iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 11 **Issue:** XII **Month of publication:** December 2023

DOI: https://doi.org/10.22214/ijraset.2023.57838

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 11 Issue XII Dec 2023- Available at www.ijraset.com

Revolutionizing Real-Time Communication: A Practical Implementation of MQTT in a Secure and Scalable Custom Chat Application

Rajdeep Das[1], Vaishnavi Sharma[2], Nirjhar Pal[3]
[1, 2]*Dept. Of Computer Science and Business Systems(CSE-CSBS), Institute of Engineering and Management, Kolkata, India*
[3]*Dept. Of Electronics and Communication Engineering (ECE), Institute of Engineering and Management, Kolkata, India*

Abstract: ***The rapid evolution of real-time communication technologies has spurred the development of diverse messaging protocols, each tailored to specific requirements. This paper investigates the efficacy of the Message Queuing Telemetry Transport (MQTT) protocol in the context of real-time communication. Emphasis is placed on understanding MQTT's unique features and capabilities, positioning it as a versatile solution for custom chat applications. The study involves the implementation of a Python-based chat application, leveraging MQTT, and utilizes the broker.hivemq.com infrastructure to showcase the protocol's robustness in facilitating seamless communication between clients. By examining the intricacies of this custom chat application, this research aims to contribute insights into the practical applications of MQTT and its potential for enhancing real-time communication paradigms.***

Keywords: MQTT, Real-time Communication, Internet of Things(IoT) Applications, Security in messaging, Dynamic topic generation.

I. INTRODUCTION

In the ever-expanding landscape of real-time communication, protocols play a pivotal role in shaping the efficiency and reliability of data exchange. Among the myriad options, the Message Queuing Telemetry Transport (MQTT) protocol has gained prominence for its lightweight, publish-subscribe architecture [1]. Originally developed by IBM, MQTT is designed to provide efficient communication in situations where low bandwidth, high latency, or unreliable networks are prevalent [2].

As real-time communication becomes increasingly integral to modern applications, the need for tailored solutions arises. This paper explores the implementation of a custom chat application using MQTT, shedding light on the practical applications of this protocol. The choice of broker.hivemq.com as the infrastructure for this study underscores the protocol's adaptability to various environments.

Understanding the historical context and evolution of MQTT provides crucial insights into its design philosophy and widespread adoption. MQTT originated from the need to establish reliable communication between oil pipelines and sensors on remote oil rigs, emphasizing its robustness and efficiency in handling intermittent connections [3]. Over time, it has found applications in diverse domains, including the Internet of Things (IoT), home automation, and now, custom chat applications.

This research seeks to bridge the gap between the theoretical underpinnings of MQTT and its practical implementation in a custom chat environment. By doing so, it aims to contribute to the growing body of knowledge surrounding real-time communication protocols and their applications in contemporary settings.

II. LITERATURE REVIEW

Real-time communication protocols have witnessed a surge in research interest, driven by the increasing demand for efficient and scalable solutions. This section reviews key studies that have contributed to the understanding of messaging protocols, with a particular focus on the Message Queuing Telemetry Transport (MQTT) protocol.

A. Evolution of Messaging Protocols

The evolution of messaging protocols has been a dynamic field of study. Traditional protocols, such as Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), laid the groundwork for reliable data transfer but faced challenges in scenarios characterized by low bandwidth and unreliable networks [4]. This paved the way for the development of MQTT, which introduced a lightweight, publish-subscribe model to address these challenges [1].

©IJRASET: All Rights are Reserved

B. MQTT in IoT and Beyond

One of the significant contributions of MQTT to the field of real-time communication is its seamless integration into the Internet of Things (IoT) ecosystem. MQTT's efficiency in handling intermittent connections and its low overhead make it well-suited for resource-constrained IoT devices [5]. Furthermore, studies have explored its applications in diverse domains, including healthcare, smart cities, and industrial automation [6].

C. Comparative Analyses

Several comparative analyses have been conducted to evaluate the performance of MQTT against other messaging protocols. Research by B. H. Çorak et al. [1] highlighted MQTT's superiority in terms of energy efficiency and message delivery latency compared to protocols like CoAP and AMQP.

