. It is observed that the presence of a Wi-Fi connection significantly increases the average current consumption, while the deviation stays nearly the same. It is interesting to notice that the wake up latency is lower in the program where a Wi-Fi connection is present. This suggests that the initialization and establishment of the Wi-Fi connection may contribute to a faster stabilization of the board current values upon waking up. The exact reasons for this observation may be attributed to factors such as the efficiency of the Wi-Fi module or the optimization of the software implementation.

The measurement of latency is a challenging task that involves information that is hard to acquire, such as the precise point when the Arduino initiates task execution. Despite

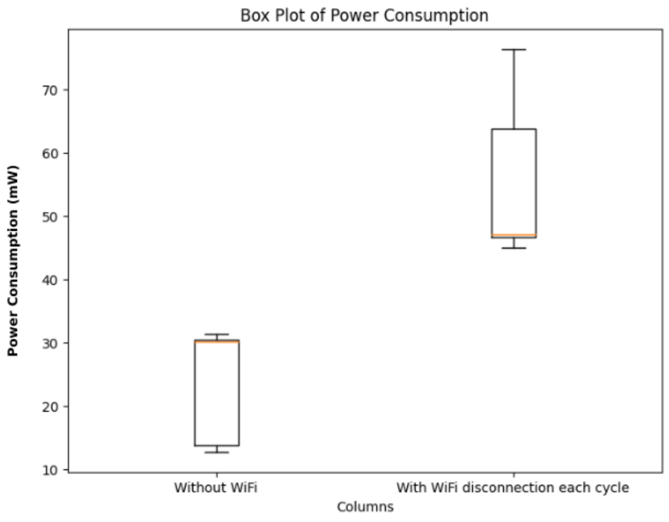


Figure 5.4: Arduino power consumption: without Wi-Fi vs. with Wi-Fi connection

this complexity, we have made a rough estimation by considering two key metrics: "wake up latency" and "sleep latency."

The "wake up latency" refers to the duration required for the Arduino board to achieve a stable state and exhibit consistent values after the power consumption starts to rise. This metric helps us understand the time it takes for the board to become fully operational from a low-power state (Figure 5.5).

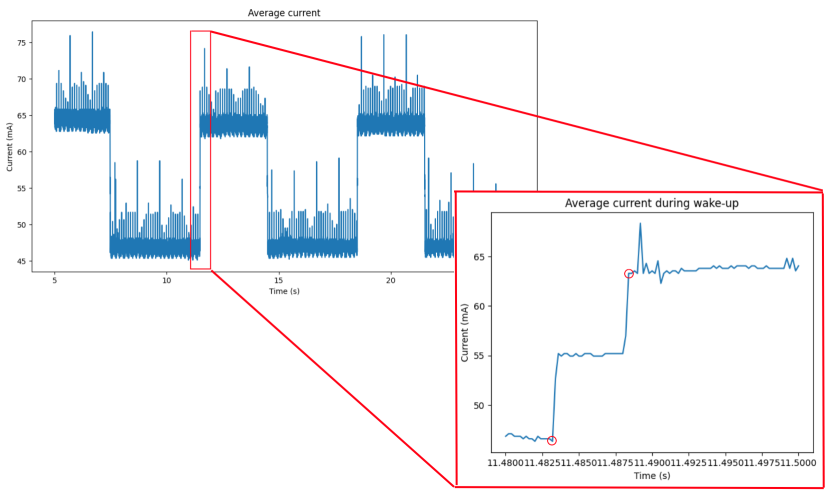


Figure 5.5: Measurement of wake-up latency

Similarly, the "sleep latency" indicates the duration needed for the Arduino board to settle into a stable state with consistent values after entering the sleep mode. This metric provides insights into the time it takes for the board to enter a low-power mode and stabilize its operations (Figure 5.6).

Statistical Measures	Without Wi-Fi	With Wi-Fi
Average	25.17 mA	59.72 mA
Minimum	12.80 mA	34.45 mA
Maximum	31.45 mA	262.6 mA
Standard deviation	7.79 mA	23.62 mA
Wake up latency	19.8 ms	5.2 ms
Sleep latency	15.8 ms	15.4 ms

Table 5.1: Statistical Measures between programs with and without Wi-Fi connection

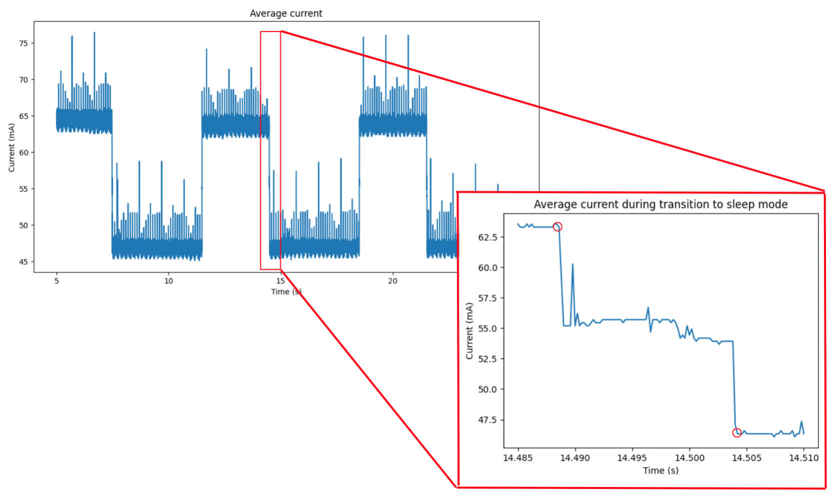


Figure 5.6: Measurement of sleep latency

A striking aspect in the values of Table 5.1 is the clear discrepancy between the values seen here and the values mentioned in Chapter 4.

Firstly, the absence of official sleep mode libraries for the Arduino Nano 33 IoT can make it challenging to compare power consumption values obtained from external GitHub libraries, such as Adafruit's Sleepy Dog or Arduino Low Power, with the official values derived from directly manipulating registers. These third-party libraries may introduce variations in power consumption measurements due to differences in implementation approaches and library overhead.

Secondly, while the SAMD21 microcontroller is known for its low power consumption capabilities, there are several other components that contribute to the overall power consumption. Figure 5.7 illustrates the maximum current drawn by each one.

Besides the SAMD21 microcontroller, we have the NINA-W102 module, responsible for providing wireless connectivity - in this case, Wi-Fi - to the board, consuming significantly more power when transmitting or receiving data, which is an important step of the experiences above. Importantly, even when the SAMD21 microcontroller is put into

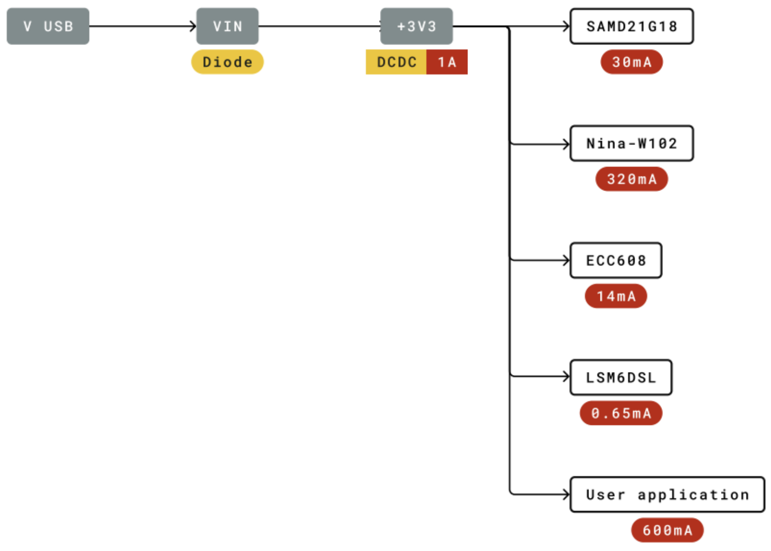


Figure 5.7: Power Tree of the Arduino Nano 33 IoT [56]

sleep mode, the NINA-W102 module may continue to draw some current, as it needs to maintain the association to the AP and receive broadcasts/multicasts that may arrive without being requested. The ECC608 secure element chip is utilized for cryptographic operations and secure communication. Although its power consumption is relatively low, it still contributes to the overall power consumption of the board. During sleep modes, the power consumed by the ECC608 is typically negligible, as its functionalities are not actively used. The LSM6DSL accelerometer sensor, responsible for detecting motion and orientation, also adds to the power consumption. While its power requirements are relatively modest, they do contribute to the overall current draw. Similarly to the ECC608, the LSM6DSL sensor power consumption is typically minimal when the board is in sleep mode. The "user application", the code written and run on Arduino, might also greatly influence the overall power consumption.

5.3 ESP8266

In ESP8266, unlike with the Arduino, there are official libraries that ensure the good functioning of the existent sleep modes. When using *deepSleep*, the ESP8266 can only be awoken by either a timed interrupt or a specific external interrupt - GPIO 16. This pin is considered a special pin since it serves this specific purpose of being used to wake up the board.

Under normal circumstances, the RST pin maintains a HIGH signal while the ESP8266 is running. However, when a LOW signal is applied to GPIO 16, it triggers a reset action, consequently restarting the board and bringing it out of the deep sleep state. Therefore, in order to carry out these experiments, the ESP8266 was put to sleep and a wire was connected between GPIO 16 and the RST pin. Additionally, a timer of 10 seconds was used; after this time elapsed, the board woke up.

Similarly to the case with the Arduino, the board was connected to Wi-Fi each time it woke up, and disconnected from Wi-Fi immediately before going back to sleep.

Using the power monitor mentioned in Section 5.2 with a similar setup of connecting the red probe to the 3.3V pin and the black probe to the ground, new power consumption measurements were obtained.

