

A project report on

**DEVELOPMENT OF MQTT PROTOCOL-
BASED CLIENT-SERVER SYSTEM FOR IOT
APPLICATIONS IN NUCLEAR POWER
PLANTS**

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology

in

**Electronics and Communication
Engineering**

By

DHRUVA RAKESH B (19BES7003)



**VIT-AP
UNIVERSITY**

**SCHOOL OF ELECTRONICS ENGINEERING (SENSE)
VIT-AP UNIVERSITY
AMARAVATI- 522237**

APRIL 2023

DEVELOPMENT OF MQTT PROTOCOL- BASED CLIENT-SERVER SYSTEM FOR IOT APPLICATIONS IN NUCLEAR POWER PLANTS

Submitted in partial fulfilment for the award of the degree of

Bachelor of Technology

in

**Electronics and Communication
Engineering**

By

DHRUVA RAKESH B (19BES7003)



**VIT-AP
UNIVERSITY**

**SCHOOL OF ELECTRONICS ENGINEERING (SENSE)
VIT-AP UNIVERSITY
AMARAVATI- 522237**

April 2023

DECLARATION

I hereby declare that the thesis entitled “DEVELOPMENT OF MQTT PROTOCOL BASED CLIENT SERVER SYSTEM FOR IOT APPLICATIONS IN NUCLEAR POWER PLANTS” submitted by me, for the award of the degree of Bachelor of Technology in Electronics and Communication Engineering with Specialization in Embedded Systems to VIT is a record of bonafide work carried out by me under the supervision of Sreejith Sasidharan.

I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Place: Amaravati


Date: 28-04-2023



Signature of the Candidate

CERTIFICATE

This is to certify that the Internship titled “**DEVELOPMENT OF MQTT PROTOCOL BASED CLIENT SERVER SYSTEM FOR IOT APPLICATIONS IN NUCLEAR POWER PLANTS**” that is being submitted by **Dhruva Rakesh B (19BES7003)** is in partial fulfilment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma and the same is certified.


28/4/23
(Sreejith Sasidharan)
Electronics Engineering
इंजीनियरिंग
Indira Gandhi
कल्याणम् / Kalyan

The thesis is satisfactory/unsatisfactory

Internal Examiner

External Examiner

Approved by

PROGRAM CHAIR

B. Tech. ECE

DEAN

School Of Electronics
Engineering



Government of India
Department of Atomic Energy
Indira Gandhi Centre for Atomic Research
Kalpakkam – 603102. Tamil Nadu, INDIA

Sreejith Sasidharan
Scientific Officer E
Real Time Systems Division
Electronics & Instrumentation Group
Indira Gandhi Center for Atomic Research

Phone : 044-27480500
Mobile : 9445607184
E-mail : sreejith@igcar.gov.in


April 28, 2023

BONAFIDE CERTIFICATE

This is to certify that **Shri. Dhruva Rakesh B** (Roll No: 19BES7003) student of Fourth Year B.Tech. (Electronics and Communication Engineering), Vellore Institute of Technology, AP, has executed the project titled "**Development of mqtt protocol based client server system for IoT applications in Nuclear power plants**" as part of his internship at Real Time Systems Division, Electronics & Instrumentation Group of Indira Gandhi Centre for Atomic Research, Kalpakkam from **07/11/2022 to 28/04/2023**.

During this period, he showed excellent understanding of C++ programming Language, node.js, Qt framework, communication protocols and Linux based software development. He has also developed an MQTT protocol based client server communication system and demonstrated the communication capabilities of MQTT based IoT data exchange architecture.

His conduct and character during this period was very good. I wish all success in his future endeavours.


(Sreejith Sasidharan)

Electronics & Instrumentation Group
Indira Gandhi Centre for Atomic Research
Kalpakkam / Tamil Nadu - 603102

ABSTRACT

The emergence of the Internet of Things (IoT) has led to the development of a wide range of smart devices, which are interconnected and communicate with each other to share data and perform complex tasks. The increasing complexity of IoT networks has created a need for reliable and secure communication channels that ensure the privacy and safety of users and data. One of the critical aspects of IoT security is the authentication of devices and users, which helps prevent unauthorized access and data breaches.

This project aims to develop a reliable and secure MQTT broker and client system to support data acquisition for continuous power plant equipment condition monitoring and predictive maintenance. The MQTT broker will be the central hub for data transmission and management activities and will be configured to support multiple clients while ensuring that only authorized devices and users have access to the system.

The MQTT client will serve as the primary interface between the user and the MQTT broker, and it will be designed to establish a secure and reliable communication channel with the broker for the transfer of data wirelessly through the MQTT protocol. The implementation of this MQTT broker and client system will provide an efficient and effective solution for secure data transfer and management in the power plant industry.

The primary objective of this project is to develop a robust and efficient data transfer system that can reliably connect the power plant equipment system with the IoT platform. The system should be easy to use, scalable, and secure, ensuring that users can easily and safely access the IoT platform.

ACKNOWLEDGEMENT

I am pleased to express with a deep sense of gratitude to Mr Sreejith Sasidharan, SO/F, RTSD/EIG, IGCAR, for his constant guidance and encouragement, understanding; more than all, he taught me patience in my endeavour. My association with him is not confined to academics only, but it is a great opportunity on my part to work with an intellectual and expert in the field of IoT.

I would like to express my gratitude to Dr G. Viswanathan Founder & Chancellor, VIT, Sankar Viswanathan Vice President, VIT, Dr Sekar Viswanathan Vice President, VIT, G. V. Selvam Vice President, VIT, Dr Sandhya Pentareddy Executive Director, Dr S. V. Kota Reddy Vice Chancellor Dr Jagadish Chandra Mudiganti Registrar and Dr Umakanta Nanda Dean, School of Electronics Engineering, for providing with an environment to work in and for his inspiration during the tenure of the course.

In a jubilant mood, I express ingeniously my whole-hearted thanks to Dr Samineni Peddakrishna, Assistant Professor Sr. Grade-1, School of Electronics Engineering, all teaching staff and members working as limbs of our university for their not-self-centred enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. Last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Amaravati

Date: 28-04-2023

Dhruva Rakesh B

ACKNOWLEDGEMENT

I humbly and sincerely pray to the Almighty and my dear Parents for their support and blessings in all my actions.

I wish to express my gratitude to, Dr B. Venkatraman, Director, IGCAR- Kalpakkam and Mr S. Raghupathy, Director of Electronics and Instrumentation Group, for permitting and providing a wonderful opportunity to carry out my internship at the research centre.

I extend my gratitude to Dr D. Thrignana Murthy, Associate Director of Electronics and Computers Group, Electronics and Instrumentation Group, for allowing me to learn and work in various spectroscopical and analytical techniques.

I submit my sincere thanks to R. P. Behra, Scientific Officer SO/G, Head, Real-Time System Division (RTSD) Division, Electronics and Instrumentation Group, for his guidance, support and kind words, which have enabled me to complete my training successfully.

I convey my heartfelt thanks to Mr. Sreejith Sasidharan, who has been my Guide throughout the session with constant support and help which paved the way for the successful completion of the internship.

Dhruva Rakesh B

CONTENTS

ABSTRACT	6
CONTENTS	9
LIST OF TABLES	11
LIST OF FIGURES.....	12
LIST OF SYMBOLS	13
Chapter 1	14
Introduction	14
1.1 BACKGROUND AND SIGNIFICANCE	14
1.1.1 OVERVIEW OF INDIRA GANDHI CENTRE FOR ATOMIC.....	14
RESEARCH (IGCAR).....	14
1.1.2 DIVISIONS IN IGCAR.....	16
1.1.3 ELECTRONICS AND INSTRUMENTATION GROUP (EIG)	17
1.1.4 REAL-TIME SYSTEMS DIVISION (RTSD)	18
1.2 PROBLEM STATEMENT	19
1.3 OBJECTIVES & RESEARCH	20
1.3.1 MOTIVATION.....	20
1.4 SCOPE & LIMITATIONS	21
Chapter 2	23
Literature Review	23
2.1 INTERNET OF THINGS (IOT) & NUCLEAR POWER PLANTS	23
2.2 MQTT PROTOCOL & ITS APPLICATIONS.....	24
2.3 HOW MQTT WORKS.....	25
2.3.1 IMPORTANT POINTS TO NOTE.....	26
2.3.2 MQTT CLIENT BROKER CONNECTIONS	26
2.3.3 THE CLIENT NAME / ID	27
2.3.4 CLEAN SESSIONS	27
2.3.5 LAST WILL MESSAGES	28
Chapter 3	29
Methodology	29
3.1 PROPOSED SOLUTION	29
3.2 SYSTEM ARCHITECTURE AND DESIGN	30
3.3 IMPLEMENTATION & TESTING	32

3.4	DESIGN APPROACH AND DETAILS.....	33
3.4.1	ARDUINO MEGA:.....	33
3.4.2	ETHERNET SHEILD:	34
3.4.3	QT CREATOR	34
3.4.4	VISUAL BASIC CODE:.....	35
3.5	CONSTRAINTS, ALTERNATIVES & TRADEOFFS	36
3.5.1	USING ONLY C/C++	36
3.5.2	USAGE OF ONLY LINUX BASED SYSTEMS	37
3.5.3	SAFETY CRITICAL APPLICATIONS	38
	Chapter 4	40
	Results and Analysis	40
4.1	DATA COLLECTION AND TRANSMISSION	40
4.2	SERVER-SIDE PROCESSING AND VISUALISATION.....	41
4.2.1	TESTING USING ARDUINO & ETHERNET SHIELD	42
4.2.2	TESTING USING MOSQUITTO SERVER AND CLIENT	43
4.2.3	TESTING USING HIVEMQ BROKER	45
	Chapter 5	52
	Conclusion & Future Work.....	52
	REFERENCES.....	54
	Appendix A	56
	Source code for MQTT Client Application	56
A.1	SIMPLE MQTT CLIENT APPLICATION.....	56
A.2	CODE FILES	56
A.2.1	CMakeLists.txt	56
A.2.2	main.cpp	58
A.2.3	mainwindow.cpp.....	58
A.2.4	mainwindow.h	60
A.2.5	mainwindow.ui	61
A.2.6	simpleclient.pro	65

LIST OF TABLES

Table 1	MAIN FEATURES OF MQTT	25
Table 2	APPLICATIONS OF MQTT.....	28
Table 3	SYSTEM ARCHITECTURE & DESIGN	31

LIST OF FIGURES

1	Fig. 2.1 MQTT Publish / Subscribe Architecture	24
2	Fig. 2.2 MQTT message flow	27
3	Fig. 3.1 MQTT Broker Architecture	31
4	Fig. 3.2 MQTT Process.....	32
5	Fig 4.1: Arduino code and messages published, subscribed.....	41
6	Fig. 4.2 Arduino as a MQTT Server	42
7	Fig. 4.3 Arduino as a MQTT Client.....	43
8	Fig. 4.4 Starting mosquito server.....	44
9	Fig. 4.5 Mosquito subscribe client.....	44
10	Fig. 4.6 Mosquito publish client	45
11	Fig. 4.7 Starting HiveMQ Server.....	45
12	Fig. 4.8 Started HiveMQ Broker.....	46
13	Fig. 4.9 HiveMQ Dashboard.....	47
14	Fig. 4.10: QT code 1	48
15	Fig. 4.11: QT code 2	48
16	Fig. 4.12: UI design of the client	49
17	Fig. 4.13: Dashboard.....	50
18	Fig. 4.14: Dashboard of the Broker after giving 5 rounds of publish/subscribe...50	
19	Fig. A.1 simple MQTT Client GUI Application.....	56

LIST OF SYMBOLS

Indhira gandhi Center for Atomic
Reasearch

IGCAR, 14

Internet of Things

IoT. *See*

Message Queuing Telemetry Transport

MQTT. *See*

Vellore Institute of Technology

VIT. *See*

Chapter 1

Introduction

1.1 BACKGROUND AND SIGNIFICANCE

The Internet of Things (IoT) has revolutionized the way devices communicate with each other, and its implementation in nuclear power plants can significantly improve their efficiency, safety, and reliability. MQTT (Message Queuing Telemetry Transport) is a lightweight messaging protocol widely used in IoT applications due to its low bandwidth requirements and high scalability.