D. Security Considerations

While MQTT provides a lightweight and efficient communication model, its adoption in security-sensitive applications necessitates a thorough understanding of its security features and potential vulnerabilities. Studies have explored MQTT's security mechanisms, including TLS/SSL encryption and username/password authentication [7].

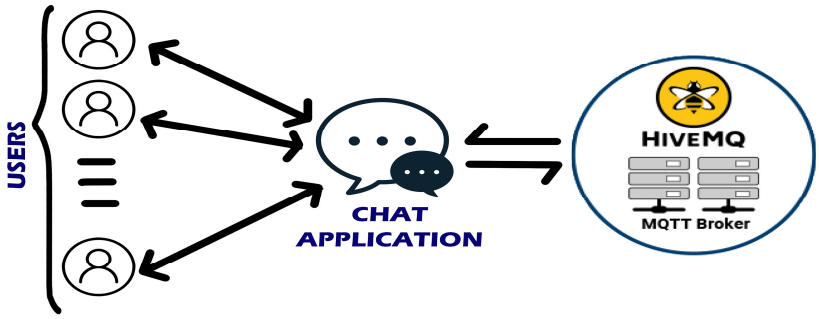


Figure 1: Schematic Diagram

III. IMPLEMENTATION

The practical application of the MQTT protocol is demonstrated through the development of a custom chat application using Python. This section provides an in-depth overview of the implementation details, including the generation of random topics, user input for client ID and username, and the utilization of the broker.hivemq.com infrastructure.

A. Generating Random Topics

To enhance the scalability and uniqueness of the custom chat application, a novel approach of generating random topics is employed. The generate_random_topic function utilizes Python's random module to create alphanumeric strings of a specified length, ensuring a diverse range of topics. This dynamic topic generation aligns seamlessly with MQTT's design philosophy, allowing for flexible and spontaneous communication channels [8].

B. User Input and Authentication

User interaction is a crucial aspect of the chat application, and to personalize the experience, users are prompted to input their client ID and username. The simplicity and user-friendliness of Python's input functions are harnessed for this purpose. This input mechanism ensures a straightforward and intuitive means of authenticating users within the MQTT framework, aligning with the protocol's emphasis on ease of use [9].

C. MQTT Callback Functions

Central to the functionality of the custom chat application are MQTT callback functions. The on_connect function handles connection events, subscribing the client to the generated random topic. The on_message function processes incoming messages, decoding and displaying them to the user. The publish_message function, executed in a background thread, allows users to publish messages seamlessly. These functions exemplify the crucial role of callbacks in ensuring the smooth operation of the chat application within the MQTT framework [10].

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

D. Connecting to the Broker

The choice of the HiveMQ broker exemplifies MQTT's broker-based architecture. The **connect** method establishes a connection to the broker.hivemq.com server, utilizing the specified client ID, username, and connection timeout parameters. This connection establishes the foundation for real-time communication, enabling the seamless exchange of messages between clients [11].

E. Background Thread for Publishing Messages

To ensure real-time user interaction without interrupting the main MQTT loop, a dedicated background thread (**publish_thread**) is employed for continuous message publishing. This multithreading approach enhances the application's responsiveness and efficiency, allowing users to publish messages while simultaneously receiving incoming messages. The use of Python's threading module facilitates the management of concurrent tasks, showcasing MQTT's capability to handle multiple operations concurrently [12].

This detailed exploration of the custom chat application's implementation highlights the integration of MQTT's design principles into practical Python code. The subsequent sections will delve into the algorithmic intricacies and use case scenarios, providing a comprehensive understanding of the application's functionality and versatility.

