

PUBLISH/SUBSCRIBE METHOD FOR DATA
PROCESSING IN MASSIVE IOT LEVERAGING
BLOCKCHAIN FOR SECURED STORAGE

by

Mohammad Hossein ATA EI KACHOU EI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLEMENT FOR A MASTER'S DEGREE
WITH THESIS IN ELECTRICAL ENGINEERING
M.A.Sc.

MONTREAL, JULY 02, 2024

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC



Mohammad Hossein Ataei, 2024



This Creative Commons licence allows readers to download this work and share it with others as long as the author is credited. The content of this work can't be modified in any way or used commercially.

BOARD OF EXAMINERS (THESIS M.Sc.A.)

THIS THESIS HAS BEEN EVALUATED
BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Rene Jr. Landry, Thesis Supervisor
Department of Electrical Engineering at École de technologie supérieure

Mr. Michel Kadoch, President of the Board of Examiners
Department of Electrical Engineering at École de technologie supérieure

Mrs. Manel Abdellatif, Member of the jury
Department of Software Engineering and IT at École de technologie supérieure

Mr. Guy Chevrette, External Evaluator
iMETRIK Global Inc.

THIS THESIS WAS PRESENTED AND DEFENDED
IN THE PRESENCE OF A BOARD OF EXAMINERS AND PUBLIC
ON JUNE 28, 2024
AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Professor Rene Jr. Landry, for his support, guidance, and invaluable insights throughout the entire process of conducting this research and writing this thesis. Their mentorship has been instrumental in shaping the direction of my work and refining my academic endeavors.

I extend my heartfelt thanks to the industrial partner of the project, Mr. Guy Chevrete at iMETRIK Global Inc for his encouragement and constructive feedback. Their expertise has been an asset in enhancing the quality of my research.

I am grateful to my teammates in LASSENA laboratory, Mr. Ali Eghmazi, Mr. Ali Shakerian, Mr. Mohammadhadi Foroutan, and Ms. Yasmine Bahria, for their understanding, patience, and encouragement during the demanding phases of thesis preparation. Their support has been a constant source of motivation.

Finally, special appreciation goes to my family members, who have consistently served as a wellspring of motivation throughout my master's program.

Méthode de publication/abonnement pour le traitement en temps réel des données dans l'internet des objets massif en tirant parti de la blockchain pour le stockage sécurisé

Mohammad Hossein ATAEI KACHOUEI

RÉSUMÉ

À l'ère de l'Internet des objets (IdO), l'essor des appareils de type machine (MTD) a entraîné l'avènement de l'Internet des objets massif (MIoT) ou de la communication massive de type machine (mMTC), créant de nouvelles possibilités pour les appareils connectés. Cependant, le traitement en temps réel des flux de données substantiels et divers pose des défis. Pour y remédier, nous utilisons une architecture de messagerie Publish/Subscribe, spécialement conçue pour les capacités de traitement des données en temps réel. Notre méthode implique un système de publication/abonnement, où les données des appareils sont envoyées à un courtier dédié agissant en tant que serveur distinct. Ce courtier permet à toute application de s'abonner à l'ensemble de données, favorisant un environnement de données dynamique et réactif. Au cœur de notre configuration de transmission de données se trouve une plateforme de stockage et de traitement de flux distribuée (Apache Kafka version 3.6.1), reconnue pour sa gestion efficace des flux de données. Kafka, qui tire parti de l'architecture Publish/Subscribe, facilite la connexion entre les capteurs MIoT et les courtiers, en prenant en charge des clusters parallèles pour une évolutivité améliorée. En plus de notre moteur de diffusion d'événements en temps réel, nous utilisons des logiciels de base de données analytiques tels que Apache Druid version 28.0.0, connus pour leurs capacités d'analyse en temps réel. D'autre part, pour garantir une connectivité ininterrompue, nous intégrons un mécanisme de secours à l'aide de deux radios définies par logiciel (SDR) appelées Nutaq PicoLTE Release 1.5 dans notre modèle. Cette redondance améliore la disponibilité de la transmission de données, protégeant contre les interruptions de connectivité. Pour la sécurité du référentiel de données, nous tirons parti de la technologie de la chaîne de blocs, notamment du système de fichiers interplanétaire et de Hyperledger Fabric, connus pour leurs caractéristiques de haute performance, garantissant l'intégrité, l'immutabilité et la sécurité des données. Nos résultats de latence montrent que notre plateforme réduit considérablement la latence pour 100 000 appareils classés comme MIoT à moins de 25 millisecondes. De plus, notre recherche sur les performances de la chaîne de blocs met en évidence notre modèle comme une plateforme sécurisée, atteignant plus de 800 transactions par seconde dans un ensemble de données de 14 000 transactions, démontrant son efficacité élevée.

Mots-clés : méthode de publication/abonnement, Internet des objets massif, traitement de données en temps réel (Apache Kafka), analyse de données (Apache Druid), scalabilité, disponibilité, sécurité

Publish/Subscribe method for real-time data processing in massive IoT leveraging blockchain for secured storage

Mohammad Hossein ATAIE KACHOUEI

ABSTRACT

In the era of the Internet of Things (IoT), the rise of Machine-Type Devices (MTDs) has brought about Massive IoT (MIoT) or massive Machine-Type Communication (mMTC), creating new possibilities for connected devices. However, dealing with the substantial and diverse data streams in real time poses challenges. To address this, we employ Publish/Subscribe messaging architecture, specifically designed for real-time data processing capabilities. Our method involves a publish/subscribe system, where device data is sent to a dedicated broker acting as a separate server. This broker allows any application to subscribe to the dataset, fostering a dynamic and responsive data environment. At the heart of our data transmission setup is a Distributed Store and Stream-Processing Platform (Apache Kafka version 3.6.1), recognized for its effective data flow management. Kafka, which takes advantage of Publish/Subscribe architecture, facilitates the connection between MIoT sensors and brokers, supporting parallel clusters for improved scalability. On top of our real-time event streaming engine, we employ analytical database software such as Apache Druid version 28.0.0, known for its real-time analysis capabilities. On the other hand, to ensure uninterrupted connectivity, we integrate a fail-safe mechanism using two Software-Defined Radios (SDR) called Nutaq PicoLTE Release 1.5 in our model. This redundancy enhances data transmission availability, protecting against connectivity disruptions. For data repository security, we leverage blockchain technology, specifically InetrPlanetary File System and Hyperledger Fabric, known for their high-performance characteristics, ensuring data integrity, immutability, and security. Our latency results show that our platform significantly reduces latency for 100,000 devices classified as MIoT to under 25 milliseconds. Additionally, our research on blockchain performance highlights our model as a secure platform, achieving over 800 Transactions Per Second in a dataset of 14,000 transactions, demonstrating its high efficiency.