Figure 5.8 shows that this board is successful in achieving particularly low power consumption values. In fact, while in its deep sleep state, the board reaches an average of 18.0 μA.

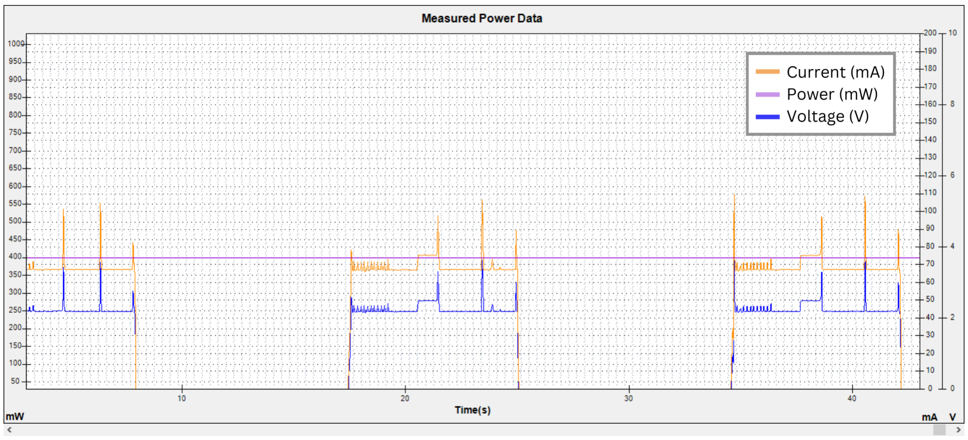


Figure 5.8: ESP8266 power consumption

This value of 18.0μA is similar to the 20μA value for current consumption in ESP8266's sleep mode, in Table 4.3. While it may seem surprising that the ESP8266 board presents a current consumption so similar to the predicted consumption of the microcontroller alone, unlike what happened with the Arduino, several factors contribute to this phenomenon. The ESP8266 is specifically designed to optimize power consumption, and several functionalities, including Wi-Fi connectivity and peripherals, are designed to keep power management techniques in mind. On the other hand, the Arduino Nano 33 IoT has to power other components which consume additional power and which the ESP8266 does not use. Additionally, ESP8266 benefits from having official libraries and resources that guarantee that official sleep modes are actively being implemented.

As seen in Chapter 4, the DTIM and the beacon interval play a crucial role in the behavior and power consumption of the board [57]. In these experiments, the DTIM was set to 3 and the Beacon Interval do 100ms.

5.4 ESP32

Although, as seen in Chapter 4 the board has advanced features like different interrupt modes, we chose to use timers for simplicity and better task tracking. To ensure a controlled time frame for each task, we implemented the following approach:

1. Before attempting to connect to Wi-Fi, the program waited for two seconds to ensure stability.

2. After successfully establishing an internet connection, the program waited for three seconds before going back to sleep. This delay allowed sufficient time for any necessary data transfer or synchronization. The internet connection was closed implicitly before going back to sleep.

3. Each time the program woke up, it reconnected to Wi-Fi, ensuring a fresh connection.

By using timers and following this procedure, we aimed to streamline the execution of tasks and maintain a predictable time frame for each operation.

The power monitor was connected with the same setup as before. Figure 5.9 illustrates the results of a *deepSleep* cycle.

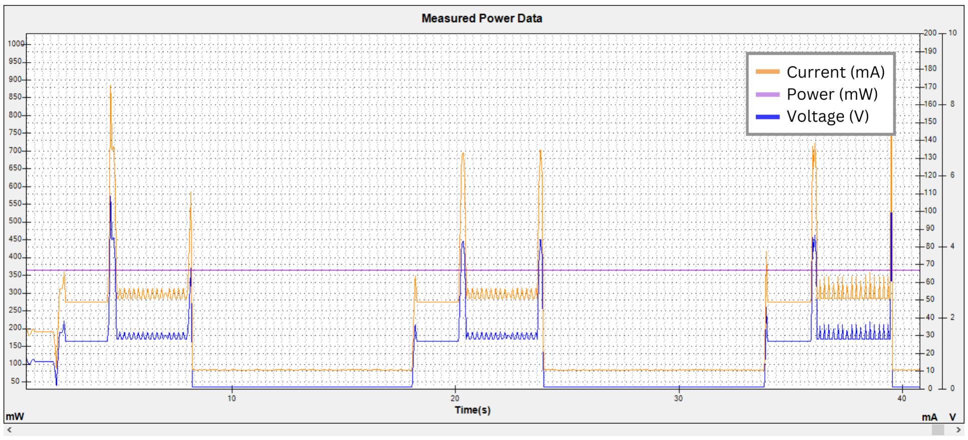


Figure 5.9: ESP32 power consumption

A notable observation is the presence of a current peak of approximately 135 mA each time the Wi-Fi connection is established. This peak indicates an increase in power demand during the process of establishing the Wi-Fi connection. Additionally, a second peak, slightly higher than the first, is detected just before the board going to sleep. The reasons for these peaks could be attributed to Wi-Fi module initialization and data transmission, which require higher power.

Interestingly, it was discovered that current consumption significantly drops when the Wi-Fi connection is actively disconnected before putting the board to sleep. This reduction in power consumption suggests that the board may not fully disconnect the Wi-Fi connection when entering sleep mode, unlike what was predicted. During the sleep state, current consumption remains relatively stable with an average value of 10.74 mA.

5.5 Summary

The aforementioned programs with a Wi-Fi connection were run and the data was filtered so that each CSV file contained values related to the same timespan and cycle. Figure 5.10 showcases the results.

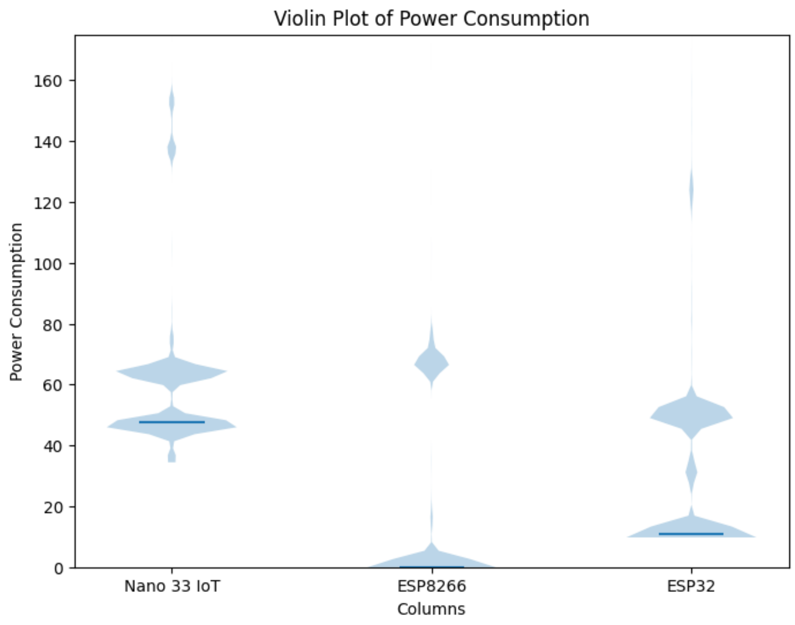


Figure 5.10: Power consumption by the three boards, with Wi-Fi connection

Statistical Measures	Arduino Nano 33 IoT	ESP8266	ESP32
Average Sleep	46.18 mA	0.42 mA	10.75 mA
Average Awake	73.94 mA	63.43 mA	52.34 mA
Standard deviation	23.62 mA	29.51 mA	28.03 mA
Wake up latency	5.2 ms	3.6 ms	2.0 ms
Sleep latency	15.4 ms	4.4 ms	1.2 ms

Although the Arduino Nano 33 IoT is comparatively less efficient in achieving extremely low power consumption values during sleep mode, it was selected and used in Chapter 6 for its convenience and ease of use. SWRS's preference for the Arduino board also influenced the decision to utilize it in the project. It should be noted, however, that the system described is designed with scalability in mind, allowing for the possibility of substituting the Arduino board with a more suitable alternative should the need arise.

Furthermore, it should be noted that the high power consumption observed during the awake state of the three boards could potentially be reduced if the USB peripheral was not in use. When the boards are connected to a power source via USB, they rely on the provided power to operate. However, since the goal is to power the boards using a battery, unnecessary power drain associated with the USB interface can be eliminated. This can lead to more efficient power utilization and an overall improvement in power consumption, making it more suitable for battery-powered applications.

Chapter 6

System Implementation

The entire module will focus on three main components: the microcontroller module inside the drain, the flush control module, and the gateway.

6.1 Microcontroller Module

The microcontroller module includes the microcontroller itself, a water flow sensor, a TDS sensor, a battery, a battery charger module, and a water turbine, as well as the associated software.

6.1.1 Power Supply

Battery and Charging Module

As previously mentioned, it is important that the system be portable and lightweight since it will have to fit inside a drain. With that being the case, the microcontroller will have to be battery-powered. The use of a battery ensures that the module is activated at the first signal of water flow, without having to wait for the turbine (described below) to generate enough energy.

Two types of batteries can be used: primary batteries and secondary batteries. Primary batteries are non-reachable and deliver 1.5V to power a device, and are particularly suited for low-energy portable devices in non-remote locations [58]. Secondary batteries are rechargeable and more expensive and come with additional safety issues. Although the affordability of this module is of extreme importance, the rechargeability of the battery (considering the number of times data will be transmitted) outweighs it, so a secondary battery was selected. Rechargeable battery options include nickel-cadmium (NiCd), nickel-metal hydride (NiMH), and lithium-ion (Li-Po).