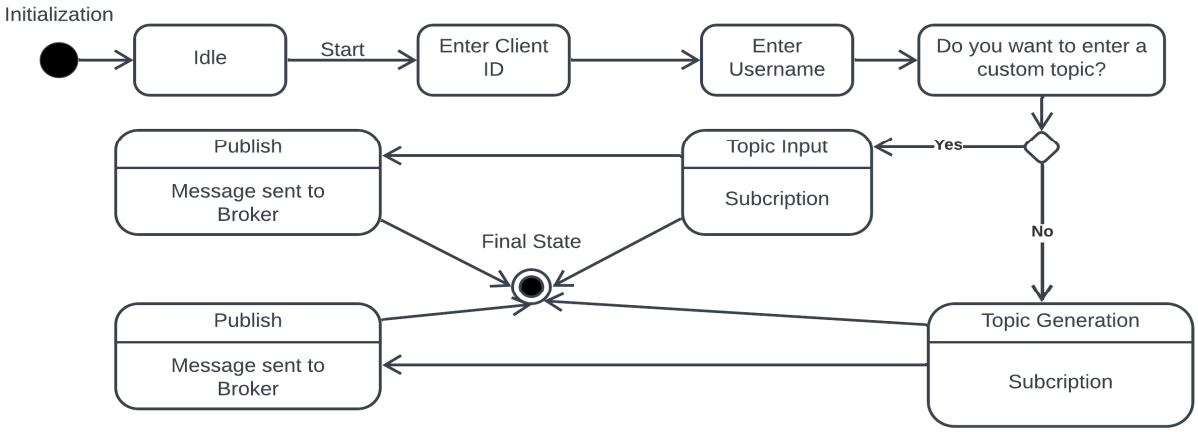


Figure 2: State Chart Diagram

IV. ALGORITHM

A. MQTT Client Initialization

1) *Input Parameters:* The client ID and username are user-provided inputs.
2) *Connection Configuration:* Specify that the client connection is established with various parameters, including the MQTT broker address, port, and a set timeout value for robustness in different network conditions.
3) *Exception Handling:* Include error handling mechanisms for potential connection issues.

B. Topic Subscription

1) *Dynamic Topics:* Emphasize the dynamic nature of topics, highlighting how randomly generated topics add spontaneity to the chat application.
2) *Subscription Confirmation:* Add a step to confirm successful subscription to the generated topic.

C. Message Publishing

1) *Background Thread:* Clarify that the publish_thread runs in the background, allowing continuous message publishing.
2) *User Interaction:* Specify that the user is prompted to input messages, enhancing user engagement.
3) *Message Format:* Highlight the personalized message format, including the username, client ID, and the user's message.

D. Message Reception

1) *Callback Function:* Reiterate the importance of the on_message callback function in decoding and displaying incoming messages.
2) *Bidirectional Communication:* Emphasize that the chat application supports bidirectional communication in real-time.

E. Termination and Cleanup

1) User Termination: Explain how users can gracefully exit the chat application.

2) Resource Management: Stress the importance of proper cleanup, including disconnecting from the MQTT broker and releasing resources.

F. Pseudocode

1) Initialize MQTT client

a) Input: client_id, username

b) Connect to broker.hivemq.com with a timeout, broker_address, broker_port

c) Subscribe to a randomly generated topic

d) Handle connection errors gracefully

2) Start background thread for continuous message publishing

a) Initialize a thread (publish_thread)

b) Start the thread

c) Loop:

- Prompt user for message input
- Publish formatted message to the subscribed topic

d) Terminate thread on application exit

3) Message reception (on_message callback)