The proposed system aims to develop a client-server architecture based on MQTT protocol for real-time monitoring of various sensors in nuclear power plants. This will enable the operators to have instant access to the data collected by sensors like temperature, pressure, radiation levels, and flow rates, facilitating timely decision-making, and enhancing safety.

The significance of this research lies in the fact that it offers a cost-effective and reliable solution for the implementation of IoT in nuclear power plants, which can improve the plant's performance, reduce maintenance costs, and mitigate safety risks. The research will provide valuable insights into the development and implementation of MQTT-based systems in the nuclear power industry.

1.1.1 OVERVIEW OF INDIRA GANDHI CENTRE FOR ATOMIC RESEARCH (IGCAR)

Indira Gandhi Centre for Atomic Research [IGCAR], the second largest establishment of the Department of Atomic Energy next to Bhabha Atomic Research Centre, was set up at Kalpakkam [80 KMs south of Chennai], in 1971 with the main objective of conducting a broad-based multidisciplinary programme of scientific research and advanced Engineering, directed towards the development of sodium-cooled Fast Breeder Reactor [FBR] technology, in India. This is part of the second stage

of the Indian Atomic Energy Programme, which is aimed at preparing the country for utilization of the extensive Thorium reserves and providing means to meet the large demands of electrical energy in the 21st century.

In meeting the objectives, a modest beginning was made by constructing a sodium-cooled Fast Breeder Test Reactor [FBTR], with a nominal power of 40MWt, based on the French Reactor, RAPSODIE. The reactor attained its first criticality on 18th October 1985 and has been in operation at its maximum attainable power level of 10.5 MWt with a small core. It is the first of its kind in the world to use Plutonium Uranium mixed carbide as a driver fuel. Over the years, the centre has established comprehensive R & D facilities covering the entire spectrum of FBR technology related to Sodium Technology, Reactor Engineering, Reactor Physics, Metallurgy and Materials, Chemistry of Fuels and its materials, Fuel Reprocessing, Reactor Safety, Control and Instrumentation, Computer Applications etc., and has developed a strong base in a variety of disciplines related to this advanced technology.

With the experience and expertise gained by the successful operation of FBTR, the Centre has embarked upon the design and construction of a 500 MW, Prototype Fast Breeder Reactor [PFBR]. Various R & D activities in the areas of Structural Mechanics, Thermal Hydraulics and flow-induced vibration, Component Testing in high-temperature sodium environment sodium-water reaction, hydraulic development of sodium pumps etc. were pursued and the design was completed. The PFBR is in the advanced stage of construction and commissioning by BHAVINI.

The centre is committed to carrying out cutting-edge research in various aspects of nuclear energy and technology, including nuclear reactor design and development, nuclear fuel cycle, materials science, and safety research, among others.

IGCAR is one of the few institutions in India that focuses on the entire spectrum of nuclear science and technology. The centre has made significant contributions to the development of India's nuclear power program, including the design and development of nuclear reactors, fuel fabrication, reprocessing, waste management, and safety

analysis. The centre also undertakes research in the areas of basic and applied sciences, materials science, and technology development.

The centre has a highly qualified and experienced workforce, including scientists, engineers, and technicians, who work collaboratively to achieve the centre's mission of developing safe, sustainable, and cost-effective nuclear energy solutions. IGCAR has several world-class research facilities, including nuclear reactors, particle accelerators, and advanced analytical and testing laboratories.

In addition to its research and development activities, IGCAR also provides training and educational programs for students and professionals in various areas of nuclear science and technology. The centre has collaborations with several national and international institutions, including universities, research organizations, and industries, which enables it to access the latest knowledge and expertise in its areas of research.

Broadly, IGCAR plays a vital role in India's nuclear energy program, and its contributions have been recognized nationally and internationally. The centre's commitment to excellence, innovation, and sustainability makes it a prominent institution in the field of nuclear science and technology.

The centre is divided into several groups and divisions, each focusing on a specific area of research and development related to nuclear energy and technology.

1.1.2 DIVISIONS IN IGCAR

The Indira Gandhi Centre for Atomic Research (IGCAR) is a premier research institution that is home to a diverse range of research divisions, each of which is dedicated to exploring different domains of nuclear science and technology. These divisions are staffed by highly trained and experienced researchers and engineers who bring their expertise to bear on some of the most pressing challenges facing the nuclear energy sector today.

At IGCAR, there are many different divisions, each with its unique area of focus. These divisions include, but are not limited to,

- Engineering Services Group (ESG)
 - Material Services and Metal Fuel Cycle Group (MS&MFCG)
 - Metallurgy and Materials Group (MMG)
 - Electronics and Instrumentation Group (EIG)
 - Materials Science Group (MSG)
 - Reactor Design & Technology Group (RDTG)
 - Reactor Facilities Group (RFG)
 - Reprocessing Group (RpG)
 - Safety Quality and Resource Management Group (SQ&RMG)
 - Radiochemistry Laboratory (RCL)
 - Water Steam Chemistry Division (WSCD)
- and many more...

These different divisions represent a diverse range of research activities taking place at IGCAR and are a testament to the institution's commitment to advancing the state of nuclear science and technology in India and beyond.

1.1.3 ELECTRONICS AND INSTRUMENTATION GROUP (EIG)

Electronics and Instrumentation Group is responsible for the design, development, and maintenance of electronic instrumentation and control systems for fast breeder reactors and reprocessing plants. This includes the development of distributed digital control systems, safety-critical and safety-related systems, safe and secure programmable logic controllers (PLCs), virtual control panel-based control rooms, and full-scope operator training simulators.

In addition to these activities, the Electronics and Instrumentation Group is also involved in the design and development of advanced equipment and technology, such as indigenous wireless sensor networks for nuclear facilities, strategic and societal applications, and Time Domain Electromagnetic (TDEM) for Deep-Seated Atomic Minerals Exploration. The group also develops Plutonium Condition Air Monitoring

Systems for reprocessing plants, test instruments for steam generator tube inspection, radar level probes for liquid sodium level measurement, radiation-resistant MEMS-based sensors for nuclear applications, and innovative sensors and instruments for nuclear facilities.

The group has considerable expertise in designing, building, and maintaining state-of-the-art high-performance supercomputing facilities that meet large-scale computing and data-intensive requirements in multi-disciplinary domains. They Also, implement IT-enabled Nuclear Knowledge Management systems for fast reactors and associated domains, computational intelligence systems, cryptography, and cyber-security solutions. The group also develops and deploys modern security systems for access control and physical protection of nuclear complexes.

The Electronics and Instrumentation Group's work is critical to ensuring the safe and efficient operation of nuclear facilities, and their contributions have been recognized nationally and internationally. Their commitment to excellence, innovation and sustainability makes them a vital component of IGCAR's research and development activities in the field of nuclear science and technology.

1.1.4 REAL-TIME SYSTEMS DIVISION (RTSD)

The Real-Time Systems Division (RTSD) is a subdivision of the EIG group at IGCAR and is one of the key divisions at the Indira Gandhi Centre for Atomic Research (IGCAR), which is located in Kalpakkam, India. The division is involved in developing, designing, and testing real-time control systems for nuclear reactors, accelerators, and associated facilities.

The RTSD at IGCAR has expertise in areas such as process control, distributed control systems, embedded systems, and real-time data acquisition and analysis. The division plays a key role in the design, development, and commissioning of control and instrumentation systems for various nuclear facilities in India, including fast breeder reactors, research reactors, and fuel reprocessing plants.

The RTSD has state-of-the-art facilities such as a real-time simulator, a hardware-in-the-loop test bed, and various software development tools. These facilities enable the division to design, develop and test sophisticated control and instrumentation systems that are capable of meeting the stringent safety and reliability requirements of nuclear facilities.

The RTSD has a team of highly qualified and experienced scientists, engineers, and technicians who work together to develop innovative solutions for real-time control and instrumentation systems. The division also collaborates with other research organizations in India and abroad to keep up with the latest developments in this field.

Overall, the RTSD at IGCAR Kalpakkam is a premier research division in the field of real-time systems and plays a crucial role in advancing the state-of-the-art in this field for the Indian nuclear industry.

1.2 PROBLEM STATEMENT

The goal of the project is to develop a Broker and a Client that helps MQTT and establish an MQTT protocol communication among them. The proposed challenge of developing an MQTT broker and the MQTT consumer is in particular aimed toward statistics acquisition for non-stop energy plant gadget condition monitoring and predictive protection at IGCAR. This mission pursues to develop a dependable and secure MQTT broking and patron system to guide statistics acquisition for continuous strength plant equipment circumstance monitoring and predictive upkeep. The MQTT broking might be the relevant hub for statistics transmission and management sports and will be configured to support a couple of clients at the same time as making sure that best authorized gadgets and customers have access to the device.

1.3 OBJECTIVES & RESEARCH

The objective of the project is to develop a Broker and a Client that supports MQTT and establish an MQTT protocol communication between them. The Internet of Things (IoT) has become a significant technological advancement in recent years, allowing for the interconnectivity of devices and the exchange of data. With the growth of IoT, the need for secure and reliable data transmission has become increasingly important. The identity of devices and users accessing IoT systems is crucial for ensuring security, privacy, and trust. The proposed project aims to develop communication that enables the communication and transfer of data wirelessly through the MQTT protocol.

The proposed project of developing an MQTT broker and the MQTT client is specifically aimed at data acquisition for continuous power plant equipment condition monitoring and predictive maintenance at IGCAR. IGCAR deals with sensitive data that requires high accuracy and precision. The proposed project will provide an additional layer of data collection and processing to the IoT systems utilized at IGCAR, ensuring that only authorized personnel have access to the data. The proposed project will add an extra layer of data collecting and processing to the IoT systems used at IGCAR, making sure that only authorized individuals can access the data. This will enhance the overall prediction and reliability of the system, making it more suitable for the sensitive data utilized at IGCAR.

1.3.1 MOTIVATION

The motivation for this project stems from the growing need for secure and reliable data transmission in IoT systems. With the number of interconnected devices increasing rapidly, the data and performance of these devices have become a major concern. In particular, the authentication of devices and users is a critical aspect of IoT security that requires attention. To address this issue, the proposed project aims to develop communication and data acquisition that enables the communication and transfer of data wirelessly through the MQTT protocol.

The integration of an MQTT broker with an IoT platform would enhance accuracy and trust by ensuring that only authorized devices and users have access to the system. The broker would enable the equipment and experiments system to securely transmit data to the IoT platform, thereby preventing unauthorized access and data breaches and space to store all the data which would later be processed and interpreted, and acted upon. This integration would also provide additional trust and knowledge features to the power plant, ensuring that sensitive data is protected and reliable. In summary, the proposed project aims to develop a communication and data acquisition model that enables the communication and transfer of data wirelessly through the MQTT protocol.

The following sections of the report provide details of the system design, implementation, and results. This report presents a comprehensive overview of the system design, implementation, and results of the project. The report is structured as follows, First, we provide a background and motivation for the project, explaining the importance of data storage and monitoring it. Next, we review the existing literature and research on ID verification systems and IoT platforms, analysing and comparing different approaches and solutions. Then, we describe the architecture and components of the Model, including the technical details of the integration process. We also discuss the implementation of the system and the results of the integration process, presenting any performance metrics and data. Finally, we summarize the objectives and contributions of the project, discuss the significance and potential impact of the system, and suggest future work and improvements for the system.

1.4 SCOPE & LIMITATIONS

Data storage and monitoring are essential for maintaining the performance and reliability of equipment. Proper data management can provide insights into the equipment's operation, usage, and condition, allowing for predictive maintenance and optimization. By keeping track of data, equipment owners and operators can identify patterns that may indicate the need for repairs or replacements, preventing costly downtime and extending equipment lifespan.

Real-time monitoring is also crucial for quickly identifying and diagnosing issues as they arise. By tracking and analysing data, operators can detect anomalies, monitor performance indicators, and take corrective action before any problems escalate. This helps to minimize downtime, improve productivity, and optimize performance.

Data storage and monitoring are especially important in industries that require compliance with strict regulations. By maintaining accurate records of equipment usage and performance, operators can ensure compliance with industry standards and regulations, avoiding fines or penalties.