Keywords: publish/subscribe method, massive Internet of Things, real-time data processing (Apache Kafka), data analytic (Apache Druid), scalability, availability, security

TABLE OF CONTENTS

	Page
INTRODUCTION	1
0.1. Problem Description	3
0.2. Research Goals.....	6
CHAPTER 1 LITERATURE REVIEW	9
1.1. IoT Evolution.....	9
1.2. Massive IoT	10
1.3. IoT Layered Architecture.....	11
1.3.1. Request/Response Messaging Method	14
1.3.2. Publish/Subscribe Messaging Method.....	15
1.4. Related Work in MIIoT Data Processing	16
CHAPTER 2 BACKGROUND	19
2.1. Real-time Data Processing in MIIoT	19
2.1.1. Data Pipeline Tools for Data Retention & Processing.....	20
2.1.1.1. Batch vs Real-time Data Pipeline Tools	20
2.1.1.2. Open-source vs Proprietary Data Pipeline Tools.....	21
2.1.1.3. Cloud-native vs On-premise Data Pipeline Tools	21
2.1.1.4. Why Apache Kafka for Real-time Data Processing in MIIoT? ..	22
2.1.1.5. Kafka Communication Model.....	24
2.1.1.6. Kafka’s Architecture Reflecting Publish/Subscribe Feature	27
2.1.1.7. Topics & Partitions	28
2.1.1.8. Messages in Kafka	29
2.1.1.9. Batches in Kafka.....	30
2.1.1.10. Producers in Kafka.....	31
2.1.1.11. Consumers in Kafka.....	33
2.1.1.12. Brokers & Clusters in Kafka.....	34
2.1.2. Analytical Database Software (Apache Druid).....	35
2.2. Security and Privacy	36
2.3. Network Access Technologies.....	37
2.3.1. LTE as a Network Access Technology.....	39
2.3.2. NB-IoT as a Network Access Technology	39
2.3.3. 5G as a Network Access Technology	44
2.4. Communication Protocols.....	46
2.4.1. MQTT	47
2.4.2. LwM2M	48
CHAPTER 3 DESCRIPTION OF METHODOLOGY USED.....	51

3.1.	The Proposed MIoT Solution.....	51
3.1.1.	Sensors/gateway Domain.....	53
3.1.2.	Connectivity Domain.....	55
3.1.2.1.	SDR for Automatic Switchover.....	56
3.1.3.	Data Retention & Processing Domain.....	57
3.1.3.1.	Kafka for Data Retention.....	58
3.1.3.2.	Druid for Real-time Data analytic.....	59
3.1.4.	Secured Storage Domain.....	61
CHAPTER 4	IMPLEMENTATION.....	63
4.1.	Navigating SDR Setup (1 st PicoLTE).....	63
4.1.1.	Initial Hardware and Software Configuration of 1 st PicoLTE.....	63
4.1.2	LTE Stack Launch.....	65
4.1.3.	Validating PicoLTE's Network Connectivity.....	66
4.1.4.	Signal Receiver Power Testing.....	67
4.2.	Real-time Data Streaming's Implementation Procedure.....	68
4.3.	Implementation of Real-time Data Streaming (Apache Kafka).....	69
4.3.1.	Sensors as Producers.....	73
4.3.2.	Blockchain as Consumers.....	74
4.4.	Druid on Top of Kafka.....	75
4.5.	User Interface.....	78
CHAPTER 5	RESULTS AND ANALYSIS.....	81
5.1.	Simulation Parameters.....	81
5.2.	Latency of Druid without Kafka.....	81
5.3.	Latency with Kafka in Operation.....	82
5.4.	Resource Utilization.....	84
5.5.	Throughput and Availability Trade-off.....	84
5.6.	Blockchain's Performance.....	85
CONCLUSION.....		87
6.1.	Discussion.....	87
6.1.1.	Low Latency.....	88
6.1.2.	Data Retention with Kafka.....	88
6.1.3.	Scalability.....	88
6.1.4.	Availability.....	89
6.1.5.	Integrity and Security.....	89
6.2.	Research Limitation.....	89
LIST OF REFERENCES.....		90

LIST OF TABLES

	Page
Table 2.1	Event Streaming platforms comparison.....23
Table 2.2	Summary of Apache Druid advantages vs data warehouse36
Table 2.3	Main characteristic of proposed technologies.....40
Table 2.4	NB-IoT Enhancements over LTE41
Table 3.1	Brief description of the domains.....52
Table 3.2	Broker's queue properties and description60
Table 5.1	Simulation parameters81

LIST OF FIGURES

	Page
Figure 0.1	Potential MIoT with future 6G1
Figure 0.2	A standard IoT framework architecture4
Figure 0.3	Request/Response messaging architecture.5
Figure 0.4	Attack Types in IoT6
Figure 1.1	IoT and non-IoT device connections worldwide10
Figure 1.2	Three-layered architecture12
Figure 1.3	Five-layer architecture13
Figure 1.4	mMTC as a key component of next generation of 5G.....14
Figure 1.5	The logic behind Publish/subscribe system architecture15
Figure 2.1	IoT network architecture.....19
Figure 2.2	Apache Kafka's required criteria.....22
Figure 2.3	Client-server communication model.....25
Figure 2.4	Complex client-server communication26
Figure 2.5	Kafka communication model.....27
Figure 2.6	Kafka's architecture.....28
Figure 2.7	Topics and partitions in Kafka.....29
Figure 2.8	Messages in Kafka30

Figure 2.9	Batches in Kafka messages	31
Figure 2.10	An overview of Kafka producer components	32
Figure 2.11	Consumer group consuming from topics	33
Figure 2.12	Brokers and clusters in Kafka	35
Figure 2.13	Number of IoT devices	37
Figure 2.14	Wireless network access technologies	38
Figure 2.15	The NB-IoT Protocol Stack Architecture vs. OSI Reference Model.....	43
Figure 2.16	5G three service sectors	45
Figure 2.17	Application & transport layers in IoT	46
Figure 2.18	MQTT mechanism	48
Figure 2.19	LwM2M protocol stack.....	49
Figure 3.1	Architecture for MIIoT with distinct domains	51
Figure 3.2	Data flow featured with three zones	53
Figure 3.3	Sensor/Gateway domain	54
Figure 3.4	(a) BL654; (b) ibNav sensors; (c) Pinnacle 100 DVK.....	54
Figure 3.5	Connectivity domain	56
Figure 3.6	Two PicoLTEs and their antennas	57
Figure 3.7	Data retention & processing domain.....	58
Figure 4.1	Hardware setup of the 1 st PicoLTE	64

Figure 4.2	Software setup of the 1 st PicoLTE	65
Figure 4.3	LTE Stack has been launched.	66
Figure 4.4	Validating PicoLTE's connectivity.	67
Figure 4.5	LTE Signal Strength	68
Figure 4.6	Kafka implementation procedure.....	69
Figure 4.7	Kafka Containers are up and running	71
Figure 4.8	Cluster's overview	72
Figure 4.9	List of topics	73
Figure 4.10	Producer procedure.	74
Figure 4.11	Topics are receiving data	74
Figure 4.12	IPFS user interface for monitoring	75
Figure 4.13	Apache Druid's dashboard	76
Figure 4.14	Loading data from different sources into Apache Druid	77
Figure 4.15	Ingestion from Kafka	78
Figure 4.16	User interface dashboard.....	79
Figure 5.1	Apache Druid's latency without Kafka.....	82
Figure 5.2	Apache Druid latency with Kafka.....	83
Figure 5.3	Request pool usage	84
Figure 5.4	Throughput comparison with one and two brokers	85