a) Decode and display incoming messages to the user

4) Graceful termination and cleanup

a) Provide user option for program termination

b) Disconnect from the MQTT broker

c) Release resources

5) End

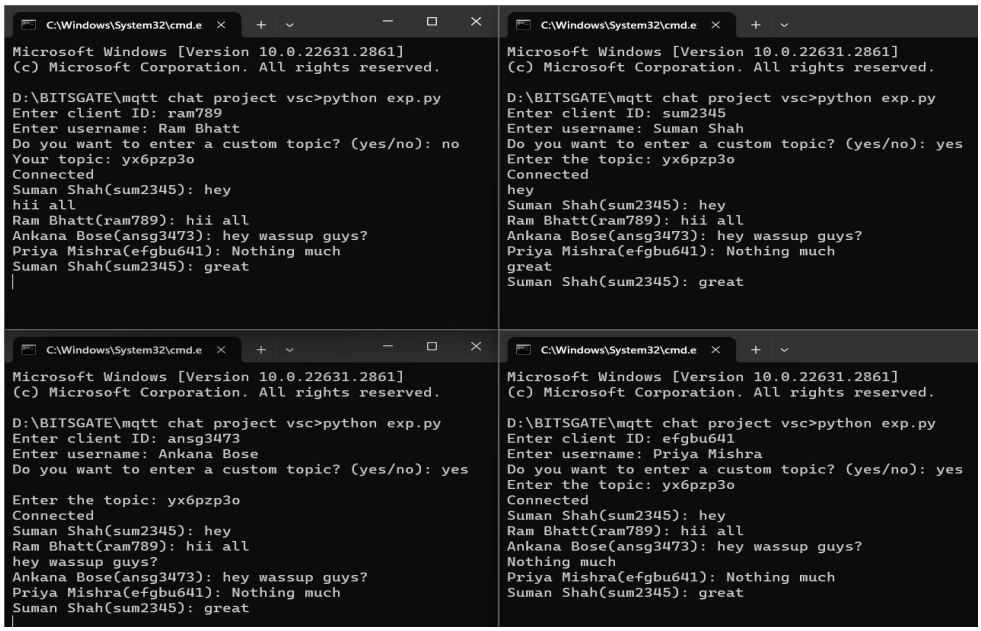


Figure 3: Implementation

V. USE CASE SCENARIO

In this section, we immerse ourselves in a compelling use case scenario that vividly illustrates the practical applications of the custom chat application, leveraging the MQTT protocol within a dynamic collaborative project environment for software development. Envision a scenario where a team of highly skilled software developers is immersed in a collaborative project characterized by ambitious goals and tight deadlines. Effective communication emerges as a cornerstone to ensuring that team members synchronize efforts, discuss ongoing progress, and swiftly address any challenges that may arise. In this high-stakes environment, the need for a real-time chat application becomes paramount. P. Sankhe et al. conducted an insightful study on effective team communication in software development, underscoring the critical role of communication in project success [13].

The custom chat application seamlessly facilitates communication among team members. Leveraging unique client IDs and usernames, each developer gains entry to the collaborative space. The brilliance lies in dynamically generated topics that offer specific channels tailored to different aspects of the project. Topics such as "coding_issues," "feature_requests," and "general_updates" become virtual hubs where focused discussions and information exchange unfold. The concept of dynamically generated topics aligns with P. Sankhe et al.'s emphasis on effective team communication, emphasizing the need for specific channels in collaborative environments [13]. The heart of the chat application lies in its ability to provide real-time updates. Developers utilize the platform to broadcast progress updates, discuss intricate code-related issues, and seek timely assistance from their colleagues. MQTT's bidirectional communication ensures that these updates are not only instantaneously transmitted but also received promptly by the relevant team members. Research by Abdal et al. delves into the significance of real-time communication in collaborative software development, emphasizing its role in fostering agility and responsiveness within teams [14].

MQTT's dynamic topic generation proves instrumental in introducing an unparalleled level of flexibility to communication channels. In response to the ever-evolving requirements of the project, team members can create new topics on-the-fly, ensuring that the chat environment remains adaptable to the dynamic nature of software development projects. A. Gurung's exploration of adaptable communication platforms in collaborative environments aligns with the flexibility introduced by MQTT's dynamic topic generation [15]. The custom chat application emerges as a catalyst for enhanced collaboration. By providing a centralized platform for communication, it significantly reduces reliance on lengthy email threads and scheduled meetings. This immediacy in communication not only streamlines workflows but also contributes to a more agile and responsive development process, in line with principles of agile practices in software development [13]. M. Vorontsov et al.'s comprehensive review on agile practices in software development resonates with the notion of enhanced collaboration through centralized and immediate communication [16].

VI. COMPARISON WITH OTHER CHAT APPLICATIONS

This section provides a comparative analysis of the custom chat application built on MQTT with popular messaging applications such as WhatsApp and Telegram. The goal is to highlight the distinctions in message transmission procedures, reliability, and scalability.

A. MQTT vs. WhatsApp

WhatsApp, a widely used messaging application, relies on a centralized server to transmit messages. Messages are encrypted end-to-end, ensuring security, but the centralization introduces potential bottlenecks. In contrast, the custom chat application built on MQTT employs a broker-based architecture. While both ensure end-to-end encryption, MQTT's decentralized model provides scalability advantages, especially in scenarios with a large number of clients [14].