Furthermore, data storage and monitoring provide a wealth of information that can be used to improve equipment performance and optimize operations. By analysing data, operators can identify areas for improvement, reduce energy consumption, and increase efficiency, leading to cost savings and improved productivity.

In conclusion, data storage and monitoring play a critical role in equipment maintenance, performance optimization, compliance, and overall operational efficiency. By utilizing the power of data, equipment owners and operators can make informed decisions and achieve long-term success.

The retention of data and monitoring are critical for sustaining device performance and dependability. Proper administration of information can provide insights into the functioning, usage, and condition of the equipment, enabling predictive maintenance and optimisation. Equipment owners and operators can spot patterns in data that may suggest the need for repairs or replacements, avoiding costly downtime and increasing equipment lifespan.

Real-time monitoring is also essential for recognising and diagnosing problems as they emerge. Operators can discover abnormalities, monitor performance indicators, and take corrective action before problems grow by tracking and analysing data. This reduces downtime increases productivity and optimises performance.

Chapter 2

Literature Review

2.1 INTERNET OF THINGS (IOT) & NUCLEAR POWER PLANTS

The Internet of Things (IoT) is a network of interconnected devices that can communicate and exchange data with each other without human intervention. These devices can be sensors, actuators, cameras, and other types of machines that can collect, analyze and share data over the internet.

In nuclear power plants, IoT has the potential to significantly improve the safety, reliability, and efficiency of operations. IoT sensors can be used to monitor critical parameters such as temperature, pressure, flow rates, and radiation levels, and provide real-time data to plant operators for timely decision-making. By automating the data collection and analysis process, IoT can help plant operators detect anomalies, predict equipment failures, and prevent accidents.

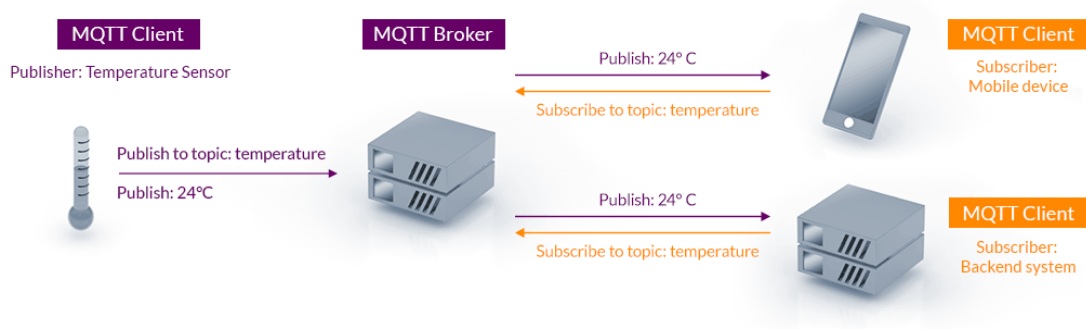
Additionally, IoT can be used for predictive maintenance of equipment, allowing plant managers to identify and address issues before they become major problems. IoT can also help optimize the use of resources like energy and water, reducing costs and improving overall plant performance.

Overall, the implementation of IoT in nuclear power plants can result in improved safety, increased reliability, and enhanced efficiency, making it an attractive option for the nuclear power industry.

2.2 MQTT PROTOCOL & ITS APPLICATIONS

MQTT (Message Queuing Telemetry Transport) is a lightweight publish-subscribe protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. It is used extensively in IoT applications, especially for devices that have limited resources, and requires a minimum amount of bandwidth and power consumption.

MQTT works on a publish-subscribe messaging model, where clients publish messages to a broker, which then distributes the messages to all subscribed clients. This makes it an ideal protocol for IoT applications, where devices can publish data to the broker and any other devices that need that data can subscribe to it.



1 Fig. 2.1 MQTT Publish / Subscribe Architecture

MQTT publish/subscribe pattern (also known as pub/sub) provides an alternative to a traditional client-server architecture. In the client-server model, a client communicates directly with an endpoint. The pub/sub model decouples the client that sends a message (the publisher) from the client or clients that receive the messages (the subscribers). The publishers and subscribers never contact each other directly. They are not even aware that the other exists. The connection between them is handled by a third component (the broker). The job of the broker is to filter all incoming messages and distribute them correctly to subscribers.

The table below summarizes the main features of MQTT:

<i>Features</i>	<i>Description</i>
<i>Publish-Subscribe Model</i>	Clients publish messages to a broker, which distributes the messages to all subscribed clients
<i>QoS</i>	Supports three Quality of Service levels: QoS 0, QoS 1, and QoS 2
<i>Retained Messages</i>	Allows a message to be retained by the broker and delivered to new subscribers when they connect
<i>Last Will and Testament (LWT)</i>	Allows a client to specify a message that will be published to a topic when it disconnects unexpectedly
<i>Lightweight</i>	Designed to be lightweight and efficient, making it ideal for IoT applications

Table 0.1. Main Features of MQTT

2.3 HOW MQTT WORKS

MQTT is a messaging protocol i.e. it was designed for transferring messages, and uses a publish and subscribe model. This model makes it possible to send messages to 0,1 or multiple clients.

A useful analogy is TV or radio. A TV broadcaster broadcasts a TV program using a specific channel and a viewer tunes into this channel to view the broadcast. There is no direct connection between the broadcaster and the viewer.

In MQTT a publisher publishes messages on a topic and a subscriber must subscribe to that topic to view the message. MQTT requires the use of a central Broker.

2.3.1 IMPORTANT POINTS TO NOTE

1. Clients do not have addresses like in email systems, and messages are not sent to clients.
2. Messages are published to a broker on a topic.
3. The job of an MQTT broker is to filter messages based on topic, and then distribute them to subscribers.
4. A client can receive these messages by subscribing to that topic on the same broker
5. There is no direct connection between a publisher and subscriber.
6. All clients can publish (broadcast) and subscribe (receive).
7. MQTT brokers do not normally store messages.

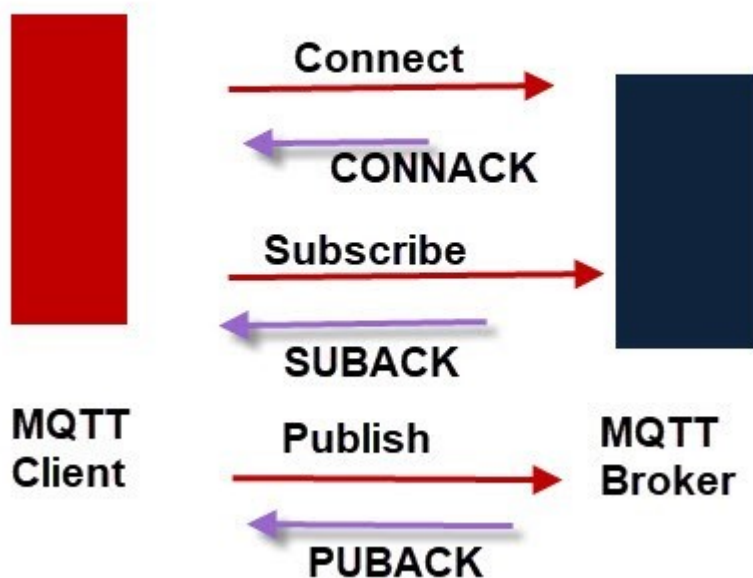
2.3.2 MQTT CLIENT BROKER CONNECTIONS

MQTT uses TCP/IP to connect to the broker. TCP is a connection orientated protocol with error correction and guarantees that packets are received in order. You can consider a TCP/IP connection to be similar to a telephone connection. Once a telephone connection is established you can talk over it until one party hangs up.

Most MQTT clients will connect to the broker and remain connected even if they aren't sending data. Connections are acknowledged by the broker using a Connection acknowledgement message. You cannot publish or subscribe unless you are connected.

MQTT clients publish a keepalive message at regular intervals (usually 60 seconds) which tells the broker that the client is still connected.

MQTT Message Flow



2 Fig. 2.2 MQTT message flow

2.3.3 THE CLIENT NAME / ID

All clients are required to have a client name or ID. The client name is used by the MQTT broker to track subscriptions etc. Client names must also be unique. If we attempt to connect to an MQTT broker with the same name as an existing client then the existing client connection is dropped. Because most MQTT clients will attempt to reconnect following a disconnect this can result in a loop of disconnect and connect.

2.3.4 CLEAN SESSIONS

MQTT clients by default establish a clean session with a broker. A clean session is one in which the broker isn't expected to remember anything about the client when it disconnects. With a non clean **session** the broker will remember client subscriptions and may hold undelivered messages for the client. However this depends on the Quality of service used when subscribing to topics, and the quality of service used when publishing to those topics.

2.3.5 LAST WILL MESSAGES

The idea of the last will message is to notify a subscriber that the publisher is unavailable due to network outage. The last will message is set by the publishing client, and is set on a per topic basis which means that each topic can have its own last will message. This means that each topic can have its own last will message associated with it. The message is stored on the broker and sent to any subscribing client (to that topic) if the connection to the publisher fails. If the publisher disconnects normally the last Will Message is not sent. The actual will messages is including with the connect request message.

The table below provides some examples of the applications of MQTT:

<i>Applications</i>	<i>Description</i>
<i>Home Automation</i>	Used to control smart home devices such as lighting, heating, and security systems
<i>Industrial Automation</i>	Used in industrial automation to monitor and control machinery and equipment
<i>Healthcare</i>	Used to monitor patients and medical equipment in hospitals and clinics
<i>Smart Agriculture</i>	Used to monitor environmental conditions in fields, track soil moisture levels, and control irrigation systems
<i>Transportation</i>	Used to monitor and track vehicles, track goods and packages, and provide real-time information to passengers
<i>Energy Management</i>	Used to monitor and control energy usage in homes, buildings, and factories

Table 0.2. Applications of MQTT

Overall, MQTT is a versatile and widely used protocol in the IoT space, providing a reliable and efficient means of data communication between devices and applications.

Chapter 3

Methodology

3.1 PROPOSED SOLUTION

The proposed system is a solution to the challenges faced by IGCAR in monitoring and analysing various sensors at the nuclear power plant. The system is designed to provide a centralized platform for managing multiple clients (sensors) without the need for physical access or multiple accesses, thereby reducing the risk of errors. The system will also provide real-time monitoring of server activities and performance to enable administrators to promptly detect and address any issues.

The development methodology involves several stages such as:

1. **Sensor Data Acquisition:** The first step would be to acquire sensor data from various nuclear power plant systems, such as temperature, pressure, radiation levels, and flow rates. The sensor data can be collected by sensor nodes connected to various devices, such as PLCs, sensors, and transmitters.
2. **MQTT Integration:** The sensor data is to be integrated with an MQTT broker, which acts as a messaging hub to receive, route, and distribute the sensor data to various MQTT subscribers, such as dashboards, web applications, and mobile apps.
3. **MQTT Topics and Payload:** MQTT topics are created for each sensor data type, such as "temperature," "pressure," and "radiation." The sensor data is then transmitted as JSON payloads, which include the sensor value, timestamp, and sensor ID.
4. **Subscriber Applications:** MQTT subscribers can be developed to receive and visualize the sensor data in real time. Subscribers can be web applications, dashboards, or mobile apps, which can display sensor data in the form of graphs, charts, or tables.
5. **Data Analytics:** The sensor data can be analysed using machine learning and

data analytics techniques to predict equipment failures, detect anomalies, and optimize system performance.

6. Alerts and Notifications: The MQTT system can also generate alerts and notifications when sensor data falls outside predefined thresholds or when equipment failures are detected.

This proposed solution can enable IGCAR to monitor various sensors in real time, detect anomalies, and optimize system performance in nuclear power plants. MQTT can provide a reliable, efficient, and secure communication channel for exchanging sensor data between various devices and applications. The solution can improve the safety and reliability of nuclear power plant operations and reduce the risk of equipment failures and accidents.

3.2 SYSTEM ARCHITECTURE AND DESIGN

The proposed client-server system for IoT applications in nuclear power plants is designed to collect and monitor various sensor data such as temperature, pressure, radiation levels, and flow rates using MQTT protocol. The system architecture comprises three main components: sensors, MQTT broker, and client application.