Figure 5.5	Performance of Hyperledger	86
------------	----------------------------------	----

LIST OF ABBREVIATIONS AND ACRONYMS

3GPP	3rd Generation Partnership Project
6LoWPAN	IPv6 over Low-Power Wireless Personal Area Networks
AMPS	Advanced Mobile Phone System
AS	Access Stratum
CIoT	Cellular Internet of Things
CoAP	Constrained Application Protocol
DHT	Distributed Hash Tables
DTLS	Datagram Transport Layer Security
ECDHE	Elliptic Curve Diffie–Hellman Ephemeral
EC-GSM-IoT	Enhanced Coverage GSM for IoT
eMBB	Enhanced Mobile Broadband
EPC	Evolved Packet Core
ETL	Extract, Transform, and Load
EXALTED	EXpanding LTE for Devices
FDD	Frequency Division Duplex
GSM	Global System for Mobile communication
IPFS	InterPlanetary File System

XX

LAN	Local Area Network
LPWA	Low Power Wide Area
LTE-M	Long-Term Evolution for Machine-Type Communication
LTS	Long Term Support
LoRaWAN	Long Range Wide Area Network
LwM2M	Lightweight Machine-to-Machine
MA	Microservice-based Architecture
MAC	Media Access Control
MBB	Mobile Broadband
MCL	Maximum Coupling Loss
mMTC	Massive Machine-Type Communication
MQTT	Message Queuing Telemetry Transport
MQTT-SN	Message Queuing Telemetry Transport for Sensor Networks
NAS	Non-Access Stratum
NB-IoT	Narrowband Internet of Things
NFR	Non-Functional Requirements
NR	New Radio

OMA	Open Mobile Alliance
OMA-DM	Open Mobile Alliance Device Management
OSI	Open Systems Interconnection
OSCORE	Object Security for Constrained RESTful Environments
P2P	Point-to-Point
PCIe	Peripheral Component Interconnect express
PDCCP	Packet Data Convergence Protocol
PRB	Physical Resource Block
PSK	Pre-Shared Key
QoS	Quality of Service
RLC	Radio Link Control
RRC	Radio Resource Control
RFID	Radio Frequency Identification
RTID	Real-Time IDentification
SDK	Software Development Kit
SDR	Software-Defined Radios
TCP	Transmission Control Protocol

TM	Transmission Mode
IoT	Internet of Things
MTD	Machine-Type Devices
TBS	Transport Block Size
TDD	Time Division Duplex
TLS	Transport Layer Security
TPS	Transaction Per Second
MIoT	Massive Internet of Things
UDP	User Datagram Protocol
UE	User Equipment
URLLC	Ultra-Reliable Low Latency Communications
UUID	Universally Unique IDentifier
V2X	Vehicle-to-X
WCDMA	Wideband Code Division Multiple Access

INTRODUCTION

The Internet of Things (IoT) embodies a revolutionary shift in the digital landscape, where everyday objects are embedded with sensors, actuators, and connectivity, enabling them to communicate and exchange data. The concept of IoT is defined as a collection of smart objects with the fundamental aim of "connecting unconnected." (Kumari & Jain, 2023). Smart devices collaborate to enhance service quality for end users, striving to streamline data collection and processing for greater speed and convenience. Research indicates a projected activation of more than 75 billion IoT devices by the year 2025 (Shehada, Gawanmeh, Yeun, & Jamal Zemerly, 2022).

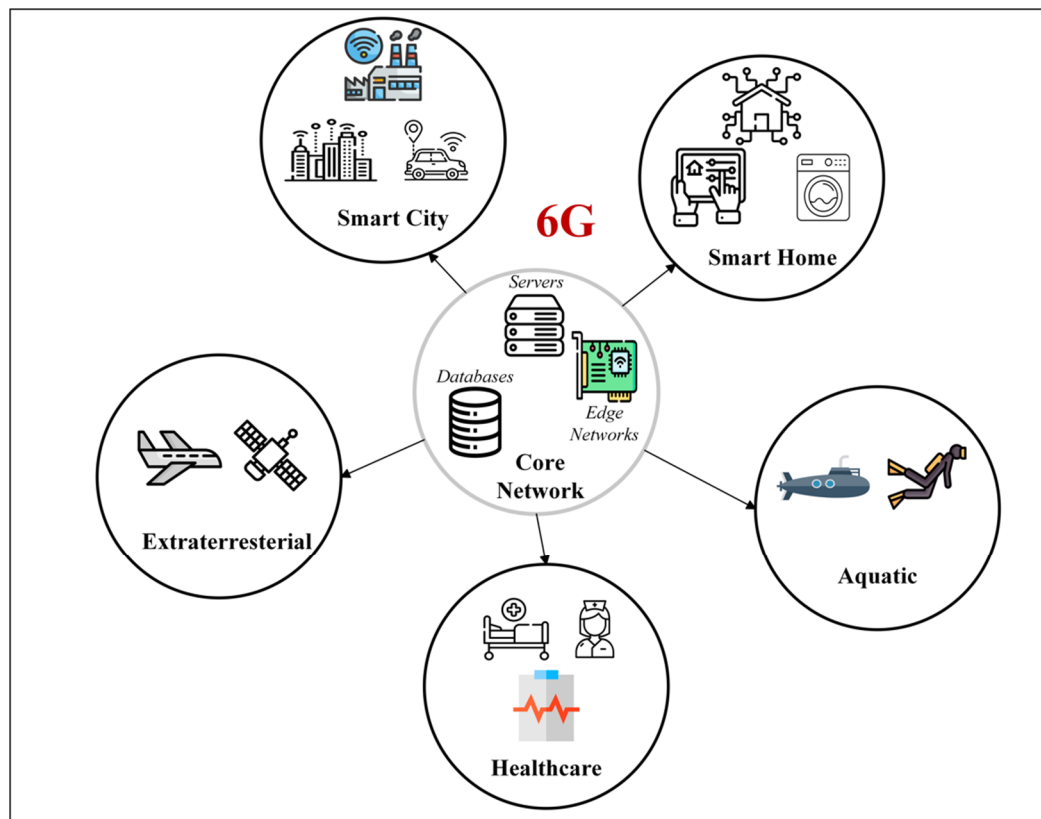


Figure 0.1 Potential MIoT with future 6G
Adapted from Alotaibi & Barnawi, (2023)

As it can be seen in Figure 0.1, in the forthcoming years, the potential for sixth-generation (6G) networks to enable communication among billions of interconnected devices and applications, operating at significantly heightened data speeds, leads us to the emergence of the Massive Internet of Things (MIoT) or massive Machine-Type Communication (mMTC) concept (Alotaibi & Barnawi, 2023).