- The NiCd battery offers a long shelf life and can withstand abuse. It is affordable and easy to charge and offers a high load current. However, the memory effect

[59], defined as the ability of the battery to remember its regular usage pattern and thus gradually reduce its effective capacity, is a problem worth considering with this material. Not only that, the cadmium chemistry makes it an environmentally unfriendly option, since this metal is a health hazard to all human beings.

- The NiMH battery addresses the problems of the previous battery, but comes with additional ones: there are no memory issues and no environmental or safety complications, but it is considerably more expensive, and one must consider the crystal formation that impacts performance, as well as the lack of ability to perform well under warm conditions. Its reduced shelf life is also a factor to keep in mind.

- The Li-Po battery has a much higher energy density than its alternatives and, like the nickel metal hydride, it usually does not struggle with memory effect. It can, however, be a safety hazard, since the flammable electrolytes of each cell are highly prone to explosions and fires if they are incorrectly charged. Research is currently in progress to eliminate these types of electrolytes. Another factor to keep in mind is the controversial extraction of lithium since it damages soil and requires massive amounts of water.

Taking into account all of these factors, a 3.7V 980mAh Li-Po battery from Mauser [60] was selected. This choice was due to several factors. First of all, it offers voltage compatibility with the Arduino Nano 33 IoT board. Second, as explained, the rechargeability of the battery was a non-negotiable factor, and reduces the need for frequent battery replacements. Finally, the Li-Po battery can store a higher amount of energy than its counterparts and ensures consistency and reliability without having to worry about memory effect.

Mauser's TP4056 battery charger module [61] was selected to charge the aforementioned battery. This module is specifically designed for Li-Po batteries and supports a charging voltage of 4.2V. It features built-in charging circuitry, including a charging integrated circuit (IC), protection circuits and a LED that indicates charging status. Additionally, it incorporates over-charging and over-discharging protection mechanisms. It may connect to the power source via USB or wires, and to the Li-Po battery via the positive and negative terminals, as shown in Figure 6.1.

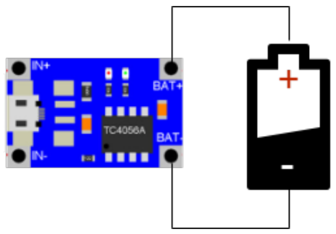


Figure 6.1: Battery Charging Circuit

Considering that the fully-charged voltage of the Li-Po battery is 4.2V, it is necessary to drop it down so that it can feed the Nano 33 IoT, whose input voltage is 3.3V. A low-dropout voltage regulator needs to be used between the Li-Po and the board. To this effect, a MCP1700 [62] was chosen. Figure 6.2 illustrates the circuit between these three components. According to MCP1700's datasheet, 1 µF output capacitors help stabilize the LDO output, so two ceramic capacitors were added: one between the positive terminal of the battery and the ground, and another one between V_{out} and the ground.

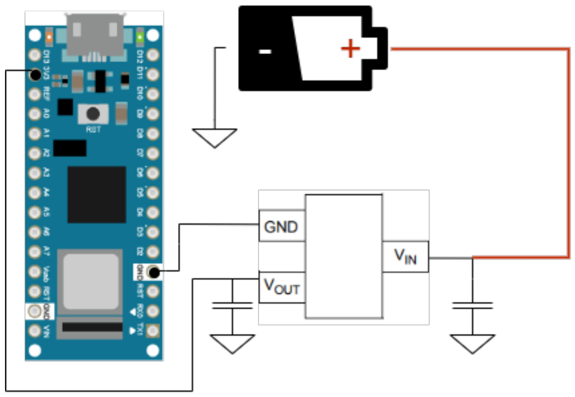


Figure 6.2: Voltage Regulator Circuit

Water Turbine

To maximize system energy efficiency, we utilized a turbine. The TP4056 battery charger module requires a stable DC voltage between 4.5V and 5.5V. Instead of powering the battery charger through USB or a wall adapter, this input voltage can be provided by a water turbine.

For this purpose, the DF Robot's Water Turbine 5V DC [63] was employed. This turbine incorporates a built-in voltage regulator, ensuring a constant output of 5V DC, i.e., within the range that the TP4056 battery charger module can endure, provided that the minimum starting condition of 2.5V input voltage (about 3.5L/minute) is met.

The initial plan involved utilizing the water turbine both as a power source and as a water flow sensor simultaneously. If the turbine did not include a built-in voltage regulator, it would be necessary to analyze the generated impulses and correlate them with a frequency proportional to the water flow. After applying a low-pass filter, a smooth voltage output could be utilized to charge the battery. However, since these functionalities are internally integrated within the turbine, it is not possible to directly measure the water flow. Consequently, it is only feasible to draw conclusions regarding the opening and closing of the tap, which is less informative than anticipated. To address this limitation, it becomes imperative to acquire a water flow sensor and integrate it with the turbine. This allows one device to generate power through water movement, while the other measures it more accurately than the turbine is able to.

6.2 Sensors

Our first goal is to take water statistics and process them. The water amount and quality are particularly important. Its quality is what will determine the type of filtering the water should go through, and its quantity is important in three steps of the process: right at the beginning to determine if water is flowing from a tap and whether our system should turn on or not, after the filtering to calculate the percentage of water which was reused (i.e. redirected to the flushing system), and at the very end to know the amount of completely clean water that the flushing system might require (ideally that would be none, but there is a low chance that the reused water matches the needed amount for flushing on a regular basis). For this reason, two types of sensors are presented, namely TDS and Water Flow sensors, which we introduce next.

6.2.1 TDS Sensor

A Total Dissolved Solids (TDS) sensor, depicted in Figure 6.3, is employed to quantify the cleanliness of water in terms of particles per million (ppm). The ppm value indicates the density of particles dissolved in the water. For instance, a fluid with a TDS value of 1 ppm signifies the presence of one soluble solid particle for every one million water particles. Higher ppm values indicate less clean water, while acceptable levels for drinking water typically range from 50 to 150 ppm. The World Health Organization (WHO) has established 300 ppm as the recommended upper limit. If the measured value exceeds 1000 ppm, it is considered unsafe for human consumption and necessitates filtration. Beyond 2000 ppm, it becomes unlikely that a filtration system can effectively purify the water. Conversely, low ppm values outside the normal range indicate high-quality water with minimal mineral content. In cases where such water is not intended for human consumption, the acceptable ppm limits may be extended.

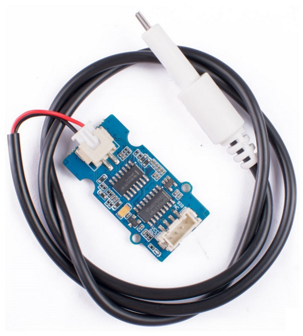


Figure 6.3: TDS Sensor [64]

The TDS sensor is affordable and user-friendly, comprising a waterproof probe designed for submersion in the fluid. It can be connected to a standard microcontroller using an

ADC. In this study, the DF Robot's Analog SEN0244 TDS Sensor [64] was utilized, with the following specifications:

- Input Voltage: DC 3.3 - 5.5 V
- Output Voltage: 0 - 2.3 V
- Working Current: 3 - 6 mA
- TDS Measurement Range: 0 - 1000 ppm
- Number of Needles: 2
- Total Length: 83 cm

To calculate the TDS content in water, a small electrical current is applied to the water solution. The dissolved solids within the water conduct electricity, leading to detectable changes in electrical conductivity by the sensor. The TDS sensor then converts these changes into a digital reading, providing an estimate of the total number of dissolved solids in the water.

To measure the TDS value, the sensor reads the analog value every 40 ms and stores it in a buffer. Every 800 ms, the values of the analog buffer are copied to a temporary buffer and a function calculates the median value using a simple bubble sort to arrange the values in ascending order, which helps stabilize the readings. This value is then converted to an average value by multiplying it by the reference voltage and dividing it by the ADC resolution (4096.0). Then, the temperature compensation is applied. For a default temperature of 25ºC,

$$\begin{aligned}\text{tdsValue} = (&133.42 \times \text{averageVoltage}^3 \\ &- 255.86 \times \text{averageVoltage}^2 \\ &+ 857.39 \times \text{averageVoltage}) \times 0.5; \qquad (6.1)\end{aligned}$$

Upon connecting the three pins of the sensor to the Arduino and developing the necessary code, we performed calibration. Distilled water, known to have a TDS value of 0, was utilized as a reference for calibration purposes.

Subsequently, experiments were conducted using various types of water to validate the sensor's functionality. These experiments recorded the ppm values for the following solutions:

- Bottled water with no added sugar;
- Bottled water with a sugar concentration of 4.17%;
- Tap water;

- Bottled water with a sugar concentration of 4.17% exposed to air for 14 days.

Around 40 measurements were taken for each solution, and the results are presented in Figure 6.4. It is evident that the water samples exhibit varying levels of TDS values, which provide insights into their respective qualities. Analyzing key statistics reveals important trends and comparisons among the different samples, specifically that the bottled water solution exhibits the lowest TDS values when compared to solutions containing external agents such as sugar or air exposure.