The sensors are responsible for collecting data from various locations in the nuclear power plant and sending it to the MQTT broker. The MQTT broker acts as a middleware that receives data from the sensors and sends it to the client application for further processing.

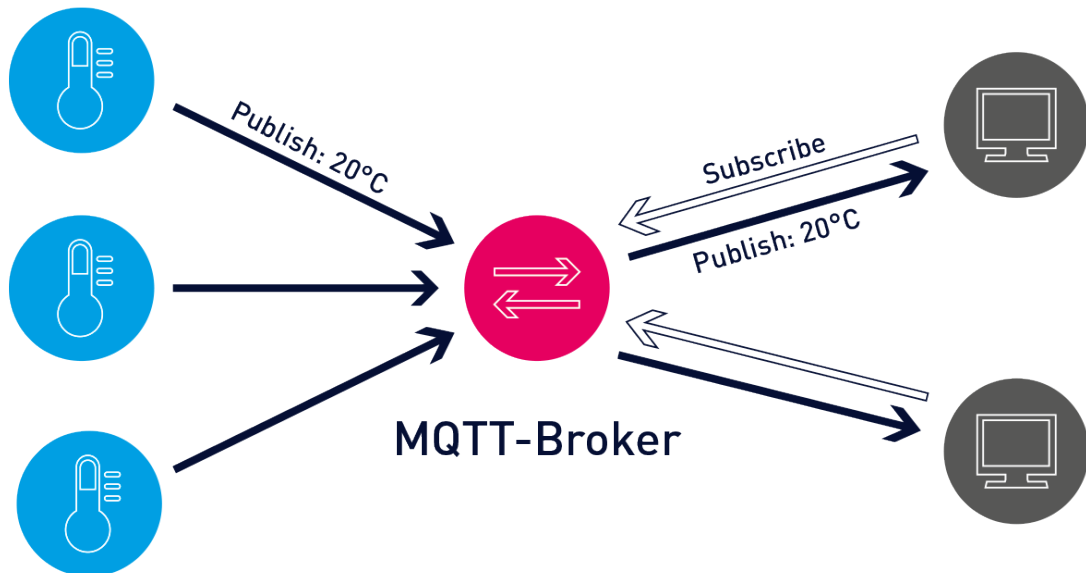
The client application is designed to visualize the sensor data in a user-friendly graphical user interface (GUI) and provide real-time monitoring and analysis capabilities. The GUI is designed using HTML, CSS, and JavaScript and allows the user to view various sensor data in the form of tables, graphs, and charts.

The following table summarizes the system architecture and design:

<i>Component</i>	<i>Description</i>
<i>Sensors</i>	Collect sensor data and send it to the MQTT broker
<i>MQTT Broker</i>	Acts as a middleware to receive sensor data and send it to the client application
<i>Client Application</i>	Visualize sensor data in a user-friendly GUI using HTML, CSS, and JavaScript

Table 0.3. System Architecture & Design

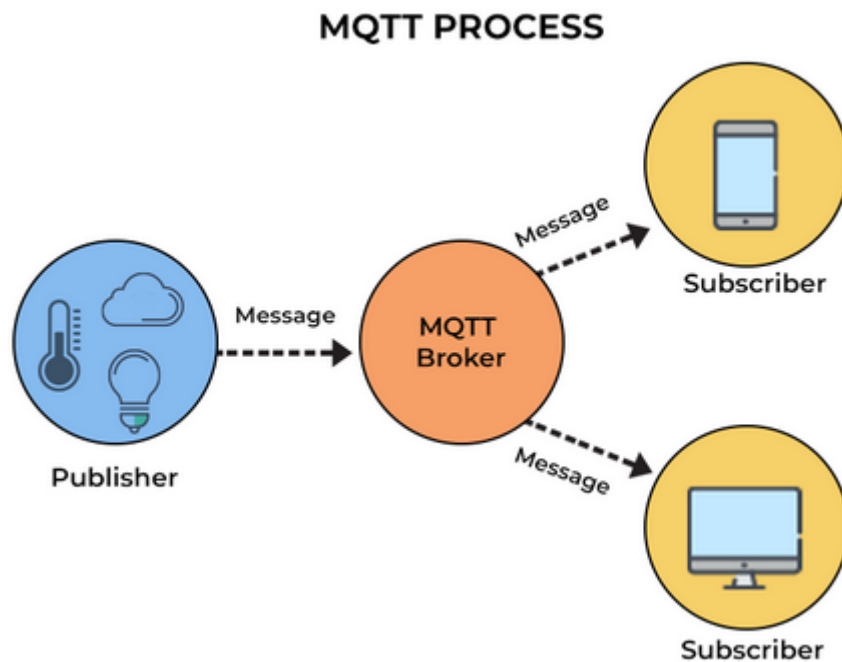
The proposed system architecture is designed to provide a reliable and scalable solution for monitoring and analyzing sensor data in nuclear power plants using MQTT protocol.



3Fig. 3.1 MQTT Broker Architecture

3.3 IMPLEMENTATION & TESTING

The implementation of the MQTT protocol-based client-server system for IoT applications in nuclear power plants involves several stages, including designing and developing the system architecture, configuring the MQTT broker and clients, integrating the system with various sensors, and developing the user interface for data visualization and analysis.



4Fig. 3.2 MQTT Process

For implementation and testing we have used various design approach and materials such as Arduino mega, Ethernet shield, CAT 5 cable, Arduino IDE, Mosquitto broker, hivemq broker, TCP communication, HK Telnet server, Qt Creator for client GUI application in Ubuntu Linux.

3.4 DESIGN APPROACH AND DETAILS

3.4.1 ARDUINO MEGA:

The Arduino Mega is a microcontroller board that is part of the Arduino family. It is based on the ATmega2560 microcontroller chip, which has 256KB of flash memory, 8KB of SRAM, and 4KB of EEPROM. The board has a total of 54 digital input/output (I/O) pins, 16 analog inputs, and 4 hardware UARTs (serial communication ports).

One of the key features of the Arduino Mega is its large number of I/O pins, which makes it ideal for projects that require the control of many different sensors, actuators, and other devices. The 54 digital I/O pins are divided into 15 pins that can be used as PWM outputs, and 6 pins that can be used as analog outputs. The 16 analog inputs provide a range of 0 to 5 volts and are ideal for reading values from sensors such as light sensors, temperature sensors, and potentiometers.

In addition to its high number of I/O pins, the Arduino Mega also has a larger memory size compared to other Arduino boards. This allows for the development of more complex projects that require more advanced programming and data storage. The board is compatible with a wide range of software libraries and programming languages, including the Arduino IDE, which makes it easy for beginners to get started with programming.

The Arduino Mega is widely used in a variety of applications, including robotics, automation, and data logging. Its large number of I/O pins and memory size makes it ideal for controlling multiple devices and collecting data from multiple sensors simultaneously. It is also a popular choice for educational purposes, as it allows for the development of more advanced projects that require a higher level of programming knowledge and skill.

Overall, the Arduino Mega is a powerful microcontroller board that is designed for more complex projects that require a larger number of input and output pins. Its high memory sizes and compatibility with a wide range of software libraries make it a popular choice among hobbyists, students, and professionals alike.

3.4.2 ETHERNET SHIELD:

The Ethernet Shield Rev2 is an add-on module for Arduino boards that allows them to connect to the internet through a wired Ethernet connection. It is based on the W5500 Ethernet chip, which supports up to four simultaneous socket connections and provides a high level of stability and reliability.

The Ethernet Shield Rev2 has a standard Ethernet interface, which means that it can be easily connected to any Ethernet network with a standard RJ45 connector. It also has an onboard micro-SD card slot, which can be used to store data such as web pages, sensor data, and other information.

The shield uses the Ethernet library for Arduino, which makes it easy to connect to the internet and communicate with other devices on the network. The library provides a range of functions for sending and receiving data, as well as configuring network settings such as IP address, subnet mask, and gateway address.

One of the key features of the Ethernet Shield Rev2 is its compatibility with a wide range of Arduino boards, including the Uno, Mega, and Due. This makes it easy for users to add internet connectivity to their existing Arduino projects, without the need for additional hardware or complex wiring.

The Ethernet Shield Rev2 is widely used in a variety of applications, including home automation, data logging, and remote monitoring. It can be used to control devices such as lights, motors, and sensors, and to collect data from a variety of sources such as temperature sensors, humidity sensors, and air quality sensors.

3.4.3 QT CREATOR

QT is a cross-platform software development framework that is used to create desktop, mobile, and embedded applications. It is written in C++ and provides a range of tools and libraries for building applications that run on a variety of platforms, including Windows, macOS, Linux, Android, and iOS.

One of the key features of QT is its ability to provide a consistent user interface across multiple platforms. This is achieved through the use of QT's widget toolkit, which provides a range of graphical user interface (GUI) components such as buttons,

text boxes, and sliders. These components are designed to look and behave the same way across all supported platforms, which makes it easy for developers to create applications that work seamlessly on different devices and operating systems.

QT also provides a range of libraries and tools for developing applications that require network connectivity, multimedia support, and 3D graphics. The network library provides support for protocols such as HTTP, FTP, and SMTP, while the multimedia library provides support for playing and recording audio and video. The 3D graphics library allows developers to create advanced 3D graphics and animations.

One of the key advantages of QT is its ability to be used with a range of programming languages, including C++, Python, and JavaScript. This allows developers to choose the language that best suits their needs and expertise, while still being able to take advantage of the full range of QT's features and capabilities.

QT is widely used in a variety of industries, including automotive, aerospace, and medical devices. Its cross-platform compatibility, combined with its powerful features and tools, make it a popular choice for developing complex applications that require a high level of performance, reliability, and functionality.

Overall, QT is a powerful and versatile software development framework that provides a range of tools and libraries for building applications that run on multiple platforms. Its ability to provide a consistent user interface, along with its support for a wide range of programming languages and industries, make it a popular choice for developers looking to create advanced applications that require a high level of performance and functionality.

3.4.4 VISUAL BASIC CODE:

Visual Basic (VB) code is a programming language developed by Microsoft for building Windows-based applications. It is an event-driven language that allows developers to create graphical user interfaces (GUIs) for their applications quickly and easily. VB code is based on the BASIC programming language, which was developed in the 1960s and 1970s and is known for its simplicity and ease of use.

One of the key features of VB code is its ability to use drag-and-drop tools to create user interfaces. This allows developers to create interfaces quickly and easily without having to write code manually. VB code also provides a range of pre-built controls and components, such as buttons, text boxes, and menus, which can be easily added to the interface.

VB code is widely used for developing a range of applications, including desktop applications, web applications, and games. It is also commonly used in the development of Microsoft Office applications, such as Excel, Word, and Access.

VB code is relatively easy to learn and is suitable for beginners who are new to programming. However, it is also powerful enough to create complex applications that require advanced functionality and features. VB code is commonly used by businesses and organizations to create custom applications that meet their specific needs.

Overall, VB code is a powerful and versatile programming language that allows developers to create user interfaces quickly and easily. Its ease of use and pre-built components make it a popular choice for beginners, while its power and flexibility make it a suitable choice for advanced developers who require advanced functionality and features.

3.5 CONSTRAINTS, ALTERNATIVES & TRADEOFFS

Working on this project for the Indira Gandhi Centre for Atomic Research which deals with sensitive data that requires a high level of security and privacy. There were many constraints in the methods and procedures we could utilize for our project, and those are:

3.5.1 USING ONLY C/C++

C/C++ are often used over Python or Java in security-sensitive industries due to their lower level of abstraction and greater control over the hardware. This means that C/C++ can be used to write code that directly interacts with the computer's hardware, such as memory management and system calls. This control is important in security-sensitive industries because it allows developers to write code that is optimized for

performance and security, without relying on higher-level abstractions that may introduce security vulnerabilities.

Another reason C/C++ is often preferred in security-sensitive industries is that it allows for more fine-grained control over memory management. In C/C++, developers must manually allocate and deallocate memory, which can help prevent memory leaks and buffer overflow vulnerabilities. On the other hand, Python and Java use automatic memory management, which can be less secure because it can lead to unanticipated memory usage and make it more difficult to ensure that sensitive data is not left in memory.