At the commencement of the 3rd Generation Partnership Project (3GPP), a collection of requirements for mMTC was delineated, emphasizing the significance of maintaining communication quality-of-service (QoS) for one million devices per square kilometer (1 device/km²). Furthermore, a 10 year battery lifespan is sought for intermittent transmission of compact uplink packets, along with a latency requirement of less than 10 ms for dispatching alarm messages, even in areas with limited coverage (Youn, Park, Kim, You, & Cho, 2021). Nevertheless, this widespread expansion of mMTC and its requirements poses challenges, particularly in the real-time storage and analysis of massive, heterogenous data streams. In addition to these challenges, it is essential to focus on implementing a solution that incorporates scalability, availability, security, and minimal latency for MIoT.

Effective operation of IoT-based frameworks depends on essential characteristics, encompassing: i) fostering connectivity among devices for collaborative intelligence, ii) precise and timely sensing to detect environmental changes, iii) intelligent analysis of collected data with minimal latency for extracting meaningful insights, iv) dynamic flexibility to take into account system evolution, v) scalability to manage growing device and data volumes, vi) handling heterogeneity among devices and data types, vii) and implementing robust security measures to safeguard against cyber threats and data breaches. These attributes collectively mold a dependable and streamlined IoT ecosystem (P.K. et al., 2022). In line with the mentioned characteristics, a requisite framework capable of meeting these criteria is essential to support the full spectrum of features (Banafaa et al., 2023).

0.1. Problem Description

The importance of sensors is felt more and more when it comes to meeting the need to minimize environmental damage and maximize the use of Earth's resources. In many situations, it becomes imperative to use sensing to create a virtual twin of the real world to increase efficiency and environmental sustainability. The digital twin will be able to sense and digitize more richer data from the physical world, which will enable it to make better judgments and quickly transmit those conclusions back to the physical world, maximizing resource efficiency and reducing environmental impact (Beale, Uchiyama, & Clifton, 2021).

Illustrated in Figure 0.2 as a sample, a centralized environment, such as cloud, may be used to handle the massive data produced by MIoT sensors. Data must be sent from intelligent devices to the cloud in an efficient and flexible manner in order to manage massive amounts of data from such devices (Sai Lohitha & Pounambal, 2023). IoT includes energy-efficient sensor or actuators, which are distinguished by their constrained battery life, processing power, and compute capability (Mammela, Karhula, & Makela, 2019).

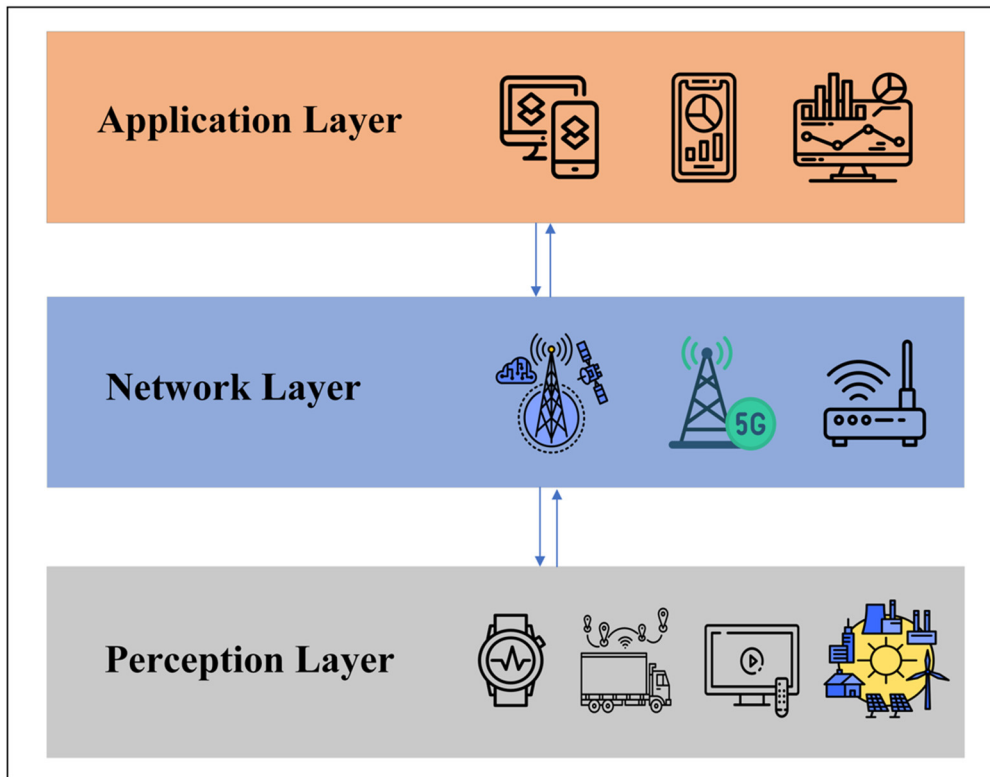


Figure 0.2 A standard IoT framework architecture

Adapted from Alotaibi & Barnawi, (2023)

However, traditional request/response data transfer methods (shown in Figure 0.3) are often insufficient for handling communication between IoT devices and the cloud, particularly when dealing with large volumes of data. With a large number of devices, each sending requests and awaiting responses, the overhead of managing these requests and responses can become substantial. When IoT devices rely on request and response mechanisms, they must maintain real-time control to effectively manage applications, leading to elevated network congestion and heightened latency. Therefore, the primary and foremost challenge to address involves the effective real-time processing of massive and diverse data with minimal latency.

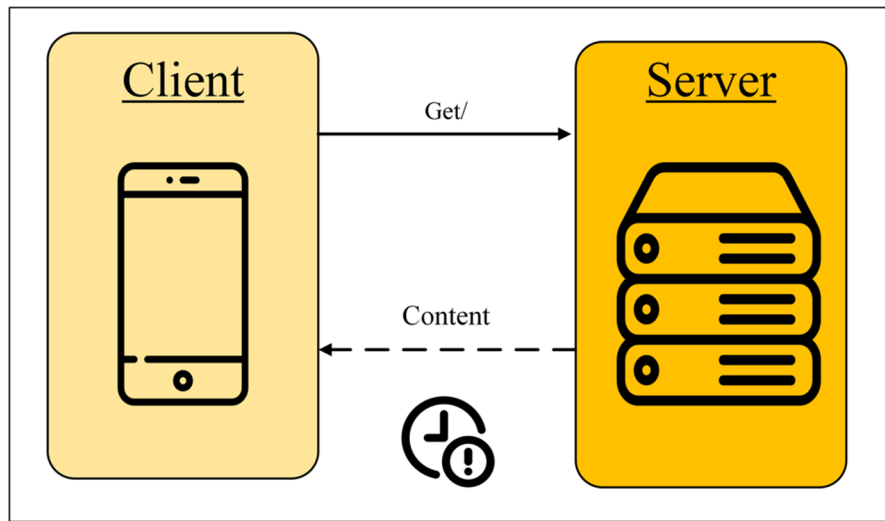


Figure 0.3 Request/Response messaging architecture

The second challenge pertains to the system's connectivity reliability. In distributed and decentralized MIoT environments, where numerous devices collaborate, ensuring constant availability is crucial. It is important to implement a solution with enhanced reliability and fault tolerance. The concern lies in the potential presence of faulty or malicious nodes within the network, posing a threat to the overall system performance. This critical feature is especially vital in MIoT systems where device communication and coordination in massive scale make the impact of a single node failure significant (Marcozzi, Gemikonakli, Gemikonakli, Ever, & Mostarda, 2023).