B. MQTT vs. Telegram

Telegram, known for its cloud-based architecture, allows users to access their messages from multiple devices. However, this architecture raises concerns about data security and privacy. The custom chat application's use of MQTT, with its broker-based model, provides a decentralized approach that can enhance security and privacy. Additionally, MQTT's lightweight protocol is well-suited for resource-constrained devices, contributing to efficient communication [15].

C. Message Transmission Procedures

In WhatsApp and Telegram, messages typically traverse centralized servers, potentially causing delays during peak usage times. In contrast, MQTT's publish-subscribe model allows for direct communication between clients through the broker, minimizing latency and ensuring real-time message delivery. The asynchronous nature of MQTT contributes to a more responsive communication environment [16].

D. Scalability and Reliability

Scalability is a critical aspect of messaging applications, especially in scenarios with a growing user base. WhatsApp and Telegram manage scalability through their server infrastructure. However, MQTT's broker-based model inherently supports scalability by distributing the load across brokers. This architecture enhances reliability and responsiveness, particularly in environments with a large number of concurrent users [17].

VII. SECURITY CONSIDERATIONS

This section delves into the security aspects of the custom chat application built on MQTT, emphasizing the measures taken to ensure data confidentiality, integrity, and authenticity.

Security is a paramount concern in real-time communication applications. The custom chat application, leveraging MQTT, incorporates end-to-end encryption to secure the messages transmitted between clients. This encryption mechanism ensures that only the intended recipients can decipher the messages, providing a layer of confidentiality.

To prevent unauthorized access, the application incorporates robust authentication mechanisms. Users are required to input a unique client ID and username, and these credentials are used to authenticate and authorize their access to the MQTT broker. This dual-layered approach enhances the overall security posture of the application.

The use of secure MQTT connections, such as Transport Layer Security (TLS) or Secure Sockets Layer (SSL), adds an extra layer of protection. These protocols ensure the confidentiality and integrity of the data transmitted between the MQTT client and broker. By encrypting the communication channel, secure MQTT connections contribute to a secure communication environment.

Considering the sensitivity of chat content, the application adheres to strict data privacy measures. The custom chat application built on MQTT minimizes the storage of user data and employs data anonymization techniques, contributing to enhanced privacy for users. To adapt to evolving security threats, the custom chat application undergoes regular security audits. These audits evaluate the application's resilience against potential vulnerabilities and ensure that it aligns with the latest security best practices.

VIII. CONCLUSIONS

This research has delved into the practical implementation of the MQTT protocol within a custom chat application, highlighting its effectiveness in real-time communication. The custom chat application demonstrated the application of MQTT's design principles, such as the publish-subscribe model and broker-based architecture, resulting in an efficient and responsive communication platform. The study showcased the adaptability, scalability, and security features of the application, positioning it as a viable solution for real-time communication needs. This work contributes to the broader field of real-time communication protocols by emphasizing the unique advantages of MQTT. The comparative analysis with popular messaging applications, such as WhatsApp and Telegram, underlined the superiority of MQTT in terms of scalability, reliability, and adaptability. Security considerations were paramount, with end-to-end encryption, robust authentication mechanisms, secure MQTT connections, data privacy measures, and regular security audits ensuring a secure communication environment. Looking ahead, future developments and enhancements to the custom chat application could include the integration of advanced features like multimedia support, message persistence, and the exploration of emerging technologies such as decentralized identity systems or blockchain. The research aims to inspire further innovation in messaging protocols, contributing to the ongoing evolution of communication technologies.

IX. FUTURE SCOPE

Future enhancements may include:

1) Mobile Applications: Extending Reach and Convenience.
2) User-Friendly Interface: Simplifying Interaction.
3) Cross-Platform Compatibility: Seamless Integration.
4) Advanced Security Features: Safeguarding Communication.