Finally, C/C++ has a long history of use in security-sensitive industries, and a wealth of security-specific knowledge and tools are available for these languages. This includes libraries and frameworks for cryptography, secure communication, and intrusion detection, as well as a large community of security experts who are experienced in using C/C++ to build secure systems.

While Python and Java are popular programming languages for a variety of applications, including some in security-sensitive industries, C/C++ remains the go-to choice for many security-sensitive applications where performance, control, and security are critical factors.

3.5.2 USAGE OF ONLY LINUX BASED SYSTEMS

Linux is a free and open-source operating system based on the Unix operating system. It was first developed by Linus Torvalds in 1991 as a hobby project while studying computer science at the University of Helsinki. Since then, it has become one of the most widely used operating systems in the world, powering everything from servers and supercomputers to mobile phones and embedded devices.

Linux is known for its stability, security, and flexibility. It allows users to customize every aspect of their system, from the graphical interface to the kernel itself. It also has a large and active community of developers and users who contribute to its development and support.

There are many different distributions of Linux, each with its own set of features and tools. Some popular distributions include Ubuntu, Debian, Fedora, and CentOS.

Linux is used in safety-critical applications for various reasons. Firstly, it is an open-source operating system, which means it can be freely modified and distributed, making it highly customizable. Secondly, Linux is a highly reliable and stable operating system, with built-in features for fault tolerance, high availability, and redundancy. Additionally, Linux has a strong security track record and is well-known for its ability to handle large workloads and high traffic volumes. All of these features make it a popular choice for safety-critical applications, such as those found in the nuclear power industry, where reliability, security, and availability are of utmost importance.

3.5.3 SAFETY CRITICAL APPLICATIONS

Safety-critical applications are systems or processes where failure can cause harm to people, damage to equipment, or environmental impact. These applications require a higher degree of safety, reliability, and accuracy than normal applications. Examples of safety-critical applications include nuclear power plants, medical devices, transportation systems, aerospace systems, and military applications. These systems must be designed, implemented, and maintained to the highest standards to ensure the safety of the public and the environment.

IGCAR deals with sensitive data that requires high security and privacy. As a result, we were limited with the resources provided to us because of the safety rules.

Limited resources:

Security-sensitive industries may operate under tight budgets, which can limit the resources available for investing in security measures. This may require careful prioritization of security initiatives and an emphasis on cost-effective solutions that offer the most protection for the least investment.

Complex IT infrastructure:

Many security-sensitive industries operate complex IT infrastructure that may include a mix of legacy systems, cloud-based services, and on-premises hardware. This can create challenges in maintaining consistent security controls and ensuring that data is properly secured across all systems and devices.

Chapter 4

Results and Analysis

4.1 DATA COLLECTION AND TRANSMISSION

Nuclear power plants use a variety of sensors to monitor different aspects of plant operation, safety, and security. Some of the sensors used in nuclear power plants are:

1. **Radiation detectors:** These sensors detect the presence of ionizing radiation and measure its intensity. They are used to monitor radiation levels in various parts of the plant, including the reactor and the containment building.
2. **Temperature sensors:** These sensors are used to monitor the temperature of the coolant, fuel rods, and other critical components in the reactor. They help ensure that the reactor is operating within safe temperature limits.
3. **Pressure sensors:** These sensors are used to monitor the pressure of the coolant, steam, and other gases in the reactor and the associated piping and equipment. They help maintain safe operating pressures and prevent accidents.
4. **Level sensors:** These sensors are used to monitor the level of liquid in various parts of the plant, including the reactor and the steam generators. They help ensure that there is sufficient coolant to remove heat from the reactor and prevent damage to the fuel rods.
5. **Flow sensors:** These sensors are used to monitor the flow of coolant, steam, and other gases in the plant. They help maintain safe flow rates and prevent blockages and other issues that can lead to accidents.
6. **Vibration sensors:** These sensors are used to monitor the vibration of rotating machinery, such as pumps and turbines. They help detect potential

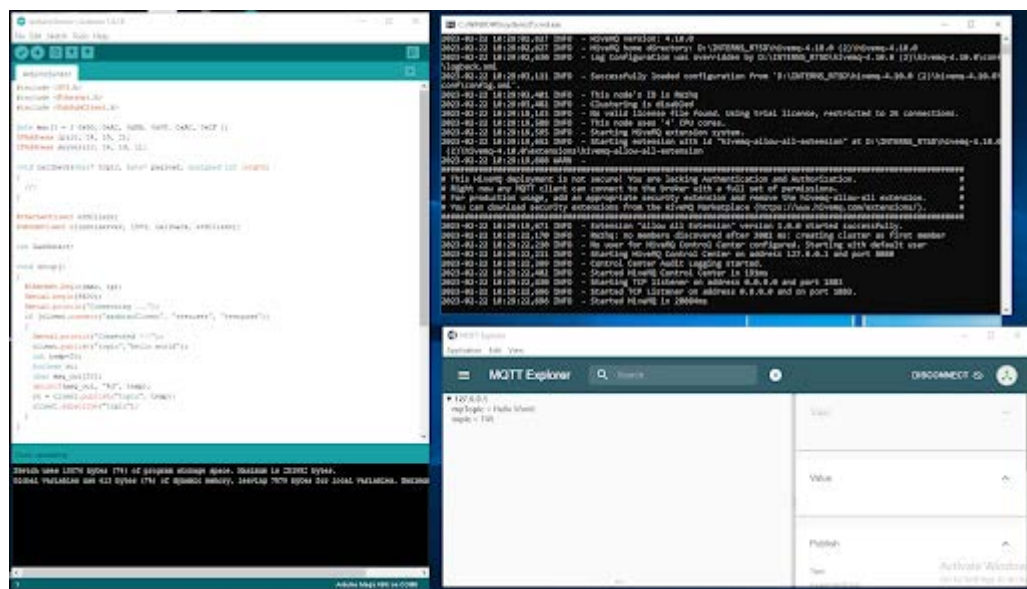
failures and prevent accidents.

7. Infrared sensors: These sensors are used to monitor the temperature of equipment and components in the plant. They help detect overheating and other issues that can lead to failures and accidents.

Overall, the use of sensors in nuclear power plants is critical to ensuring safe and reliable operation of these facilities.

4.2 SERVER-SIDE PROCESSING AND VISUALISATION

During the initial phase of the project, we implemented an Arduino Mega-based client with an Ethernet shield integrated into it, which enabled us to establish a reliable TCP communication protocol with a server running on the host PC. Subsequently, we advanced the client to support MQTT protocol and communication, allowing for improved data transfer and management capabilities.



5Fig 4.1: Arduino code and messages published, subscribed

4.2.1 TESTING USING ARDUINO & ETHERNET SHIELD

Arduino is an open-source platform used for building electronic projects. It consists of a programmable circuit board (microcontroller) and an integrated development environment (IDE) that is used to write, upload, and debug code to the board. Arduino is designed to be user-friendly, making it easy for beginners to get started in electronics and programming. It supports a wide range of sensors, actuators, and other electronic components, allowing users to create a variety of projects, from simple blinking LEDs to complex robotics projects. Arduino has gained popularity in the maker community due to its ease of use, low cost, and versatility.

The image shows a screenshot of the Arduino IDE interface. The title bar reads "sketch_feb09a | Arduino 1.8.18". The menu bar includes "File", "Edit", "Sketch", "Tools", and "Help". Below the menu bar is a toolbar with icons for opening files, saving, uploading, and downloading. The main text area contains the following C++ code:

```
sketch_feb09a
#include <SPI.h>
#include <Ethernet.h>
#include <PubSubClient.h>

byte mac[] = { 0x90, 0xA2, 0xDA, 0x0F, 0xAC, 0xCF };
IPAddress ip(10, 24, 19, 2);
IPAddress server(10, 24, 19, 1);

void callback(char* topic, byte* payload, unsigned int length)
{
  //;
}

EthernetClient ethClient;
PubSubClient client(server, 1883, callback, ethClient);

void setup()
{
  Ethernet.begin(mac, ip);
  Serial.begin(9600);
  Serial.println("Connecting ...");
  if (client.connect("arduinoClient", "testuser", "testpass"))
  {
    Serial.println("Connected !!!");
    client.publish("topic", "hello world");
    client.subscribe("topic");
  }
}

void loop()
{
  client.loop();
}
```

The status bar at the bottom indicates "1" on the left and "Arduino Mega ADK on COM8" on the right.