Another crucial concern involves secure storage challenges, which escalate alongside the growing adaptability of IoT systems. The expanding scope of IoT systems raises concerns regarding the security and privacy of IoT data (Agrawal et al., 2018). The resource-constrained nature of smart devices within the IoT architecture renders them susceptible to a variety of security threats. Each layer of the IoT architecture encounters distinct security challenges, making the design of a comprehensive security model challenging due to the inherent heterogeneity. Complicating matters further, security attacks are evolving in sophistication as it is shown in Figure 0.4. Notable threats in the IoT architecture encompass malicious node injection, impersonation, physical attacks, phishing, jamming, and data leakage. Effectively

mitigating these security attacks demands robust technological solutions (Gugueoth, Safavat, Shetty, & Rawat, 2023).

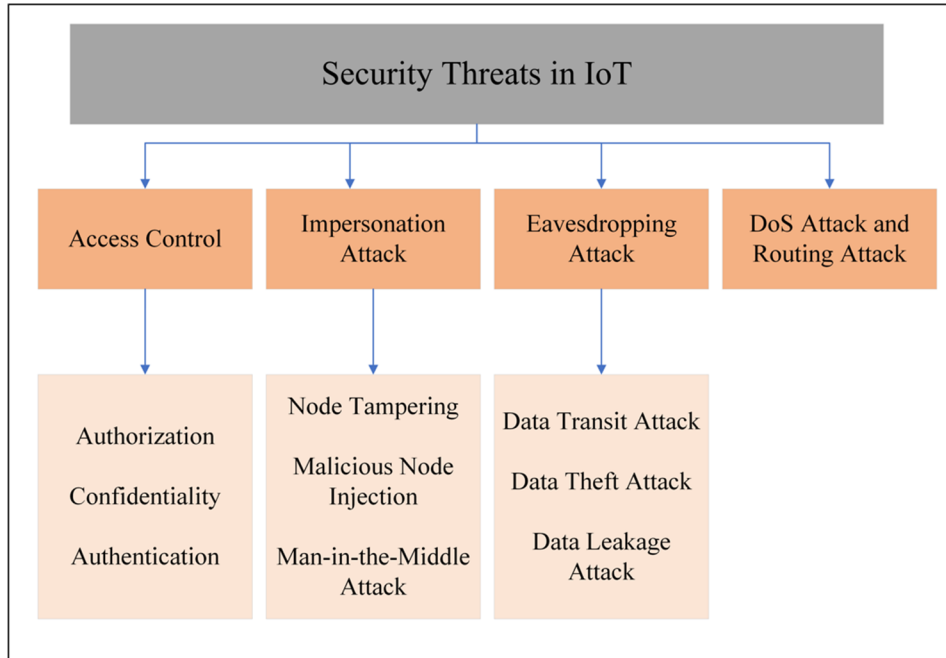


Figure 0.4 Attack Types in IoT
Adapted from Gugueoth et al., (2023)

To sum up, the challenges that we are facing lies in the effective real-time processing of diverse data with minimal latency. Additionally, ensuring the reliability of system connectivity in distributed MIoT environments is crucial, necessitating enhanced fault tolerance. Security concerns, particularly in the realm of IoT data storage and the susceptibility of resource-constrained smart devices to evolving security threats, further underscore the importance of robust technological solutions in this dynamic landscape.

0.2. Research Goals

Drawing from the problematic section presented above, this thesis aims to achieve the following main objective:

“Develop and Validate an Innovative Approach for Real-time Data Processing in Massive IoT (MIoT) Using Publish/Subscribe Method and Blockchain Technology”

A thorough examination of challenges within the domain of MIoT reveals the need for careful consideration in implementing successful solutions. The primary challenge revolves around the efficient processing of massive and heterogeneous data in real-time with minimal latency. Concurrently, ensuring network availability and establishing a secure, immutable data storage mechanism is paramount.

Core contributions of this thesis are:

1. *Comprehensive Architectural Framework*: Introduction of an architectural framework rooted in the publish/subscribe methodology designed for real-time data processing in the expansive domain of MIoT.
2. *Utilization of Apache Kafka and Apache Druid*: To address the intricacies of real-time data processing, the research employs the robust capabilities of Apache Kafka and Apache Druid.
3. *Incorporation of Hyperledger Fabric*: Recognizing the significance of data storage security and immutability, the research incorporates Hyperledger Fabric, a cutting-edge blockchain technology, to fortify the system's data storage capabilities.
4. *Network Availability and Efficiency*: Deployment of two Software-Defined Radios (SDR) based on LTE as a "Network-in-a-box" to simulate robust network connectivity.
5. *Experimental Validation*: Substantiation of the proposed model through designed experiments aimed at quantifying latency and assessing the overall performance of the blockchain-integrated solution.

In summary, this research not only introduces a comprehensive architectural framework but also provides empirical evidence of its practicality and efficiency, substantively contributing to the field of MIoT.

CHAPTER 1

LITERATURE REVIEW

1.1. IoT Evolution

IoT, short for the Internet of Things, represents a ground-breaking and swiftly advancing technology with the potential to transform our interactions with the environment (C. Li, Wang, Wang, & Zhang, 2024). The concept of the IoT was first coined in the 1990s. The primary contributions were made by IoT pioneer Kevin Ashton in 1999, co-founder of MIT's Auto-ID Laboratory. Ashton introduced the term "The Internet of Things" to characterize a system in which the Internet is linked to the physical world through pervasive sensors, including Radio-frequency identification (RFID) (Taj et al., 2023).

The ubiquity and pervasive nature of IoT have led to its widespread adoption across diverse sectors, such as healthcare, smart cities, building, energy and smart grid, manufacturing industry, weather forecasting, environmental monitoring, logistics and resource management, agriculture, smart shopping, automotive, and more (Rahim et al., 2021). The introduction of the Raspberry Pi microcontroller in 2011 marked a significant milestone in the development of IoT. This cost-effective microcontroller facilitated access to IoT, making it accessible to end-users. While other microcontrollers were available, the Raspberry Pi stood out for its affordability, simplicity, substantial processing power, and abundant online resources, including tutorials, videos, educational DIY websites, blogs, and forums. As a result, IoT was no longer confined to commercial ventures but became a viable pursuit for a broader audience (Ande, Adebisi, Hammoudeh, & Saleem, 2020).