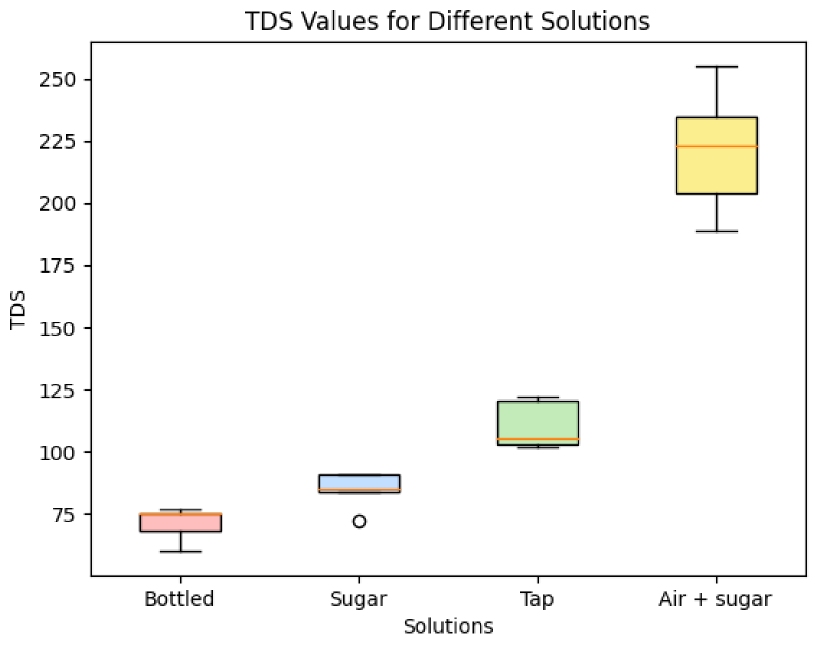


Figure 6.4: TDS Values of Different Solutions

Bottled water stands out as the purest sample. In contrast, tap water, with an average TDS value of 109.5 ppm and a variation spanning from 102 ppm to 122 ppm, demonstrates a higher concentration of dissolved substances. The water solution with 1 teaspoon of sugar exhibits an average TDS value of 85.1 ppm, indicating the contribution of dissolved substances when compared to the bottled water with no added sugar.

It is also worth noticing the wider range of values in the solution that was exposed to air. This variation can be attributed to the potential contamination and interactions with airborne particles and microorganisms present in the environment, which significantly impact TDS measurements. Specifically, the inclusion of a sugar concentration of 4.17% in water that has been left exposed to the open air for 14 days demonstrates an average TDS value of 225.9ppm, which is significantly higher than the rest, emphasizing the impact of exposure to the air and the potential accumulation of additional dissolved substances over time.

It is essential to point out that when it comes to flushing water, there are no universally defined or regulated TDS limits specifically for flushing purposes. The optimal TDS levels for flushing water can vary depending on several factors, including local water quality standards and the type of equipment or plumbing systems involved. In general, lower TDS levels are desirable for flushing water to minimize mineral deposits and scaling which can affect the efficiency and lifespan of plumbing fixtures and appliances. Several other factors, such as pH and dissolved oxygen, should be considered in order to get a full picture of the water quality.

6.2.2 Water Flow Sensor

The water flow sensor (Figure 6.5) calculates the amount of water moving through a tube. It consists of a copper body, a water rotor, and a hall effect sensor.

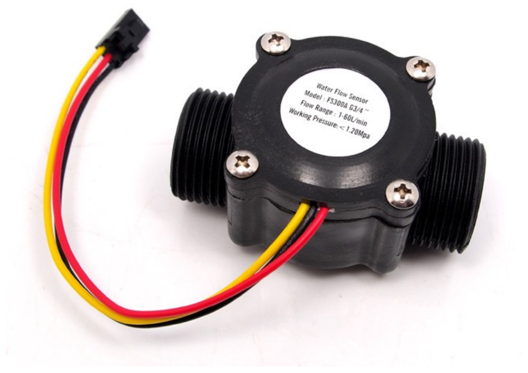


Figure 6.5: Water Flow Sensor [65]

The Hall effect sensor detects the presence and magnitude of a magnetic field using the Hall effect, which acts on the principle that when charged particles pass through a magnetic field, they get deflected from their original path [66], making one of the sides of the disk negatively charged and the other one positively charged, generating a voltage difference, illustrated in Figure 6.6.

The rotation of the paddle wheel of the sensor when water passes through it causes the magnet to move, which is then detected by the Hall effect sensor, which outputs a digital signal each time a change in magnetic field is detected.

The parameters of DF Robot's SEN0217 Water Flow Sensor [65], which was utilized in this project, are the following:

- RoHS[1] Compliant
- Working Voltage: 5 - 12 V
- Flow Rate Range: 1 - 30 L/min

[1]Restriction of the Use of certain Hazardous Substances in Electrical and Electronic Engineering, a product level compliance based on a European Union's Directive.

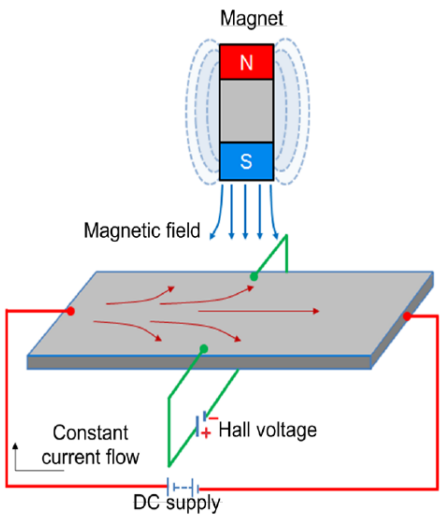


Figure 6.6: Functioning of the Hall Effect Sensor [67]

- Operating current: 15 mA (DC 5V)
- Water Pressure: <= 1.75 MPa
- Liquid Temperature: < 120º C
- High-level Output Pulses: >4.7VDC (with 5V DC input voltage)
- Low-level Output Pulses: <0.5V DC (with 5V DC input voltage)

This sensor provides square-wave output pulses with a duty cycle of 50%±10% (Figure 6.7). As water passes through the sensor, the sensor, it causes an internal turbine to rotate. Each tooth of the turbine is magnetized and when they pass in front of the Hall effect sensor it generates square-wave pulses proportional to the flow rate, and in this case, a pulse corresponds to 1/450th of a liter. As such, whenever a rising edge trigger is detected on the digital pin to which the sensor is connected, an interrupt service routine is called and increments the *waterFlow* variable by 1/450 to obtain the total amount of water.

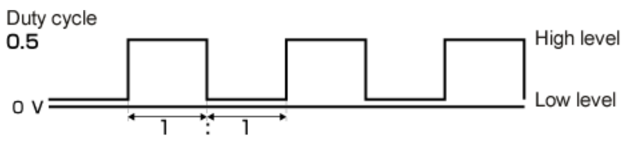


Figure 6.7: Square Wave Pulses Generated by the Water Flow Sensor

No calibration was required for the flow sensor. The accuracy of the sensor was confirmed by conducting a test using a completely filled 120 ml cup. When all of the water from the cup was passed through the flow sensor, the total flow output measured by the sensor matched the volume of the cup, indicating that the flow sensor was operating correctly and providing accurate measurements.

The results shown in Figure 6.8 were obtained by transferring 120 ml of water from one cup to another. Initially, the transfer was done at a relatively high speed, trying to simulate the fastest possible pouring speed achievable manually. After the first 0.05 liters were reached, the transfer speed was gradually reduced to reach the minimum threshold while still attempting to maintain flow.

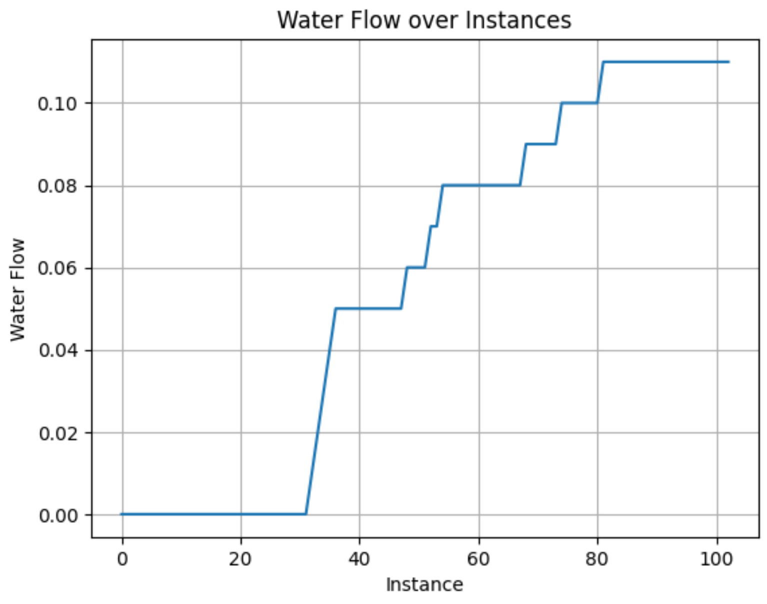


Figure 6.8: Evolution of water amount

However, it is worth noting that the accuracy of the measurements depends on a certain speed of the water flow, since the sensor does not interpret ranges of less than 1 L/minute. In this case, the total volume recorded by the sensor was 110 ml instead of the expected 120 ml. This discrepancy can be attributed to the fact that the minimum threshold required for the sensor to register a flow was not consistently reached at all times.

6.2.3 Integration of the Microcontroller Module Components

To summarize, the water turbine generates a 5V DC output, which is utilized by the battery charger to power up the battery. The battery, now charged and at 5V, needs to go through a voltage regulator to be adjusted to 3.3V before it can safely power the Arduino Nano 33 IoT. Simultaneously, the Arduino powers both the TDS sensor and the flow sensor.

Figure 6.9 illustrates this setup below. The voltage regulator is not explicitly shown in the diagram for simplicity, but it is connected as depicted in Figure 6.2.