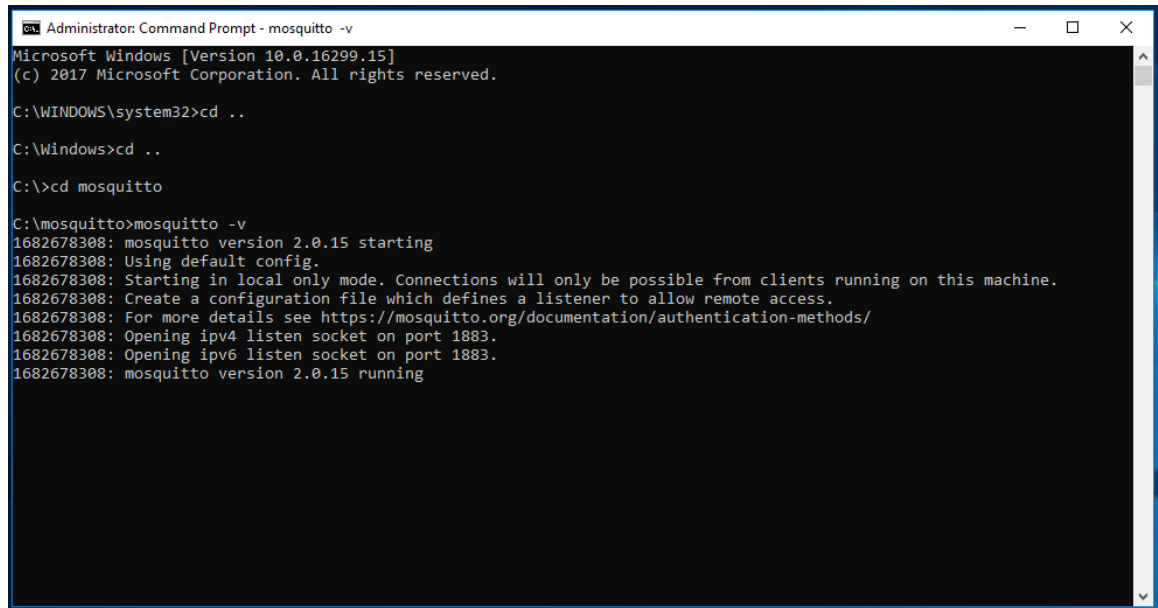
6Fig. 4.2 Arduino as a MQTT Server



7Fig. 4.3 Arduino as a MQTT Client

4.2.2 TESTING USING MOSQUITTO SERVER AND CLIENT

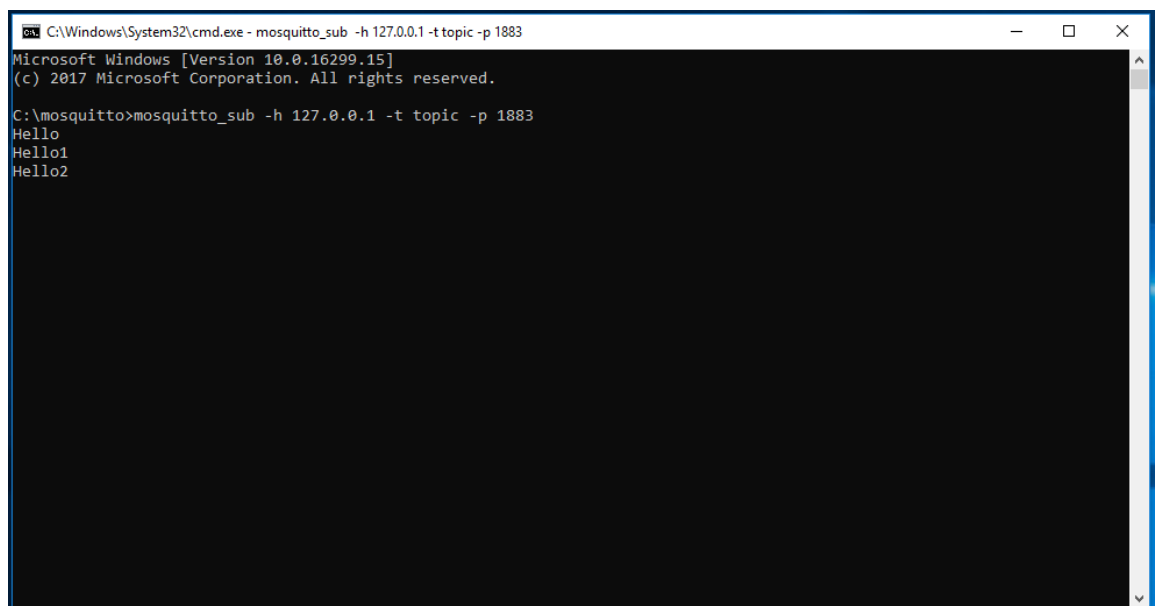
Eclipse Mosquitto is an open-source message broker that implements the MQTT (Message Queuing Telemetry Transport) protocol. It is written in C and can be run on various platforms, including Linux, Windows, and macOS. Mosquitto provides a lightweight and efficient way to handle messaging between devices in an IoT (Internet of Things) network. It allows for easy integration of different devices and sensors and provides reliable communication even in unreliable network conditions. Mosquitto also supports various authentication and security mechanisms, making it a secure choice for IoT applications. Overall, Mosquitto is a widely used and robust MQTT broker that has proven to be a reliable choice for IoT developers.



```
Administrator: Command Prompt - mosquitto -v
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd ..
C:\Windows>cd ..
C:\>cd mosquitto
C:\mosquitto>mosquitto -v
1682678308: mosquitto version 2.0.15 starting
1682678308: Using default config.
1682678308: Starting in local only mode. Connections will only be possible from clients running on this machine.
1682678308: Create a configuration file which defines a listener to allow remote access.
1682678308: For more details see https://mosquitto.org/documentation/authentication-methods/
1682678308: Opening ipv4 listen socket on port 1883.
1682678308: Opening ipv6 listen socket on port 1883.
1682678308: mosquitto version 2.0.15 running
```

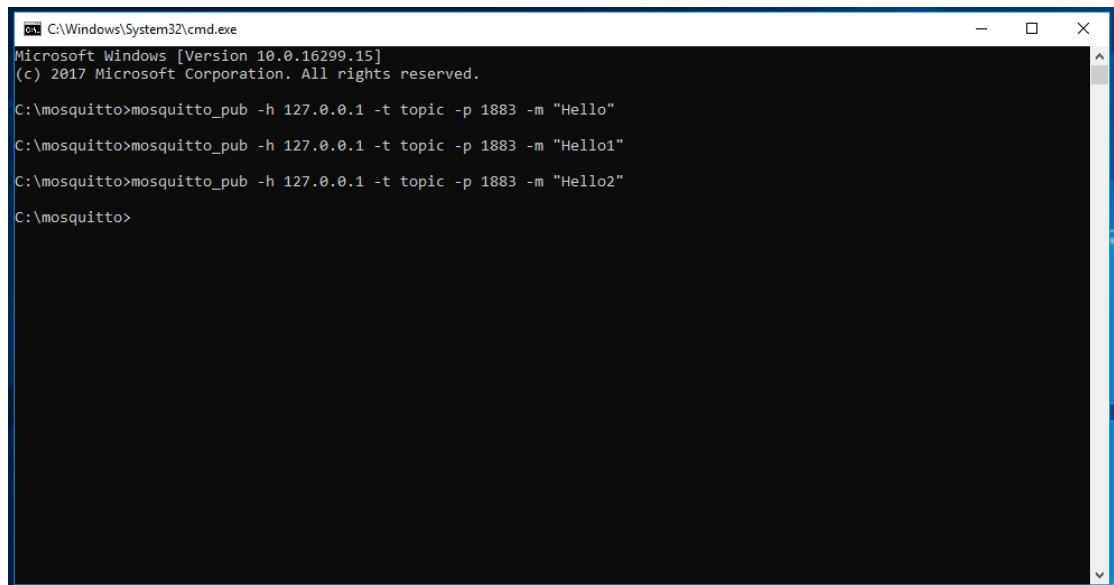
8Fig. 4.4 Starting mosquito server



```
C:\Windows\System32\cmd.exe - mosquitto_sub -h 127.0.0.1 -t topic -p 1883
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\mosquitto>mosquitto_sub -h 127.0.0.1 -t topic -p 1883
Hello
Hello1
Hello2
```

9Fig. 4.5 Mosquito subscribe client



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\mosquitto>mosquitto_pub -h 127.0.0.1 -t topic -p 1883 -m "Hello"
C:\mosquitto>mosquitto_pub -h 127.0.0.1 -t topic -p 1883 -m "Hello1"
C:\mosquitto>mosquitto_pub -h 127.0.0.1 -t topic -p 1883 -m "Hello2"
C:\mosquitto>
```

10Fig. 4.6 Mosquito publish client

4.2.3 TESTING USING HIVEMQ BROKER

HiveMQ is a popular MQTT (Message Queue Telemetry Transport) broker, which is designed for handling large-scale IoT deployments. It provides a reliable and scalable platform for the transmission of real-time data between devices, servers, and applications.



```
C:\WINDOWS\System32\cmd.exe

-----
HiveMQ
-----

HiveMQ Start Script for Windows v1.6

Running with admin rights.

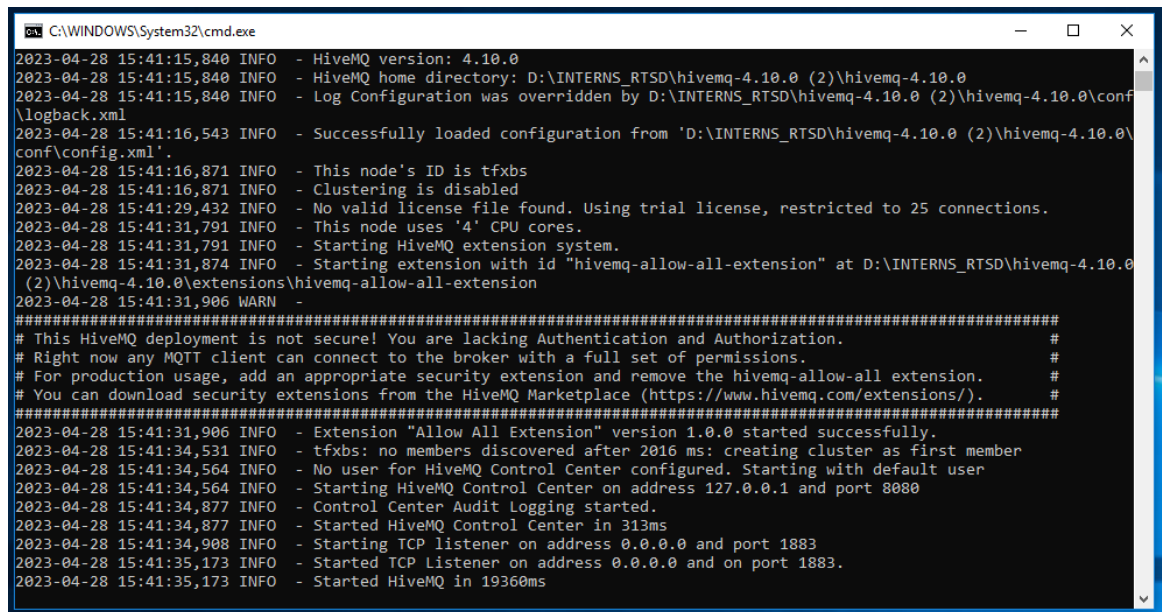
-----

HIVEMQ_HOME: "D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0"

JAVA_OPTS: -XX:+HeapDumpOnOutOfMemoryError -XX:HeapDumpPath="D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0\heap-dump.hprof" -Duser.language=en -Duser.region=US -Dcom.sun.management.jmxremote -Dcom.sun.management.jmxremote.port=9010 -Dcom.sun.management.jmxremote.local.only=false -Dcom.sun.management.jmxremote.authenticate=false -Dcom.sun.management.jmxremote.ssl=false -Djava.net.preferIPv4Stack=true -noverify --add-opens java.base/java.lang=ALL-UNNAMED --add-opens java.base/java.nio=ALL-UNNAMED --add-opens java.base/sun.nio.ch=ALL-UNNAMED --add-opens java.base/sun.security.provider=ALL-UNNAMED --add-opens jdk.management/com.sun.management.internal=ALL-UNNAMED --add-exports java.base/jdk.internal.misc=ALL-UNNAMED
```

11Fig. 4.7 Starting HiveMQ Server

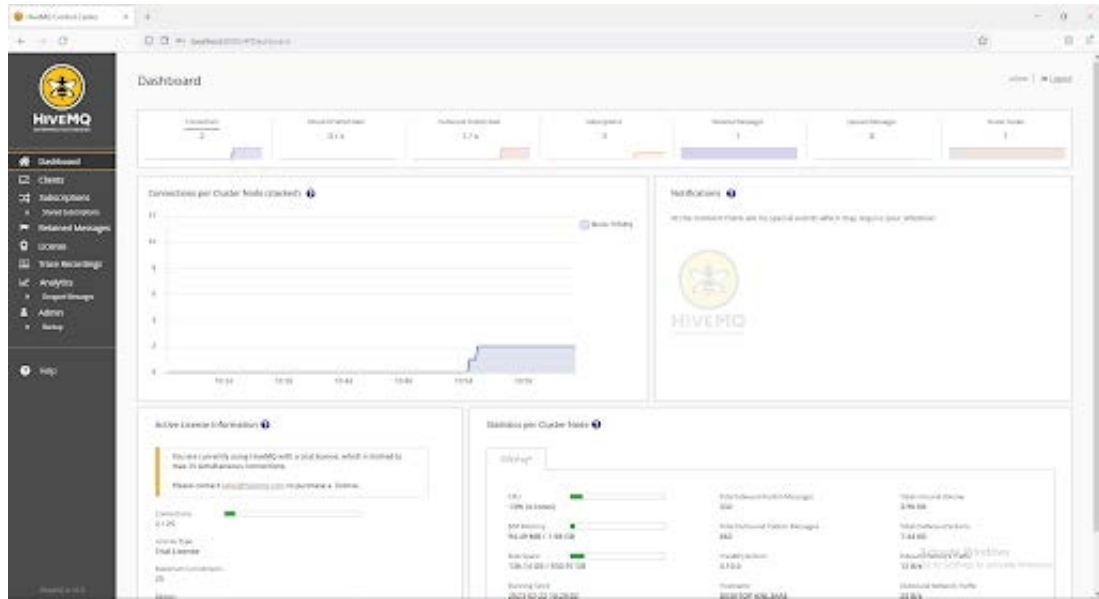
HiveMQ supports various versions of the MQTT protocol, including MQTT v3.1, v3.1.1, and v5.0, and it also provides a range of features such as clustering, security, and integration with other systems. It supports integration with cloud platforms like AWS, Azure, and Google Cloud.