The evolution of IoT in terms of the number of connected devices has been remarkable. According to (Rahim et al., 2021), it is expected that by 2025, the total number of IoT-connected devices will reach 75.44 billion. This exponential growth is a testament to the widespread adoption and integration of IoT devices into various aspects of daily life. IoT has indeed stepped out of its infancy and is on the path to becoming the next revolutionary technology, as highlighted by (Gubbi, Buyya, Marusic, & Palaniswami, 2013). The increasing

number of connected devices has significantly reduced human efforts, increased resource utilization, and saved time. However, as pointed out, this expansion has also brought about significant challenges such as the lack of security and privacy, making IoT devices more vulnerable to attacks (Shobhit Verma et al., 2022). Figure 1.1 shows IoT and non-IoT active device connections worldwide from 2010 to 2025 (“Global IoT and Non-IoT Connections 2010-2025 | Statista,” 2022)

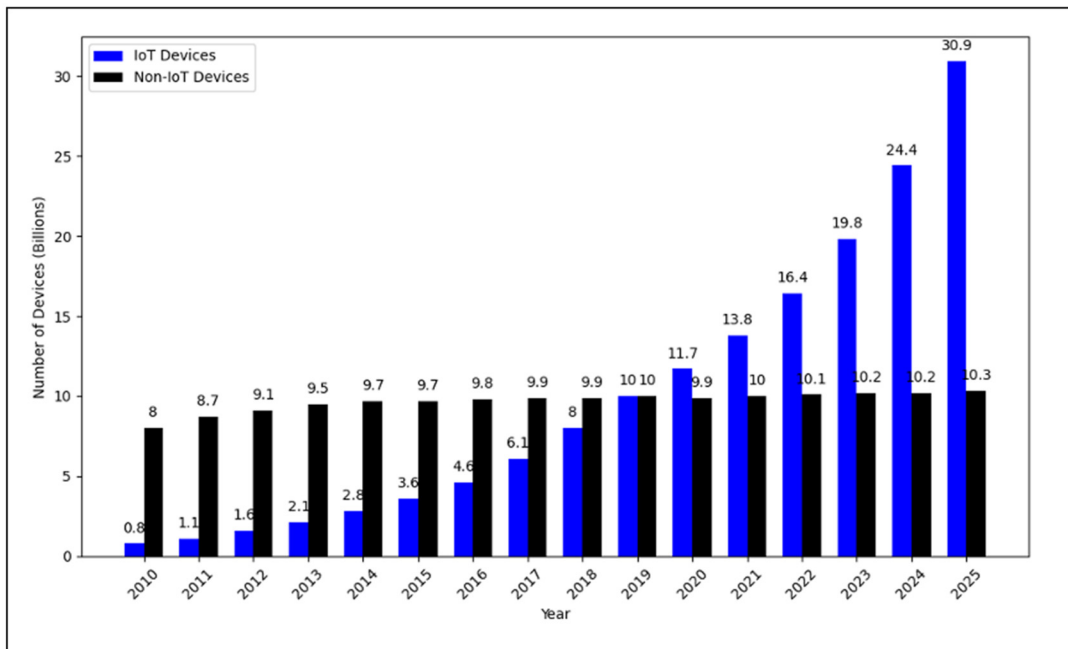


Figure 1.1 IoT and non-IoT device connections worldwide

Adapted from “Global IoT and Non-IoT Connections 2010-2025 | Statista”, (2022)

1.2. Massive IoT

The expansion of the IoT, linking billions of machines, devices, and entities, is swiftly evolving into a more expansive concept known as Massive IoT (MIoT) or mMTC (Shikhar Verma, Kawamoto, Fadlullah, Nishiyama, & Kato, 2017). MIoT has surfaced as a novel paradigm within IoT networks, characterized by its emphasis on scale rather than communication speed. In this category, the quantity of connected devices can vary from hundreds to billions, with the

primary objective being the efficient transmission of limited sensing data from a multitude of devices (Jouhari, Saeed, Alouini, & Amhoud, 2023). Consequently, the creation of intelligent, efficient, flexible, and cost-effective IoT systems within the framework of the MIoT paradigm is increasingly challenging, given the growing demands for IoT connectivity and diverse application requirements (Alshaikhli, Elfouly, Elharrouss, Mohamed, & Ottakath, 2022).

The majority of mMTC use cases depict scenarios where a substantial number of machine-type devices sporadically communicate in expansive areas, typically without specific latency requirements and reliability guarantees (Pokhrel, Ding, Park, Park, & Choi, 2020). Addressing these use cases is commonly reframed by fulfilling the following key performance indicators (KPIs) (Pokhrel et al., 2020):

1. Achieving a massive connection density of 10^6 devices per square kilometer in urban environments.
2. Maintaining a Maximum Coupling Loss (MCL) of up to 164 dB for extensive coverage.
3. Ensuring a device battery lifetime exceeding 10 years with a stored energy capacity of 5 Wh.

1.3. IoT Layered Architecture

In terms of layer, we can divide IoT architectures into 2 categories: Three-layer and five-layer. The three-layer architecture is fundamental in IoT design, comprising the perception layer, network layer, and application layer as illustrated in Figure 1.2 (Al-Qaseemi, Almulhim, Almulhim, & Chaudhry, 2016).

1. **Perception Layer:** In the overarching structure of IoT, a bottom-up methodology is employed, with the perception layer forming its foundational tier. This layer serves as the physical embodiment of IoT architecture, tasked with capturing data and physical parameters from the environment through the utilization of diverse sensors, actuators, and other intelligent equipment.

2. **Network Layer:** The information gathered from the perception layer is relayed and transmitted to additional network devices, servers, and applications for subsequent processing through the network layer.
3. **Application Layer:** Positioned at the summit of the IoT architecture, the application layer's primary role is to furnish application-specific services to end-users, thereby facilitating an intelligent IoT-driven environment. This layer encompasses programs and modules that businesses or enterprises leverage to access real-time data and formulate informed decisions for their operational activities (Taj et al., 2023)