After deploying the turbine, it was verified that the experimental conditions did not allow a water circuit, ensuring a permanent water flow at a strong enough speed and pressure to meet the minimum threshold of 3.5 L/minute. The highest possible speed

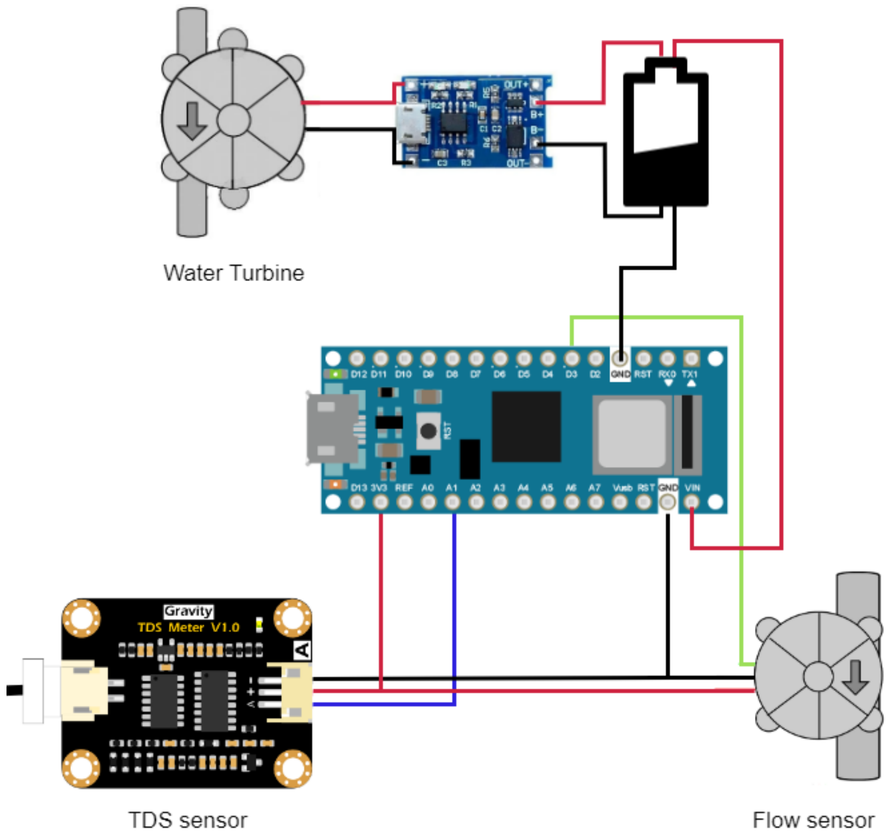


Figure 6.9: Connections between microcontroller module components

only yielded approximately 3.7V, which is less than what the battery charger module requires to operate. Consequently, we decided to temporarily use a 5V power supply, which can simulate the 5V output the turbine was originally intended to provide.

Whether using a turbine or a 5V power supply, we can calculate an estimate of the battery life. On average, an individual takes one daily shower of 8 minutes [68], and on average, a household holds 2.2 people [69], or 17.6 minutes of showers. Globally, it is advised to wash our hands about 5 times per day, with each time taking about 20 seconds. This makes out a total of 1.7 minutes. This totals about 19.3 minutes of water use. This means that out of the 1440 minutes of a day, about 19 would carry out with an active Arduino, and the remaining 1421 would be in sleep mode, with lower consumption. The average current consumption would therefore be about:

$$\frac{46.18mA \times 1421\text{minutes} + 73.94mA \times 19\text{minutes}}{1440\text{minutes}} = 46.55mA \tag{6.2}$$

With our battery of 1000 mAh, that gives out:

$$\frac{1000mAh}{46.55mA} = 21.48\text{hours} \tag{6.3}$$

It is essential to note that the current value may be higher (and the battery life lower) in practice, as the sensors will also be drawing power during operation, which could impact the overall power consumption.

However, the consecutive time the battery can operate alone is not necessarily the most important consideration since it will have opportunities to recharge between uses. What is truly relevant is determining if the battery can sustain the Arduino in sleep mode during the interval between two uses. We can estimate that approximately half of the power consumption occurs in the morning (around 10 minutes), and the remaining half occurs at night (around 9 minutes). Assuming a 12-hour separation between these two periods of utilization, we can assess if the battery can withstand 12 hours in continuous sleep mode.

$$\frac{1000mAh}{12h} = 83mA \tag{6.4}$$

The battery would be therefore able to take a maximum consumption of 83 mA without completely losing its charge. This value is a lot higher than the expected value during sleep (46.18 mA).

Another interesting factor to consider is whether or not the battery can, during its 10 active minutes, recharge the amount it discharged during the previous 12 hours. In order to make sure this balance is kept, the discharge is calculated:

$$\text{discharge} = 46.18mA \times 12\text{hours} = 554mAh \tag{6.5}$$

The voltage regulator, TP4056, is designed to handle the entire charging process, including monitoring the battery voltage and adjusting the charging current accordingly. It does not require the battery to be completely empty before starting to charge it. Assuming the maximum of 150 mA the turbine can output, the TP4056 module would charge:

$$\text{charge} = 150mA \times \frac{10}{60}\text{hours} = 25mAh \tag{6.6}$$

Therefore, since the charge is inferior to the discharge, the battery is not capable of charging what it lost. In order for the balance to be kept, the charge would have to be at least as great as the discharge. Or:

$$\text{discharge} = 554mAh = \text{current} \times \frac{10}{60}\text{hours} \tag{6.7}$$

$$\text{current} = 3320mA$$

The calculated value of 3320mA for the required charging current seems unreasonably high given the limitations of the turbine-powered TP4056 module: the turbine maximum output of 150mA suggests that sustaining a significantly higher current output is not be feasible. It is unlikely that the battery can be fully recharged during the 10 minutes of active operation. To address this limitation, alternative charging methods or a larger capacity battery may need to be considered to ensure sustained operation without relying solely on external charging.

It is important to note that the calculations provided above are based on average values and strict assumptions, which may not accurately represent the actual usage patterns or power consumption of the Arduino.

The substantial disparity between the Arduino's high power consumption during sleep mode and the limited charging capability of the battery suggests that an alternative board might be a more suitable choice. We can now revisit the calculations for the ESP32 board.

$$\text{discharge} = 10.75mA \times 12\text{hours} = 129mAh \tag{6.8}$$

The charge remains the same as in Equation 6.6. Despite the lower discharge, the current draw still exceeds the charging capacity.

$$\text{discharge} = 129mAh = \text{current} \times \frac{10}{60}\text{hours} \tag{6.9}$$

$$current = 774.0mA$$

In order to achieve a balanced charge-discharge scenario, the current would need to be approximately 774.0 mA. While a current value of 774.0mA is more acceptable than 3320mA, it still exceeds the output capability of the turbine.

Finally, we do the same for the ESP8266 board.

$$\text{discharge} = 0.42mA \times 12\text{hours} = 5.04mAh \tag{6.10}$$

The discharge during the 12-hour sleep period is calculated to be 5.04mAh, given the ESP8266's ultra-low power consumption during sleep mode. The the current drawn during this inactive period is therefore lower than the maximum charge current. Consequently, it would be feasible to have a self-sustaining battery system capable of charging the amount it discharges.

6.3 Flush Control Module

After the microcontroller module, the water goes through the filtration system that SWRS developed, which includes filtration and disinfection steps, as well as all the required mechanical pumps and valves.

Comparably to the microcontroller module, this system activates when the flush valve is opened. It communicates via Wi-Fi to the Gateway module. After leaving the filtering system, the water either goes into the wastewater stream or carries its way onto the flushing system, in which case a second water flow sensor allows the user to know the amount of

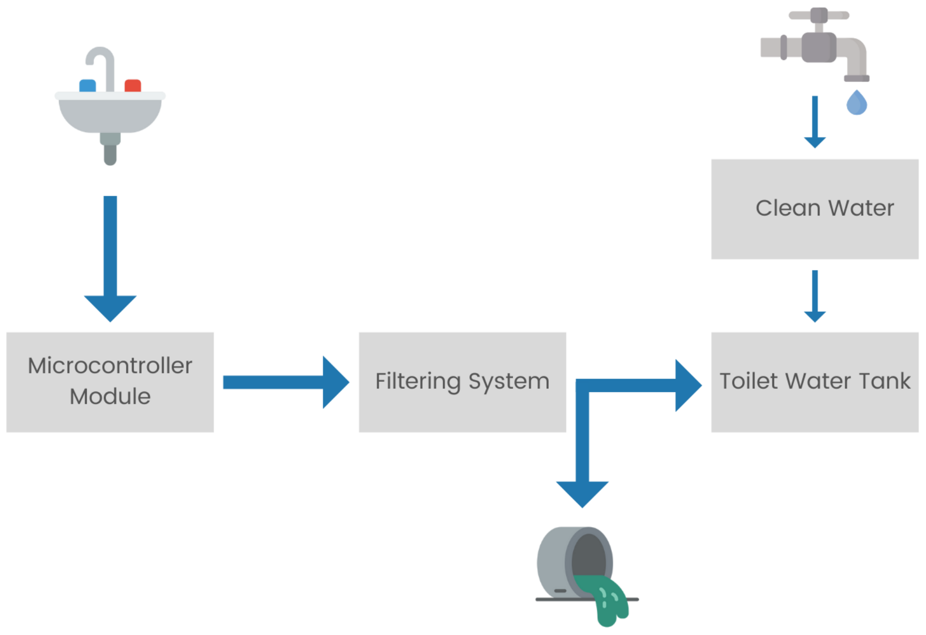


Figure 6.10: Water course and module positioning diagram

reused water. This greywater finally enters the tank of flushable water, where it might be mixed with completely clean water if it is not enough on its own. A third water flow sensor measures the amount of clean water, which is set to be as low as possible. The diagram in Figure 6.10 better illustrates the route the water goes through before reaching the final goal.