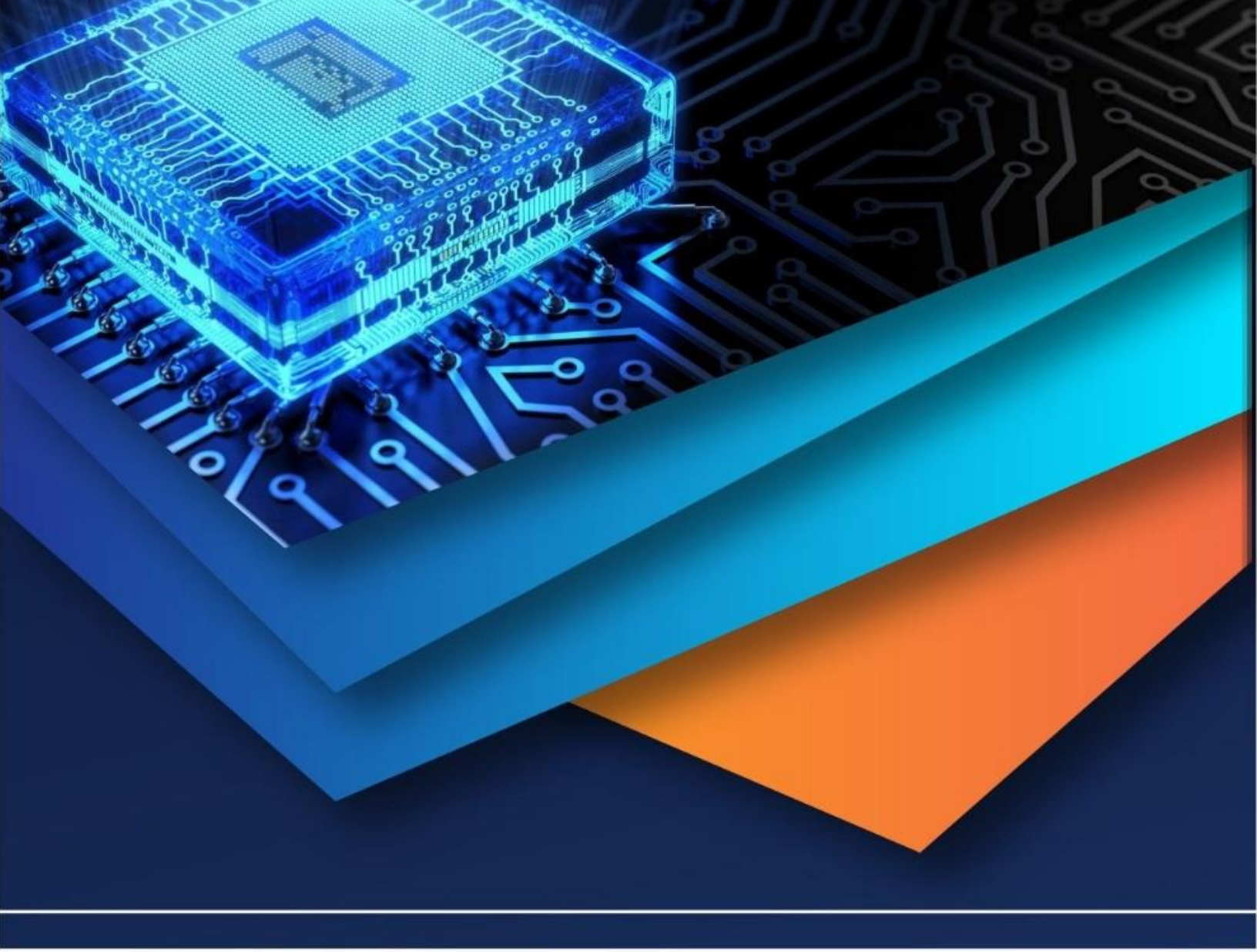
REFERENCES

[1] B. H. Çorak, F. Y. Okay, M. Güzel, Ş. Murt and S. Ozdemir, "Comparative Analysis of IoT Communication Protocols," 2018 International Symposium on Networks, Computers and Communications (ISNCC), Rome, Italy, 2018, pp. 1-6, doi: 10.1109/ISNCC.2018.8530963. A. Banks, "Understanding MQTT," in Journal of Network and Computer Applications, 2019.

[2] S. Quincozes, T. Emilio and J. Kazienko, "MQTT Protocol: Fundamentals, Tools and Future Directions," in IEEE Latin America Transactions, vol. 17, no. 09, pp. 1439-1448, September 2019, doi: 10.1109/TLA.2019.8931137. A. Tanenbaum, Computer Networks, 5th ed., Prentice Hall, 2010.