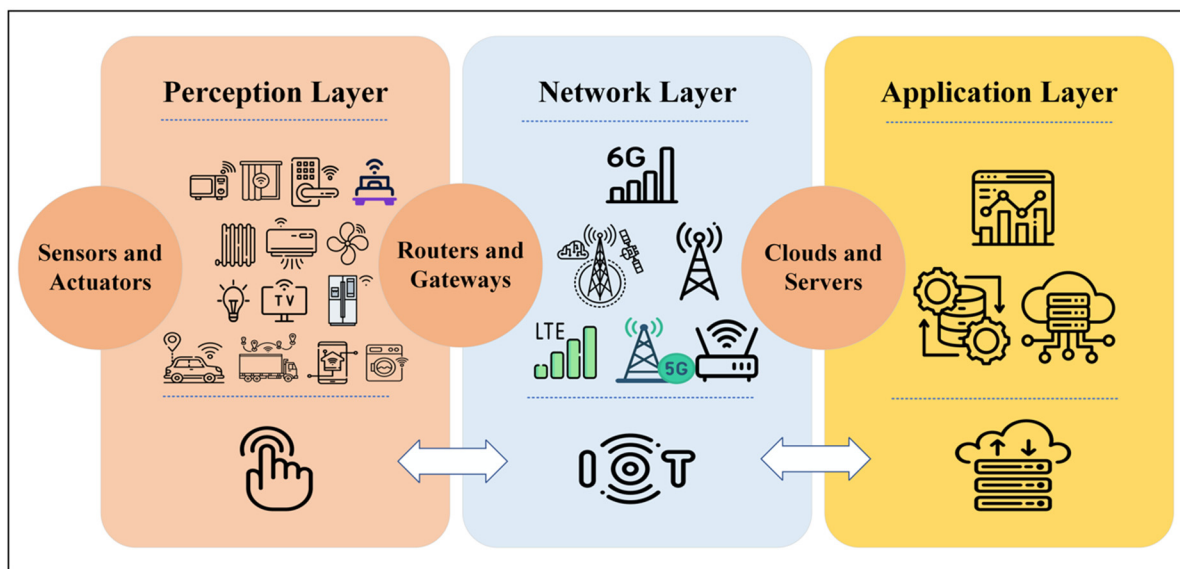


Figure 1.2 Three-layered architecture

Adapted from Taj et al., (2023)

The five-layer architecture maintains the core functionalities of the three-layer model, encompassing the perception, network, and application layers. In addition, it introduces two supplementary layers, namely the access gateway layer and the middleware layer. The access gateway layer is tasked with overseeing IoT communication within a specific environment, facilitating the exchange of messages between objects and systems. On the other hand, the middleware layer enhances the association between hardware devices and applications, offering a more adaptable connection, as depicted in Figure 1.3 (Al-Qaseemi et al., 2016).

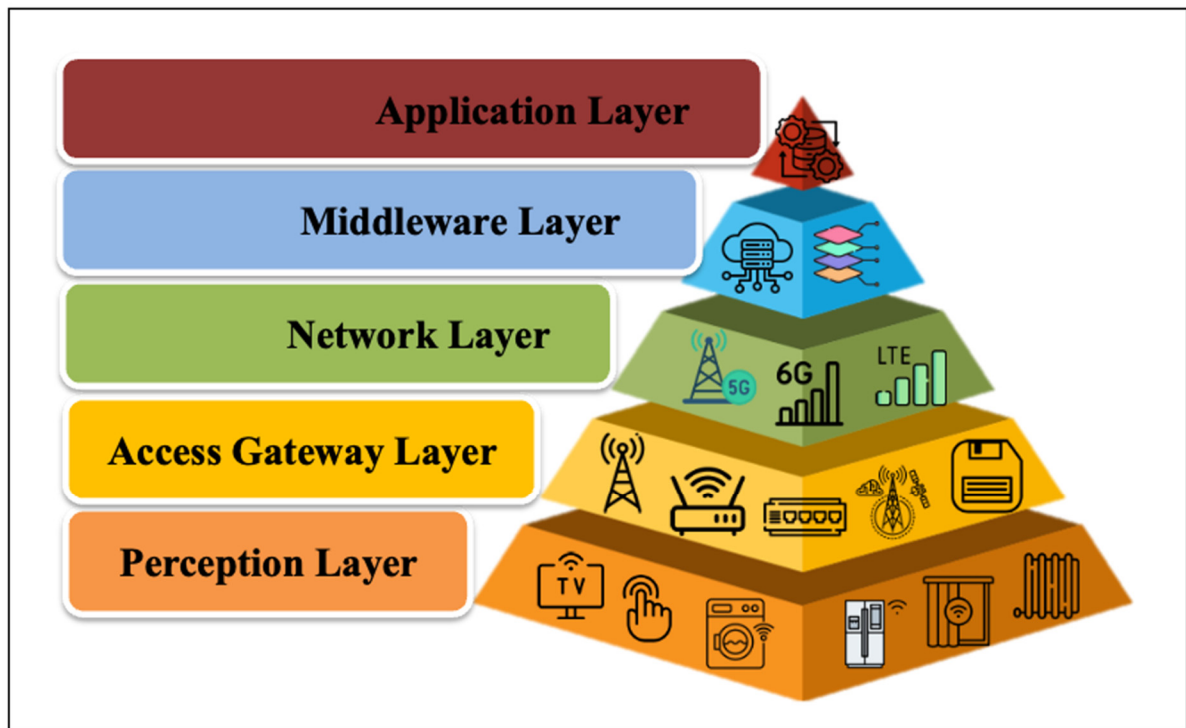


Figure 1.3 Five-layer architecture
Adapted from Al-Qaseemi et al., (2016)

As it has been mentioned earlier, mMTC is different with traditional human-type communication since it concentrates on massive number of connections, connectivity, and reliability considering limited radio resources, power consumption, and small data packet size. Although connectivity is the primary requirement for mMTC, reliable access is also required because there may be a lot of IoT applications in the future. As it is shown in Figure 1.4, mMTC is considered as one of the key components of MBB and is going to be promoted as the result of advance in 5G. This also improves two other components of MBB which are “Data rate” and “Reliability” (Wang & Ma, 2019). Therefore, in MIIoT a clearly defined architecture that will offer a scalable, dynamic, and secure foundation for its deployment is crucial to its success (Abdmeziem, Tandjaoui, & Romdhani, 2015). The development of a standard architecture and messaging exchange is intimately tied to IoT deployment which mainly can be divided into two categories: 1) Request/response and 2) Publish/subscribe.

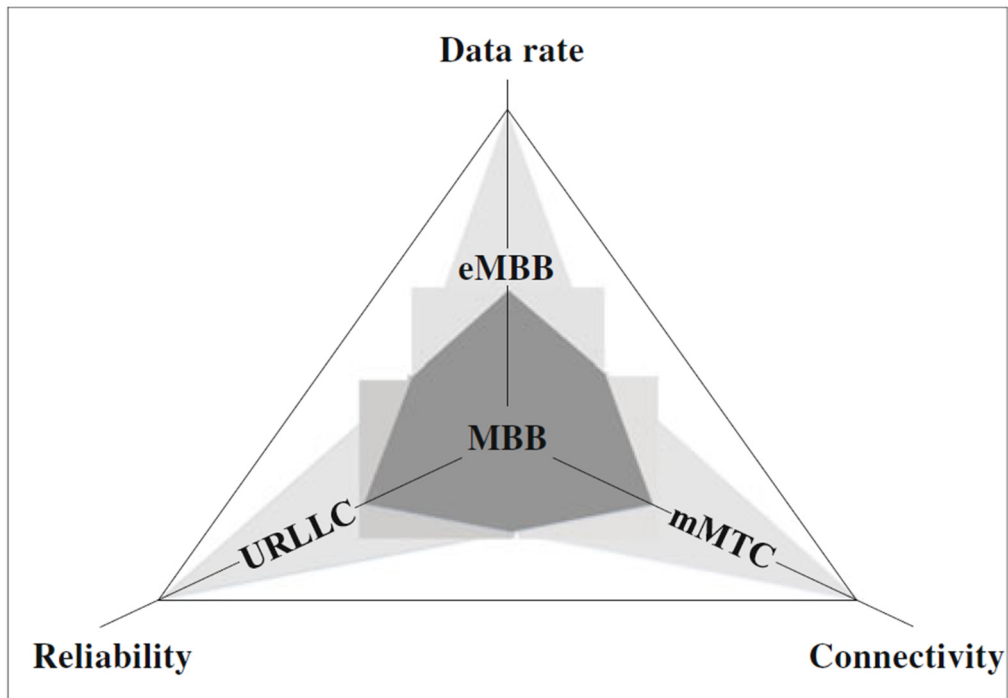


Figure 1.4 mMTC as a key component of next generation of 5G

Taken from Wang & Ma, (2019, p. 2)