6.4 Raspberry Pi as an IoT Gateway

A Single-Board computer (SBC) is, as the name says, a computer built on a single board, featuring a microprocessor, memory, I/Os, etc. One example of an SBC is the Raspberry Pi (Figure 6.11), whose main advantages are its low price, its modularity, and its open design. Since it adopted the HDMI and USB standards, it is, like Arduino, a good entry point to learning Electronics as a hobby. Because the Raspberry Pi uses a full-fledged 64-bit microprocessor, its possibilities are far vaster. It runs a complete Operating System, usually Linux, and it can process the data on the board instead of just redirecting it to an external processing device, e.g. a "traditional" computer or the cloud. However, with larger computing power comes higher power consumption, which explains the fact that the Raspberry Pi is more commonly used as a gateway.

In this specific case, the gateway is a Raspberry Pi 4, which connects to the Internet via Wi-Fi. After activating an SSH connection with a personal laptop, it was possible

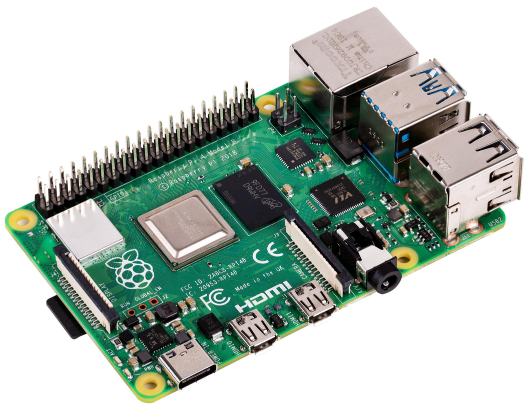


Figure 6.11: Raspberry Pi SBC [70]

to access the Raspberry Pi through the laptop terminal. The Raspberry Pi was then connected to Wi-Fi. After this initial setup, two sets of code were written on each device.

On the Arduino side, the program starts by establishing a Wi-Fi connection, using the predefined SSID and password, to the same network that the Raspberry Pi was connected to. The connection to the same Wi-Fi network is essential to ensure wireless communication between the two devices. A loop checks if the client is connected to the server, and, if not, attempts to connect. Once this connection is successfully done, the acquired values of TDS and water flow data are sent to the server. Ten values of each sensor are sent at a time. After sending the data, the client closes the connection. The acquisition and transmission of sensor data to the server can be seen in the code snippet below:

```
if (!client.connected()) {
  if (client.connect(serverIP, serverPort)) {
    Serial.println("Connected to server!");
    for (int i = 1; i <= 10; i++) {
      measureTdsValue();
      client.println(tdsValue);
      measureFlowValue();
      client.println(waterFlow);
    }
    // Close the connection after sending values
    client.stop();
  }
```

On the Raspberry Pi side, the IP address and port number on which the Raspberry Pi listens for incoming connections are set up, as well as all database-related data to store received values.

It should be noted that with each new connection of the Raspberry Pi to the mobile hotspot as a Wi-Fi source, the IP address of the Raspberry Pi changes. This behavior is attributed to the Dynamic Host Configuration Protocol (DHCP), which automatically assigns available IP addresses to devices on a network. By employing dynamic IP assignment, DHCP optimizes IP address allocation and management within the network. To accommodate the changing IP addresses, a simple *ifconfig* command executed on the Raspberry Pi's terminal retrieves the new IP, allowing for the necessary adjustments in the code. For more permanent solutions, it is advisable to reserve the IP address based on the Raspberry Pi's MAC address to ensure consistent and reliable communication.

The Raspberry Pi sets up a socket server and awaits an incoming connection. Once a connection is established successfully, it starts receiving data from the Arduino. The received data is processed in batches of ten values to achieve a smoother output by calculating their average. These averaged values are then decoded, removing any trailing characters, and split into individual data points using a pre-defined delimiter.

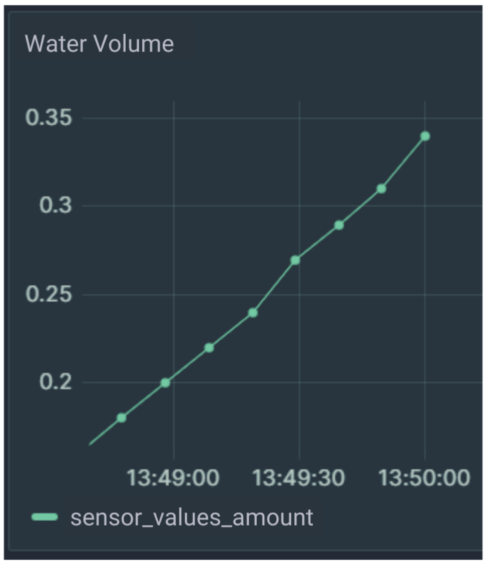
After processing the data, the socket connection with the Arduino is closed, and the Raspberry Pi continues to listen for new connections. Additionally, a connection is established with InfluxDB. Each received data point is transformed into an InfluxDB data format, with each point collected into a list. Two columns were created: *sensor_values_flow* and *sensor_values_tds*.

Finally, the Raspberry Pi writes the collected data points into the InfluxDB database. This involves making requests to the InfluxDB API using HTTP POST requests with the data formatted in the InfluxDB line protocol. These HTTP POST requests can be used to write data points, query data, create databases, etc. The data points are formatted using the InfluxDB line protocol, which specifies the measurement name, tag sets and associated timestamp.

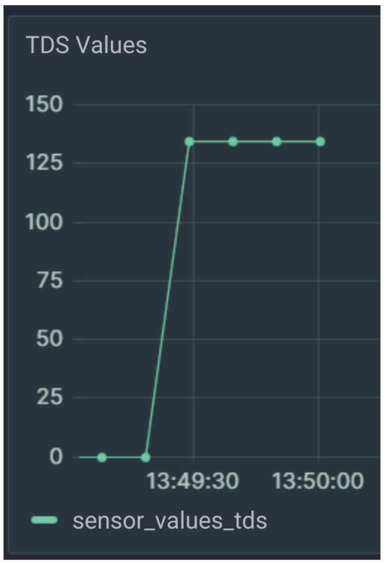
Here is the code snippet that receives values from the Arduino and stores them in the InfluxDB database:

```python
# Receive data from the Arduino
received_data = b''
while True:
    data = conn.recv(1024)
    if not data:
        break
    received_data += data
values = received_data.decode().strip().split(DELIMITER)
store_data_in_influxdb(values)
```

After the sensor data is set in the InfluxDB database, with two columns, one for each sensor, this data needs to be passed onto Grafana. To do this, Grafana was installed and all the database-related information was specified, using InfluxDB as a data source. Afterward, a panel for each variable was created to measure each variable, with customizable time ranges and formats.



(a) Water amount values



(b) TDS values

Figure 6.12: Sensor values displayed in Grafana

The data is then passed from InfluxDB to Grafana through a data source integration. Initially the necessary InfluxDB details (server URL, credentials and database) were provided, and from then on Grafana can retrieve data by sending queries to the InfluxDB API. These queries specify the desired time range, measurement or fields to retrieve the relevant data points.

Grafana uses an internal query editor interface to construct the queries, which are sent using HTTP requests as GET requests. InfluxDB processes the queries and returns the data in JSON. Grafana finally uses that data to create visualizations through customizable dashboards.

For example, in order to retrieve all values from the column of TDS values within a specific timeframe of ten minutes on the date of 1st June 2023, the appropriate query would be:

```
SELECT *
FROM "sensor_values_tds"
WHERE time >= '2023-06-01T17:45:00Z'
  AND time <= '2023-06-01T17:55:00Z'
```

In Figure 6.12, two panels are presented, one for the water amount values (6.12a) and one for the TDS values (6.12b). This test was made by gradually increasing the water content of a cup in an almost-linear way. Since the water was passed from one cup to another, it makes sense for the TDS values to be almost homogeneous, with a small variation that accounts for differences related to external factors.

It is intriguing to note that there is a noticeable time difference between when the water flow sensor began detecting non-null values, approximately at 13:48:40, and when the TDS started acquiring values, which occurred around 13:49:30. This time disparity can be attributed to the inherent nature of the respective sensors and their mechanisms of measurement. Specifically, while the water flow sensor quickly detects change in flow, the TDS has to rely on chemical processes to analyze the samples, which might takes more time, including for sample stabilization.

Meanwhile, in the Arduino, the power consumption is shown in Figure 6.13. These values were obtained using the aforementioned setup, during data transmission, starting with the sensor acquired data, going through the Raspberry Pi, using an average filter for outlier removal, and redirecting data to InfluxDB and Grafana. The peaks in power consumption represent the instances when the Arduino board connects to Wi-Fi and sends data.

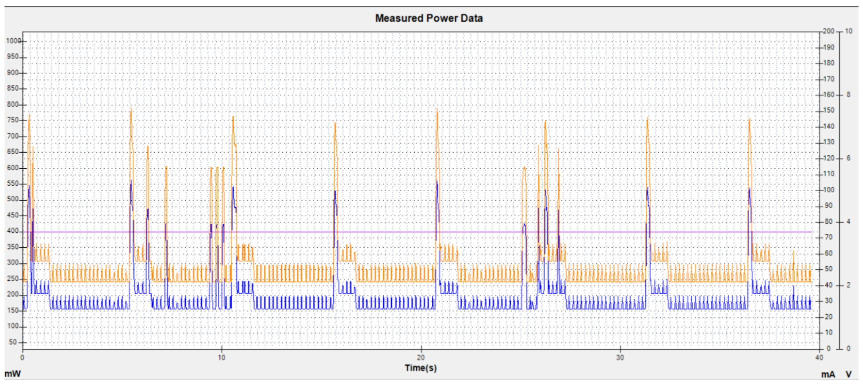


Figure 6.13: Power consumption values during periodic data transmission

6.5 Summary

In this section, we discussed the microcontroller module, which is a crucial component of the water monitoring and reuse system. We highlighted the use of a secondary Li-Po battery and the TP4056 battery charger module for portable and rechargeable power supply. Additionally, we introduced the water turbine as an energy source for the module and explained its role in generating a stable 5V DC output to power the battery charger module.