```
C:\WINDOWS\System32\cmd.exe
2023-04-28 15:41:15,840 INFO - HiveMQ version: 4.10.0
2023-04-28 15:41:15,840 INFO - HiveMQ home directory: D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0
2023-04-28 15:41:15,840 INFO - Log Configuration was overridden by D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0\conf\logback.xml
2023-04-28 15:41:16,543 INFO - Successfully loaded configuration from 'D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0\conf\config.xml'.
2023-04-28 15:41:16,871 INFO - This node's ID is tfxbs
2023-04-28 15:41:16,871 INFO - Clustering is disabled
2023-04-28 15:41:29,432 INFO - No valid license file found. Using trial license, restricted to 25 connections.
2023-04-28 15:41:31,791 INFO - This node uses '4' CPU cores.
2023-04-28 15:41:31,791 INFO - Starting HiveMQ extension system.
2023-04-28 15:41:31,874 INFO - Starting extension with id "hivemq-allow-all-extension" at D:\INTERNS_RTSD\hivemq-4.10.0 (2)\hivemq-4.10.0\extensions\hivemq-allow-all-extension
2023-04-28 15:41:31,906 WARN - 
#####
# This HiveMQ deployment is not secure! You are lacking Authentication and Authorization. #
# Right now any MQTT client can connect to the broker with a full set of permissions. #
# For production usage, add an appropriate security extension and remove the hivemq-allow-all extension. #
# You can download security extensions from the HiveMQ Marketplace (https://www.hivemq.com/extensions/). #
#####
2023-04-28 15:41:31,906 INFO - Extension "Allow All Extension" version 1.0.0 started successfully.
2023-04-28 15:41:34,531 INFO - tfxbs: no members discovered after 2016 ms: creating cluster as first member
2023-04-28 15:41:34,564 INFO - No user for HiveMQ Control Center configured. Starting with default user
2023-04-28 15:41:34,564 INFO - Starting HiveMQ Control Center on address 127.0.0.1 and port 8080
2023-04-28 15:41:34,877 INFO - Control Center Audit Logging started.
2023-04-28 15:41:34,877 INFO - Started HiveMQ Control Center in 313ms
2023-04-28 15:41:34,908 INFO - Starting TCP listener on address 0.0.0.0 and port 1883
2023-04-28 15:41:35,173 INFO - Started TCP Listener on address 0.0.0.0 and on port 1883.
2023-04-28 15:41:35,173 INFO - Started HiveMQ in 19360ms
```

12Fig. 4.8 Started HiveMQ Broker

HiveMQ can handle a large number of concurrent connections, making it suitable for applications that require real-time data transmission and low-latency communication. It also provides features like message persistence, message queuing, and message routing, which ensure that data is delivered reliably and securely.



13Fig. 4.9 HiveMQ Dashboard

Overall, HiveMQ is a robust and feature-rich MQTT broker that is widely used in industrial IoT applications, including the nuclear power industry, where reliable and secure data transmission is critical.

During the subsequent phase of our project, we undertook the design and development of a highly robust and versatile MQTT client application, utilizing the versatile QT cross-platform software in the CentOS Linux operating system. With the implementation of the MQTT client, we were able to establish a highly secure and reliable communication channel that effectively transferred data between the client and an already established open-source broker system.

The MQTT client was designed with a highly intuitive and user-friendly interface, which facilitated ease of use and seamless integration with a wide range of systems and applications. The use of QT software provided us with the flexibility to integrate a broad range of advanced features, such as automatic re-connection, support for multiple clients and brokers, and optimized data compression, which significantly enhanced the overall efficiency and reliability of our system.

Overall, the implementation of the MQTT client utilizing QT software represents a highly effective and efficient solution for secure data transfer and management in complex and highly demanding environments, such as those found in security-sensitive industries.

The screenshot shows the Qt Creator IDE with the project 'simplemqttclient [6.2.4]' open. The file 'mainwindow.cpp' is selected in the Projects pane. The code editor displays the implementation of the 'MainWindow::MainWindow' constructor. The code includes necessary headers, initializes the QtMqttClient, sets up the UI, and connects signals to slots for handling MQTT messages.

```

1 //*****
50
51 #include "mainwindow.h"
52 #include "ui_mainwindow.h"
53
54 #include <QtCore/QDateTime>
55 #include <QtMqtt/QtMqttClient>
56 #include <QtWidgets/QMessageBox>
57
58 MainWindow::MainWindow(QWidget *parent) :
59     QMainWindow(parent),
60     ui(new Ui::MainWindow)
61 {
62     ui->setupUi(this);
63
64     m_client = new QtMqttClient(this);
65     m_client->setHostname(ui->lineEditHost->text());
66     m_client->setPort(ui->spinBoxPort->value());
67
68     connect(m_client, &QtMqttClient::stateChanged, this, &MainWindow::updateLogStateChange);
69     connect(m_client, &QtMqttClient::disconnected, this, &MainWindow::brokerDisconnected);
70
71     connect(m_client, &QtMqttClient::messageReceived, this, [this](const QByteArray &message,
72         const QString content = QDateTime::currentDateTime().toString()
73         + QLatin1String(" Received Topic: ")
74         + topic.name()
75         + QLatin1String(" Message: ")
76         + message

```

14Fig. 4.10: QT code 1

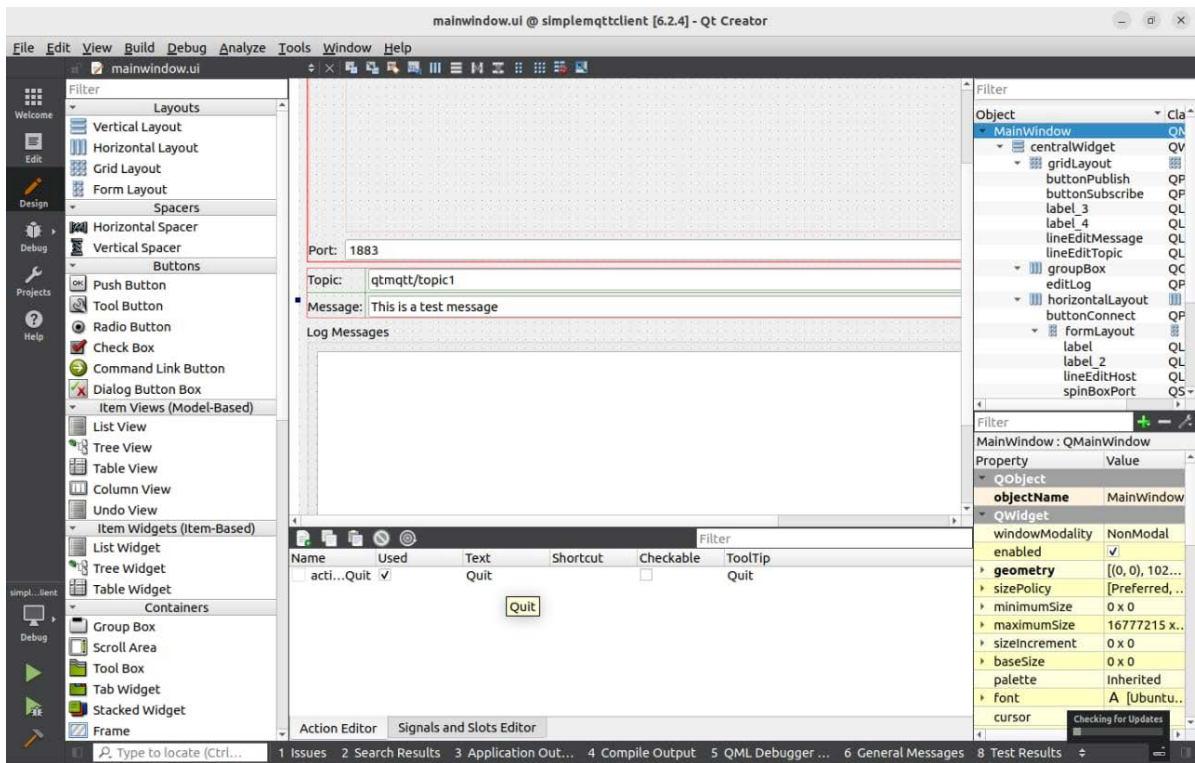
The screenshot shows the Qt Creator IDE with the project 'simplemqttclient [6.2.4]' open. The file 'main.cpp' is selected in the Projects pane. The code editor displays the implementation of the 'main' function, which creates a QApplication, instantiates the MainWindow, and calls exec() to start the application.

```

1 //*****
50
51 #include "mainwindow.h"
52 #include <QApplication>
53
54 int main(int argc, char *argv[])
55 {
56     QApplication a(argc, argv);
57     MainWindow w;
58     w.show();
59
60     return a.exec();
61 }
62

```

15Fig. 4.11: QT code 2



16Fig. 4.12: UI design of the client

In the ultimate phase of the project, we designed and developed an MQTT broker, which we subsequently connected to the previously created client as well as the Arduino client. The MQTT broker enabled the transfer of data between the various clients, and its implementation ensures secure and reliable communication channel establishment, allowing for accurate data transmission between connected devices.

By deploying this MQTT broker in conjunction with the Arduino client and the previously created client, we successfully established a network of interconnected devices capable of transmitting and receiving data reliably and securely. This approach to data acquisition is expected to have a significant impact on the continuous monitoring of power plant equipment condition and predictive maintenance in the industry.



17Fig. 4.13: Dashboard



18Fig. 4.14: Dashboard of the Broker after giving 5 rounds of publish/subscribe

Finally, the Evaluation of the system's performance is done in terms of data collection and transmission of various sensor data like radiation, temperature, pressure, level, flow, vibration, and infrared. It also includes the Analysis of the system's reliability, security, and scalability. We have compared the system's performance with existing solutions for data collection and transmission in nuclear power plants and Identified potential areas for improvement and future work.

Overall, the results and analysis of an MQTT-based client-server system has demonstrated the effectiveness of the system in collecting and transmitting sensor data in real-time, providing a more efficient and reliable solution for monitoring nuclear power plants. It will also provide valuable insights for further optimization and enhancement of the system in the future.

Chapter 5

Conclusion & Future Work

In conclusion, the development of an MQTT protocol-based client-server system for IoT applications in nuclear power plants has been successfully accomplished. The system architecture and design were developed, and the client-server communication was established using the MQTT protocol. The system has been tested and evaluated for its performance and reliability.

MQTT protocol has proved to be a reliable and efficient method for collecting and transmitting data from various sensors used in nuclear power plants. The low bandwidth usage, small message sizes, and publish/subscribe model of MQTT make it suitable for IoT applications in nuclear power plants where real-time data analysis and decision making are critical.

The use of an MQTT client-server system and a graphical user interface allows for seamless data collection, storage, and visualization. However, it is essential to consider security measures such as authentication and encryption to protect the data being transmitted. Future work can involve the integration of machine learning algorithms for predictive maintenance and anomaly detection in nuclear power plants.

The use of Qt Creator provided a powerful and user-friendly interface for creating custom GUI elements, and the integrated support for the MQTT protocol streamlined the development process.

By utilizing this technology, we can create robust and responsive GUI applications that can interface with a wide variety of MQTT-enabled IoT devices. The combination of MQTT and Qt Creator allows for a flexible and scalable solution that can be easily adapted to a wide range of applications and industries.

The development of a MQTT client-based GUI application using Qt Creator in Ubuntu Linux has the potential to greatly enhance the efficiency and effectiveness of IoT systems, and is a promising area of future development and research.

The results of the testing showed that the developed system can efficiently collect data from various sensors in real-time and transmit it to the server for analysis. The system can also be extended to include additional sensors and features for more comprehensive monitoring and control of nuclear power plants.

Future work can include the integration of advanced machine learning algorithms for predictive maintenance and improved system performance. Additionally, the development of a mobile application to remotely monitor and control the system can also be explored. Overall, the developed system can provide a reliable and efficient solution for monitoring and controlling nuclear power plants, ensuring the safety and reliability of these critical infrastructures.

REFERENCES

- A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash. (2015) "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in *IEEE Communications Surveys & Tutorials*, vol. 17, no. 4, pp. 2347-2376, Fourth quarter.
- Santhosh Kumar, July, (2019) "A Review on Client-Server Based Applications And Research Opportunity" in the *International Journal of Recent Scientific Research* Vol. 10, Issue, 07(H), pp. 33857-33862.
- M. Carratú, V. Gallo and V. Paciello, (2022) "IEEE 1451: Communication among smart sensors using MQTT protocol," *2022 IEEE International Symposium on Measurements & Networking (M&N)*, Padua, Italy, 2022, pp. 1-6.
- M. Z. Ahmad, A. R. Adenan, M. S. Rohmad and Y. M. Yussoff, (2023) "Performance Analysis of Secure MQTT Communication Protocol," *2023 19th IEEE International Colloquium on Signal Processing & Its Applications (CSPA)*, Kedah, Malaysia, pp. 225-229.
- A. Sahadevan, D. Mathew, J. Mookathana and B. A. Jose, (2017) "An Offline Online Strategy for IoT Using MQTT," *2017 IEEE 4th International Conference on Cyber Security and Cloud Computing (CSCloud)*, New York, NY, USA, pp. 369-373.
- L. Staglianò, E. Longo and A. E. C. Redondi, (2021) "D-MQTT: design and implementation of a pub/sub broker for distributed environments," *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, Barcelona, Spain, pp. 1-6.
- Xiangtao Liu, Tianle Zhang, Ning Hu, Peng Zhang, Yu Zhang. (2020) The method of Internet of Things accesses and network communication based on MQTT,

Computer Communications, Volume 153, Pages 169-176, ISSN 0140-3664.

Pietro Colombo, Elena Ferrari, Engin Deniz Tümer, (2021) Regulating data sharing across MQTT environments, Journal of Network and Computer Applications, Volume 174, 102907, ISSN 1084-8045

Liher Granado, Oskar Berreteaga, (2015) Creating Rich Human-machine Interfaces with Rational Rhapsody and Qt for Industrial Multi-core Real-time Applications, Procedia Manufacturing, Volume 3, Pages 1903-1909, ISSN 2351-9789.

Adam Mechtley, Ryan Trowbridge, (2012), Chapter 8 - Advanced Graphical User Interfaces with Qt, Editor(s): Adam Mechtley, Ryan Trowbridge, Maya Python for Games and Film, Morgan Kaufmann, Pages 233-258, ISBN 9780123785787

Appendix A

Source code for MQTT Client Application

A.1 SIMPLE MQTT CLIENT APPLICATION

19Fig. A.1 simple MQTT Client GUI Application

This MQTT Client is build using Qt Creator Software in Ubuntu 22 system. I have used the `QMqttClient` class to create the MQTT client. The class provides properties for setting a unique client ID as well as the broker host name and port to connect to.

A.2 CODE FILES

A.2.1 CMakeLists.txt