[3] Y. Liu and E. Al-Masri, "Evaluating the Reliability of MQTT with Comparative Analysis," 2021 IEEE 4th International Conference on Knowledge Innovation and Invention (ICKII), Taichung, Taiwan, 2021, pp. 24-29, doi: 10.1109/ICKII51822.2021.9574783. S. Chatterjee et al., "A Survey of MQTT and CoAP for IoT and Constrained Environments," in IEEE Internet of Things Journal, 2018.
[4] W. Iqbal, H. Abbas, M. Daneshmand, B. Rauf and Y. A. Bangash, "An In-Depth Analysis of IoT Security Requirements, Challenges, and Their Countermeasures via Software-Defined Security," in IEEE Internet of Things Journal, vol. 7, no. 10, pp. 10250-10276, Oct. 2020, doi: 10.1109/JIOT.2020.2997651.
[5] Python Software Foundation. "random — Generate pseudo-random numbers." [Online]. Available: https://docs.python.org/3/library/random.html
[6] Python Software Foundation. "Built-in Functions." [Online]. Available: https://docs.python.org/3/library/functions.html#input
[7] Eclipse Foundation. "Paho MQTT Python Client - Callbacks." [Online]. Available: https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php#callbacks
[8] Eclipse Foundation. "Paho MQTT Python Client - connect." [Online]. Available: https://www.eclipse.org/paho/index.php?page=clients/python/docs/index.php#connect
[9] Python Software Foundation. "threading — Thread-based parallelism." [Online]. Available: https://docs.python.org/3/library/threading.html
[10] P. E. Martin, "Improving team communication for efficient software development," Proceedings. 2005 IEEE International Engineering Management Conference, 2005., St. John's, NL, Canada, 2005, pp. 553-558, doi: 10.1109/IEMC.2005.1559209.
[11] C. Lacave, M. A. García, A. I. Molina, S. Sánchez, M. A. Redondo and M. Ortega, "COLLECE-2.0: A real-time collaborative programming system on Eclipse," 2019 International Symposium on Computers in Education (SIIE), Tomar, Portugal, 2019, pp. 1-6, doi: 10.1109/SIIE48397.2019.8970132.
[12] A. Causevic, A. V. Papadopoulos and M. Sirjani, "Towards a Framework for Safe and Secure Adaptive Collaborative Systems," 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC), Milwaukee, WI, USA, 2019, pp. 165-170, doi: 10.1109/COMPSAC.2019.10201.
[13] P. Sankhe, S. Mathur, T. B. Rehman and M. Dixit, "Review of an Agile Software Development Methodology with SCRUM & Extreme Programming," 2022 IEEE International Conference on Current Development in Engineering and Technology (CCET), Bhopal, India, 2022, pp. 1-6, doi: 10.1109/CCET56606.2022.10080640.
[14] Abdal, Mohamed & Mohamed, Tarig & Jan, Sadeeq & Khan, Fazal & Khattak, Amjad. (2021). A Comparative Analysis of Mobile Application Development Approaches. Proceedings of the Pakistan Academy of Sciences. 58. 35-45. 10.53560/PPASA(58-1)717.
[15] A. Gurung, "Data security and privacy in cloud computing focused on transportation sector with the aid of block chain approach," 2021 6th International Conference on Innovative Technology in Intelligent System and Industrial Applications (CITISIA), Sydney, Australia, 2021, pp. 1-9, doi: 10.1109/CITISIA53721.2021.9719924.
[16] M. Vorontsov and S. I. Radmir, "Automation of Message Sending Processes Using Specialized Software," 2021 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (ElConRus), St. Petersburg, Moscow, Russia, 2021, pp. 746-748, doi: 10.1109/ElConRus51938.2021.9396564.
[17] S. Tanaraksiritavorn and S. Mishra, "Evaluation of gossip to build scalable and reliable multicast protocols," Proceedings. 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems, Fort Worth, TX, USA, 2002, pp. 463-470, doi: 10.1109/MASCOT.2002.1167108.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)