We obtained insightful results regarding the varying levels of TDS values in different water sources. The same was done for the water flow sensor in order to calculate the accumulated water amount in real-time. Furthermore, we explained the component integration and the data transmission process between the microcontroller and the Raspberry Pi. Lastly, we touched upon the data storage in the InfluxDB database and visualization using Grafana, demonstrating the overall system architecture and data flow.

Chapter 7

Conclusion and Future Work

IoT has been extensively investigated in the development of smart meters for the analysis of water consumption. By harnessing the power of the IoT, we can examine the potential of embedded systems to collect, monitor, and analyze data on water usage in real-time. The implementation of smart meters facilitates the retrieval of valuable water statistics for water reuse and conservation efforts, while also providing a platform for further analysis and filtering to identify patterns, anomalies, and opportunities for optimization.

This dissertation has specifically delved into the development of an embedded IoT system designed for real-time monitoring and analysis of water consumption. We have investigated the power consumption modes of three different microcontroller options: Arduino Nano 33 IoT, ESP8266, and ESP32. By analyzing power consumption patterns during data transmission, we have observed that peaks in power usage correspond to instances of Wi-Fi connectivity and data transmission. Our analysis of power consumption patterns during data transmission has led us to conclude that the Arduino Nano 33 IoT exhibits lower efficiency in achieving low power consumption values compared to the ESP8266 and ESP32 microcontroller options. This observation can be attributed to the lack of optimized sleep libraries for the Arduino Nano 33 IoT and the utilization of peripherals that require higher power compared to the ESPs. These findings contribute to a comprehensive understanding of energy consumption in IoT-based water meters.

Moreover, we have proposed and successfully implemented the various components of a water meter system with energy constraints. The deployment of the necessary sensors for water quality data retrieval, utilizing the Arduino Nano 33 IoT, has been accomplished. To enable seamless communication between the Arduino board and the Raspberry Pi, we established a TCP socket connection over Wi-Fi, allowing the Raspberry Pi to receive sensor data. The Raspberry Pi then utilized HTTP POST requests to interface with the InfluxDB API, allowing for data point storage. Finally, data could be visualized through the Grafana dashboard, providing users with convenient access to their usage patterns.

7.1 Future Work

There are several avenues for improving and expanding the system proposed in this dissertation.

One of the primary areas for improvement in the proposed system is the measurement of water quality. Currently, the system is based solely on Total Dissolved Solids (TDS) values as a basis to determine water quality. This sensor is not comprehensive enough to capture the full spectrum of water contaminants. Therefore, in future iterations of the system, additional sensors, such as pH meters, can be integrated, enabling more accurate filtration and a better assessment of the suitability of the water for flushing.

Another avenue for improvement is the integration of the system with ESP boards. As observed in the previous study, ESP boards exhibit more energy-efficient sleep modes compared to the Arduino Nano 33 IoT. Energy efficiency is a crucial requirement for this system, as it directly impacts the longevity of battery-powered devices.

In the future, this system has the potential to be enhanced by integrating it with the existing flushing module. Additional water flow sensors can be incorporated after water filtration to enable complete water reuse and provide information on post-filtration water quality. Moreover, it would be interesting to implement user alerts that monitor water pollution levels. These alerts would not only indicate the level of water pollution but also provide relevant information to users. For instance, the alerts could inform users about the amount of water wasted due to its unsuitability for reuse, or the quantity of clean water consumed instead. By offering such information and alerts, the system would empower users by providing them with valuable insights about their water usage patterns and the potential for water conservation.

References

[1] Sam d21 datasheet. URL: https://content.arduino.cc/assets/mkr-microchip_samd21_family_full_datasheet-ds40001882d.pdf?_gl=1*k7l70y*_ga*MTIyNzAxNzY2OC4xNjg2OTA5NjY3*_ga_NEXN8H46L5*MTY4NzEwMTczNC4zLjEuMTY4NzEwMjA3MS4wLjAuMA..

[2] Esp8266ex datasheet. URL: https://ecksteinimg.de/Datasheet/CP06031/Datasheet.pdf.

[3] Esp32 datasheet. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[4] Binyam Seboka, Misrak Negashe, Delelegn Yehualashet, Chalachew Kassawe, Mulugeta Namaro, and Mahlet Yigeremu. Health literacy and health information sources in relation to foodborne and waterborne diseases among adults in gedeo zone, southern ethiopia, 2022: A community-based cross-sectional study. *Heliyon*, 9, 2023. doi:10.1016/j.heliyon.2023.e15856.

[5] Huayao Li, Xinqiang Du, Xiangqin Lu, and Min Fang. Analysis of groundwater overexploitation based on groundwater regime information, 2022. doi:10.1111/gwat.13285.

[6] Water use in europe — quantity and quality face big challenges — european environment agency, 2014. URL: https://www.eea.europa.eu/signals/signals-2018-content-list/articles/water-use-in-europe-2014.

[7] Arjen van de Walle, Minseok Kim, Md Alam, Xiaofei Wang, Di Wu, Smruti Ranjan Dash, Korneel Rabaey, and Jeonghwan Kim. Greywater reuse as a key enabler for improving urban wastewater management, 2023. doi:10.1016/j.ese.2023.100277.

[8] Peter Gleick and Heather Cooley. Freshwater scarcity. *Annual Review of Environment and Resources*, 46, 10 2021. doi:10.1146/annurev-environ-012220-101319.

[9] Water costs on the rise across western europe. URL: https://www.mynewsdesk.com/smartvatten/blog_posts/water-costs-on-the-rise-across-western-europe-106080.

[10] Hang Song. Chapter1 of "internet of everything:key technologies, practical applications and security of iot" : What is iot on earth, 2023. doi:10.1142/9789811246272_0002.

[11] Nitin Naik. Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7, 2017. doi:10.1109/SysEng.2017.8088251.

[12] Tetsuya Yokotani and Yuya Sasaki. Comparison with http and mqtt on required network resources for iot. In *2016 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, pages 1–6, 2016. doi:10.1109/ICCEREC.2016.7814989.

[13] What is mqtt? URL: https://www.paessler.com/it-explained/mqtt.

[14] A. Rahman and E. Dijk. Group communication for the constrained application protocol, 10 2014. URL: https://www.rfc-editor.org/info/rfc7390, doi:10.17487/RFC7390.

[15] Influxdata releases influxdb. URL: https://venturebeat.com/enterprise-analytics/influxdata-releases-influxdb-3-0-product-suite-for-time-series-ana

[16] Prometheus overview. URL: https://prometheus.io/docs/introduction/overview/.

[17] Timescaledb hypertables. URL: https://docs.timescale.com/use-timescale/latest/hypertables/about-hypertables/.

[18] Manish Saraswat and R.C. Tripathi. Cloud computing: Comparison and analysis of cloud service providers-aws, microsoft and google. In *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*, pages 281–285, 2020. doi:10.1109/SMART50582.2020.9337100.

[19] Current iot forecast highlights. URL: https://transformainsights.com/research/forecast/highlights.

[20] Ramon Sanchez-Iborra and Maria Dolores Cano. State of the art in lp-wan solutions for industrial iot services. *Sensors (Basel, Switzerland)*, 16, 5 2016. URL: /pmc/articles/PMC4883399//pmc/articles/PMC4883399/?report=abstracthttps://www.ncbi.nlm.nih.gov/pmc/articles/PMC4883399/, doi:10.3390/S16050708.

[21] Kais Mekki, Eddy Bajic, Frederic Chaxel, and Fernand Meyer. A comparative study of lpwan technologies for large-scale iot deployment. *ICT Express*, 5:1–7, 3 2019. doi:10.1016/J.ICTE.2017.12.005.

[22] V.P.G. Jimenez and A.G. Armada. Bit-loaded h-ofdm to increase capacity in wlan/wpan. In *IEEE 60th Vehicular Technology Conference, 2004. VTC2004-Fall. 2004*, volume 5, pages 3373–3377 Vol. 5, 2004. doi:10.1109/VETECF.2004.1404689.

[23] Todd Kennedy and Ray Hunt. A review of wpan security: Attacks and prevention. In *Proceedings of the International Conference on Mobile Technology, Applications, and Systems*, Mobility '08, New York, NY, USA, 2008. Association for Computing Machinery. URL: https://doi.org/10.1145/1506270.1506342, doi:10.1145/1506270.1506342.

[24] Elke Mackensen, Matthias Lai, and Thomas M. Wendt. Bluetooth low energy (ble) based wireless sensors. In *SENSORS, 2012 IEEE*, pages 1–4, 2012. doi:10.1109/ICSENS.2012.6411303.

[25] Pedro de Castro Albergaria. Remote biometrical monitoring system via iot, 2020. URL: https://hdl.handle.net/10216/132783.

[26] F. Campoccia, E. Riva Sanseverino, and G. Zizzo. Analysis of the limitations of wifi communications managed by the ieee 802.11 protocol in data transmission in automated power distribution systems. In *SPEEDAM 2010*, pages 352–357, 2010. doi:10.1109/SPEEDAM.2010.5545051.

[27] Aymen Zreikat. Performance evaluation of 5g/wifi-6 coexistence. *International Journal of Circuits, Systems and Signal Processing*, 14:904–913, 2020. doi:10.46300/9106.2020.14.116.

[28] What are the different types of sensors and their applications? URL: https://www.smlease.com/entries/automation/what-are-different-types-of-sensors-and-their-applications/.

[29] Fernanda Lopez Ornelas. *The Mexican Water Forest: benefits of using remote sensing techniques to assess changes in land use and land cover.* PhD thesis, 05 2016. doi:10.13140/RG.2.2.11855.43685.