```
cmake_minimum_required(VERSION 3.16)
project(simplemqttclient LANGUAGES CXX)

set(CMAKE_AUTOMOC ON)
```



```

set(CMAKE_AUTOUIC ON)

if(NOT DEFINED INSTALL_EXAMPLESDIR)
    set(INSTALL_EXAMPLESDIR "examples")
endif()

set(INSTALL_EXAMPLEDIR
"${INSTALL_EXAMPLESDIR}/mqtt/simpleclient")

find_package(Qt6 REQUIRED COMPONENTS Core Gui Mqtt
Network Widgets)

qt_add_executable(simplemqttclient
    main.cpp
    mainwindow.cpp mainwindow.h mainwindow.ui
)

set_target_properties(simplemqttclient PROPERTIES
    WIN32_EXECUTABLE TRUE
    MACOSX_BUNDLE TRUE
)

target_compile_definitions(simplemqttclient PUBLIC
    QT_DEPRECATED_WARNINGS
)

target_link_libraries(simplemqttclient PUBLIC
    Qt::Core
    Qt::Gui
    Qt::Mqtt
    Qt::Network
)

target_link_libraries(simplemqttclient PUBLIC
    Qt::Widgets)

if((QT_MAJOR_VERSION GREATER 4))
    target_link_libraries(simplemqttclient PUBLIC
        Qt::Widgets
    )
endif()

install(TARGETS simplemqttclient
    RUNTIME DESTINATION "${INSTALL_EXAMPLEDIR}"
    BUNDLE DESTINATION "${INSTALL_EXAMPLEDIR}"
    LIBRARY DESTINATION "${INSTALL_EXAMPLEDIR}"
)

```

A.2.2 main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

A.2.3 mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include <QtCore/QDateTime>
#include <QtMqtt/QMqttClient>
#include <QtWidgets/QMessageBox>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    m_client = new QMqttClient(this);
    m_client->setHostname(ui->lineEditHost->text());
    m_client->setPort(ui->spinBoxPort->value());

    connect(m_client, &QMqttClient::stateChanged, this,
        &MainWindow::updateLogStateChange);
    connect(m_client, &QMqttClient::disconnected, this,
        &MainWindow::brokerDisconnected);

    connect(m_client, &QMqttClient::messageReceived,
        this, [this](const QByteArray &message, const
        QMqttTopicName &topic) {
        const QString content =
        QDateTime::currentDateTime().toString()
            + QLatin1String(" Received Topic: ")
            + topic.name()
            + QLatin1String(" Message: ")
            + message
            + QLatin1Char('\n');
    });
}
```

```

        ui->editLog->insertPlainText(content);
    });

    connect(m_client, &QMqttClient::pingResponseReceived,
this, [this]() {
        const QString content =
QDateTime::currentDateTime().toString()
            + QLatin1String(" PingResponse")
            + QLatin1Char('\n');
        ui->editLog->insertPlainText(content);
    });

    connect(ui->lineEditHost, &QLineEdit::textChanged,
m_client, &QMqttClient::setHostname);
    connect(ui->spinBoxPort,
QOverload<int>::of(&QSpinBox::valueChanged), this,
&MainWindow::setClientPort);
    updateLogStateChange();
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::on_buttonConnect_clicked()
{
    if (m_client->state() == QMqttClient::Disconnected) {
        ui->lineEditHost->setEnabled(false);
        ui->spinBoxPort->setEnabled(false);
        ui->buttonConnect->setText(tr("Disconnect"));
        m_client->connectToHost();
    } else {
        ui->lineEditHost->setEnabled(true);
        ui->spinBoxPort->setEnabled(true);
        ui->buttonConnect->setText(tr("Connect"));
        m_client->disconnectFromHost();
    }
}

void MainWindow::on_buttonQuit_clicked()
{
    QApplication::quit();
}

void MainWindow::updateLogStateChange()
{
    const QString content =
QDateTime::currentDateTime().toString()
        + QLatin1String(": State Change")
        + QString::number(m_client->state())

```

```

        + QLatin1Char('\n');
    ui->editLog->insertPlainText(content);
}

void MainWindow::brokerDisconnected()
{
    ui->lineEditHost->setEnabled(true);
    ui->spinBoxPort->setEnabled(true);
    ui->buttonConnect->setText(tr("Connect"));
}

void MainWindow::setClientPort(int p)
{
    m_client->setPort(p);
}

void MainWindow::on_buttonPublish_clicked()
{
    if (m_client->publish(ui->lineEditTopic->text(), ui->lineEditMessage->text().toUtf8()) == -1)
        QMessageBox::critical(this,
            QLatin1String("Error"), QLatin1String("Could not publish message"));
}

void MainWindow::on_buttonSubscribe_clicked()
{
    auto subscription = m_client->subscribe(ui->lineEditTopic->text());
    if (!subscription) {
        QMessageBox::critical(this,
            QLatin1String("Error"), QLatin1String("Could not subscribe. Is there a valid connection?"));
        return;
    }
}

```

A.2.4 mainwindow.h

```

#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QMqttClient>

namespace Ui {
class MainWindow;
}

```

```

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

public slots:
    void setClientPort(int p);

private slots:
    void on_buttonConnect_clicked();
    void on_buttonQuit_clicked();
    void updateLogStateChange();

    void brokerDisconnected();

    void on_buttonPublish_clicked();

    void on_buttonSubscribe_clicked();

private:
    Ui::MainWindow *ui;
    QMqttClient *m_client;
};

#endif

```

A.2.5 mainwindow.ui

```

<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
    <class>MainWindow</class>
    <widget class="QMainWindow" name="MainWindow">
        <property name="geometry">
            <rect>
                <x>0</x>
                <y>0</y>
                <width>1024</width>
                <height>768</height>
            </rect>
        </property>
        <property name="windowTitle">
            <string>MainWindow</string>
        </property>
        <widget class="QWidget" name="centralWidget">
            <layout class="QVBoxLayout" name="verticalLayout">
                <item>

```

```

<layout class="QHBoxLayout" name="horizontalLayout">
<item>
  <layout class="QFormLayout" name="formLayout">
    <item row="0" column="0">
      <widget class="QLabel" name="label">
        <property name="text">
          <string>Host:</string>
        </property>
      </widget>
    </item>
    <item row="0" column="1">
      <widget class="QLineEdit" name="lineEditHost">
        <property name="text">
          <string/>
        </property>
      </widget>
    </item>
    <item row="2" column="0">
      <widget class="QLabel" name="label_2">
        <property name="text">
          <string>Port:</string>
        </property>
      </widget>
    </item>
    <item row="2" column="1">
      <widget class="QSpinBox" name="spinBoxPort">
        <property name="maximum">
          <number>99999</number>
        </property>
        <property name="value">
          <number>1883</number>
        </property>
      </widget>
    </item>
  </layout>
</item>
<item>
  <widget class="QPushButton" name="buttonConnect">
    <property name="text">
      <string>Connect</string>
    </property>
  </widget>
</item>
</layout>
</item>
<item>
  <layout class="QGridLayout" name="gridLayout">
    <item row="1" column="1">
      <widget class="QLineEdit" name="lineEditMessage">
        <property name="text">
          <string>This is a test message</string>

```

```

        </property>
    </widget>
</item>
<item row="1" column="0">
    <widget class="QLabel" name="label_4">
        <property name="text">
            <string>Message:</string>
        </property>
    </widget>
</item>
<item row="0" column="2">
    <widget class="QPushButton"
name="buttonSubscribe">
        <property name="text">
            <string>Subscribe</string>
        </property>
    </widget>
</item>
<item row="1" column="2">
    <widget class="QPushButton" name="buttonPublish">
        <property name="text">
            <string>Publish</string>
        </property>
    </widget>
</item>
<item row="0" column="1">
    <widget class="QLineEdit" name="lineEditTopic">
        <property name="text">
            <string>qtmqtt/topic1</string>
        </property>
    </widget>
</item>
<item row="0" column="0">
    <widget class="QLabel" name="label_3">
        <property name="text">
            <string>Topic:</string>
        </property>
    </widget>
</item>
</layout>
</item>
<item>
    <widget class="QGroupBox" name="groupBox">
        <property name="title">
            <string>Log Messages</string>
        </property>
        <layout class="QHBoxLayout"
name="horizontalLayout_2">
            <item>
                <widget class="QPlainTextEdit" name="editLog"/>
            </item>

```

```

        </layout>
    </widget>
</item>
<item>
    <layout class="QHBoxLayout"
name="horizontalLayout_3">
        <item>
            <spacer name="horizontalSpacer">
                <property name="orientation">
                    <enum>Qt::Horizontal</enum>
                </property>
                <property name="sizeHint" stdset="0">
                    <size>
                        <width>40</width>
                        <height>20</height>
                    </size>
                </property>
            </spacer>
        </item>
        <item>
            <widget class="QPushButton" name="buttonQuit">
                <property name="text">
                    <string>Quit</string>
                </property>
            </widget>
        </item>
    </layout>
</item>
</layout>
</widget>
<widget class="QMenuBar" name="menuBar">
    <property name="geometry">
        <rect>
            <x>0</x>
            <y>0</y>
            <width>1024</width>
            <height>20</height>
        </rect>
    </property>
    <widget class="QMenu" name="menuFile">
        <property name="title">
            <string>File</string>
        </property>
        <addaction name="actionQuit"/>
    </widget>
    <addaction name="menuFile"/>
</widget>
<widget class="QToolBar" name="mainToolBar">
    <attribute name="toolBarArea">
        <enum>TopToolBarArea</enum>
    </attribute>

```



```

    <attribute name="toolBarBreak">
        <bool>false</bool>
    </attribute>
</widget>
<widget class="QStatusBar" name="statusBar"/>
<action name="actionQuit">
    <property name="text">
        <string>Quit</string>
    </property>
</action>
</widget>
<layoutdefault spacing="6" margin="11"/>
<resources/>
<connections/>
</ui>

```

A.2.6 simpleclient.pro

```

QT      += core gui network mqtt

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets

TARGET = simplemqttclient
TEMPLATE = app

DEFINES += QT_DEPRECATED_WARNINGS

DEFINES += QT_DISABLE_DEPRECATED_UP_TO=0x060000

SOURCES += main.cpp\
            mainwindow.cpp

HEADERS  += mainwindow.h
FORMS    += mainwindow.ui

target.path = $$[QT_INSTALL_EXAMPLES]/mqtt/simpleclient
INSTALLS += target

```