[30] Maureen Hodgins William Deoreo. Residential end uses of water, version 2, 2016. URL: https://www.waterrf.org/research/projects/residential-end-uses-water-version-2.

[31] Marco Carratù, Consolatina Liguori, and Antonio Pietrosanto. Sensitivity of water meters to small leakage. *Measurement*, 168, 10 2020. doi:10.1016/j.measurement.2020.108479.

[32] Yang Liu, Xuehui Ma, Yuting Li, Yong Tie, Yinghui Zhang, and Jing Gao. Water pipeline leakage detection based on machine learning and wireless sensor networks. *Sensors*, 19:5086, 11 2019. doi:10.3390/s19235086.

[33] Xuan Hu, Yinlan Han, Bin Yu, Zhiqiang Geng, and Jinzhen Fan. Novel leakage detection and water loss management of urban water supply network using multiscale neural networks. *Journal of Cleaner Production*, 278:123611, 08 2020. doi:10.1016/j.jclepro.2020.123611.

[34] A Amir, Ade Rahman Fauzi, and Yusnaini Arifin. Smart water meter for automatic meter reading. *IOP Conference Series: Materials Science and Engineering*, 1212:012042, 01 2022. doi:10.1088/1757-899X/1212/1/012042.

[35] Vojkan Tasic, Thorsten Staake, Thomas Stiefmeier, Verena Tiefenbeck, Elgar Fleisch, and Gerhard Troster. Self-powered water meter for direct feedback, 10 2012. doi:10.1109/IOT.2012.6402300.

[36] Jae Yong Cho, Jae Choi, Sinwoo Jeong, Jung Ahn, Won Hwang, Hong Yoo, and T. Sung. Design of hydro electromagnetic and piezoelectric energy harvesters for a smart water meter system. *Sensors and Actuators A: Physical*, 261, 05 2017. doi:10.1016/j.sna.2017.05.018.

[37] Xue Li and P.H.J. Chong. Design and implementation of a self-powered smart water meter. *Sensors*, 19:4177, 09 2019. doi:10.3390/s19194177.

[38] Jin Wang, Rachel Cardell-Oliver, and Wei Liu. Discovering routine behaviours in smart water meter data, 2015. doi:10.1109/ISSNIP.2015.7106899.

[39] L. Mezzera, M. Carminati, M. Di Mauro, A. Turolla, M. Tizzoni, and M. Antonelli. A 7-parameter platform for smart and wireless networks monitoring on-line water quality. In *2018 25th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 709–712, 2018. doi:10.1109/ICECS.2018.8618014.

[40] Santosh Gautam Kashid and Sanjay A. Pardeshi. Intelligent water metering system: An image processing approach (matlab simulations), 2014. doi:10.1109/CNSC.2014.6906676.

[41] Lucas Pereira, Donovan Costa, and Miguel Ribeiro. A residential labeled dataset for smart meter data analytics. *Scientific Data*, 9, 12 2022. doi:10.1038/s41597-022-01252-2.

[42] Rachmad Atmoko, Rona Riantini, and M Hasin. Iot real time data acquisition using mqtt protocol. *Journal of Physics: Conference Series*, 853:012003, 05 2017. doi:10.1088/1742-6596/853/1/012003.

[43] Rajan m a, Shivraj V L, and Balamuralidhar Purushothaman. Secure mqtt for internet of things (iot), 04 2015. doi:10.1109/CSNT.2015.16.

[44] Rachmad Atmoko, Rona Riantini, and M Hasin. Iot real time data acquisition using mqtt protocol. *Journal of Physics: Conference Series*, 853:012003, 05 2017. doi:10.1088/1742-6596/853/1/012003.

[45] Dhvani Shah and Vinayak Bharadi. Iot based biometrics implementation on raspberry pi. *Procedia Computer Science*, 79:328–336, 12 2016. doi:10.1016/j.procs.2016.03.043.

[46] Sawsan Sayed, Tasnim Hussain, Adel Gastli, and Mohieddine Benammar. Design and realization of an open-source and modular smart meter. *Energy Science & Engineering*, 7:1405–1422, 8 2019. URL: https://onlinelibrary.wiley.com/doi/full/10.1002/ese3.361https://onlinelibrary.wiley.com/doi/abs/10.1002/ese3.361https://onlinelibrary.wiley.com/doi/10.1002/ese3.361, doi:10.1002/ESE3.361.

[47] M Suresh, U. Muthukumar, and Jacob Chandapillai. A novel smart water-meter based on iot and smartphone app for city distribution management. In *2017 IEEE Region 10 Symposium (TENSYMP)*, pages 1–5, 2017. doi:10.1109/TENCONSpring.2017.8070088.

[48] G. M. Madhu, C. Vyjayanthi, and Chirag Modi. An open-hardware approach for iot enabled smart meter monitoring and controlling system using mqtt protocol. *Communications in Computer and Information Science*, 1192 CCIS:303–317, 2020. doi:10.1007/978-981-15-3666-3_25.

[49] Paul Macheso and Angel Meela. Iot based patient health monitoring using esp8266 and arduino. *International Journal of Computer Communication and Informatics*, 3:75–83, 10 2021. doi:10.34256/ijcci2127.

[50] Sangeethalakshmi Kannan, Preethi S., Preethi U., Pavithra S., and Shanmuga V. Patient health monitoring system using iot. *Materials Today: Proceedings*, 80, 06 2021. doi:10.1016/j.matpr.2021.06.188.

[51] Ahmad Dziaul Islam Abdul Kadir, Mohamad Razin Naim Mohd Alias, Dzahir Rashidi Mohd Dzaki, Azizul Azizan, and Norashidah Md Din. Mobile IoT cloud-based health monitoring dashboard application for the elderly. In *2022 4th International Conference on Smart Sensors and Application (ICSSA)*, pages 161–166, 2022. doi:10.1109/ICSSA54161.2022.9870913.

[52] Arduino nano 33 iot with headers. URL: https://store.arduino.cc/products/arduino-nano-33-iot-with-headers.

[53] Nodemcu board esp8266 wifi module esp-12e lua wifi. URL: https://www.ptrobotics.com/wifi/5600-nodemcu-board-esp8266-wifi-module-esp-12e-lua-wifi.html.

[54] Espressif. Esp8266 low power solutions, 2016. URL: https://www.espressif.com/sites/default/files/9b-esp8266-low_power_solutions_en_0.pdf.

[55] Esp-32s development board. URL: https://www.fruugo.pt/esp-wroom-32-esp32-esp32-esp-32s-development-board-24ghz-dual-mode-wifi-bluetooth-p-63691741-128477193.

[56] Arduino nano 33 iot datasheet. URL: https://docs.arduino.cc/resources/datasheets/ABX00027-datasheet.pdf?_gl=1*1tv0jpz*_ga*MTIyNzAxNzY2OC4xNjg2OTA5NjY3*_ga_NEXN8H46L5*MTY4NzkONTgyOC4xMS4xLjE2ODc5NDU5MTMuMC4wLjA.

[57] João Tiago e Teixeira Mesquita. Comunicação wifi para monitorização móvel de sinais fisiológicos, 2018. URL: https://hdl.handle.net/10216/113805.

[58] Clyde Cox. How to power an arduino with a battery - circuit basics, 2021. URL: https://www.circuitbasics.com/how-to-choose-the-right-battery-to-power-up-your-arduino/.

[59] Tsuyoshi Sasaki, Yoshio Ukyo, and Petr Novák. Memory effect in a lithium-ion battery. *Nature Mater 12*, 12, Apr 2013. URL: https://www.nature.com/articles/nmat3623?_gclid=5b7691afab6304.84507596-5b7691afab6387.76857734&_utm_source=xakep&_utm_campaign=mention49257&_utm_medium=inline&_utm_content=lnk330589340640#citeas, doi:https://doi.org/10.1038/nmat3623.

[60] 3.7v li-po battery. URL: https://mauser.pt/catalog/product_info.php?cPath=74_1990&products_id=035-9012.

[61] Tp4056 battery charging module datasheet. URL: https://storage.googleapis.com/mauser-public-images/prod_description_document/2022/185/6c8c2e2343ad394269298aba84efd12a_tp4056.pdf.

[62] Mcp1700 datasheet. URL: https://ww1.microchip.com/downloads/en/DeviceDoc/MCP1700-Low-Quiescent-Current-LDO-20001826E.pdf.

[63] Water turbine generator. URL: https://www.dfrobot.com/product-1610.html.

[64] Qualidade da Água - sensor tds. URL: https://wiki.dfrobot.com/Gravity__Analog_TDS_Sensor___Meter_For_Arduino_SKU__SEN0244.

[65] G3/4 water flow sensor. URL: https://wiki.dfrobot.com/Water_Flow_Sensor_-_1_2__SKU__SEN0217.

[66] Robert Karplus and J. M. Luttinger. Hall effect in ferromagnetics. *Phys. Rev.*, 95:1154–1160, Sep 1954. URL: https://link.aps.org/doi/10.1103/PhysRev.95.1154, doi:10.1103/PhysRev.95.1154.

[67] Yanwen Xu, Anand Lalwani, Kanika Arora, Zhuoyuan Zheng, Anabel Renteria, Debbie Senesky, and Pingfeng Wang. Hall-effect sensor design with physics-informed gaussian process modeling, 2022. doi:10.1109/JSEN.2022.3216499.

[68] People's showering habits revealed in survey. URL: https://www.bbc.com/news/science-environment-15836433.

[69] Household composition statistics. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Household_composition_statistics#:~:text=Highlights&text=In%202022%2C%20198%20million%20households,members%20per%20household%20on%20average.

[70] What is a raspberry pi? URL: https://th.cytron.io/tutorial/what-is-a-raspberry-pi.