1.3.1. Request/Response Messaging Method

A crucial element in IoT involves the utilization of distributed online devices that communicate through Internet protocols. The request/response or request/reply mechanism stands as one of the fundamental methods for data exchange. In this process, the initial computer issues a request for specific data, and the second computer replies to this request. In other words, the recipient system receives data, processes, and subsequently sends back a response message. The request/response scheme was employed in situations where the server is restricted from initiating data transfers to a client, as observed in certain applications within cyber–physical systems (Henneke, Elattar, & Jasperneite, 2015). Such traditional data exchange method is inadequate for facilitating communication between IoT devices and the cloud. In instances where request and response are employed in IoT, the device must exert real-time control to

consistently manage applications. This, in turn, results in heightened network traffic and increased latency (Sai Lohitha & Pounambal, 2023).

1.3.2. Publish/Subscribe Messaging Method

On the other hand, the publish/subscribe mechanism enables entities to subscribe to topics residing in brokers and define QoS requirements. As it can be seen in Figure 1.5, publishers send data to specific topics in brokers which is automatically transmitted to entities that had subscribed to these topics beforehand. The primary distinction from the request/response approach lies in the architecture, which is not reliant on a single server but functions in a peer-to-peer (P2P) manner. In this context, the significance lies in the data's topic rather than its source. This pattern is especially fitting for IoT since numerous sensors generate data periodically. Therefore, a push-pattern can minimize network activity as opposed to employing a request/response approach (Johnsen, 2018). In other words, publish/subscribe messaging effectively manages event transmission, wherein the publisher releases information that becomes accessible to all subscribers in an asynchronous manner (Shi, Zhang, & Chen, 2020).

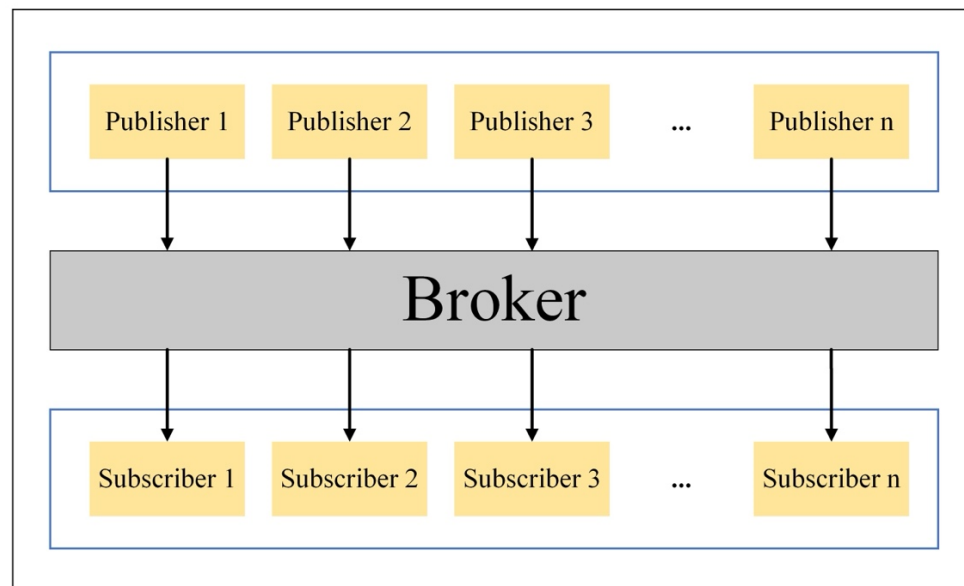


Figure 1.5 The logic behind Publish/subscribe system architecture

1.4. Related Work in MIIoT Data Processing

Real-time data processing in MIIoT is a critical aspect of enabling efficient and effective utilization of IoT-generated data. The ability to process and analyze data in real-time directly impacts the decision-making process, enhances system responsiveness, and enables the development of intelligent applications in various domains such as healthcare, transportation, and smart cities (Yang, Puthal, Mohanty, & Kougianos, 2017). In this literature review, we discuss significant research contributions that address real-time data processing in the context of MIIoT. One essential aspect that has been highlighted in the literature is the need for scalable and efficient data processing and solution for next-generation IoT and wireless sensor networks.

In their work referenced as (Siddiqui, Khendek, & Toeroe, 2023), the authors conducted an extensive review of the current state of microservice-based architecture application, emphasizing its role in enhancing non-functional aspects, particularly reliability and availability, within IoT ecosystems. Their review outlines various crucial challenges that microservices address in the realm of IoT, encompassing integration of IoT devices, heterogeneity, interoperability, fault tolerance, scalability, and system deployment and configuration. Additionally, the authors provide insights into the techniques proposed for managing non-functional requirements (NFRs) within microservice-based architectures for IoT systems. Specifically, to enhance availability, techniques such as reactive architecture, circuit breaker patterns, orchestration, and machine learning are highlighted. Messaging protocols are identified as preferred solutions for addressing interoperability challenges. Furthermore, in the context of scalability, the authors explore techniques such as orchestration and load balancing. However, the authors underscore that the utilization of MAs has not been uniformly examined across all domains within IoT. Particularly noteworthy, another study identified a significant gap in the literature regarding the comprehensive assessment of end-to-end availability and reliability. This emphasis goes beyond the mere consideration of the availability of individual components within a system (Díaz, Martín, & Rubio, 2016).

Khashan & Khafajah, 2023 suggested an architecture, which combines elements of both centralized and blockchain-based approaches, tackling critical issues encountered by resource-constrained IoT devices. The author contends that conventional cryptographic methods, while extensively explored, are not well-suited for the limited resources of IoT devices, aligning with the findings of the study by (Medileh et al., 2020). The integration of blockchain technology for IoT authentication, as introduced in that study, has gained increasing interest in recent years. Notably, the focus of that study on minimizing computational costs and addressing real-time requirements sets it apart from existing blockchain-based approaches, which often introduce substantial overhead. The innovative hybrid architecture proposed in this work, incorporating both centralized and blockchain-based components, emerges as a promising avenue for enhancing IoT security efficiently, while addressing the resource and scalability challenges inherent in IoT systems.

Within the domain of publish/subscribe systems, (Lazidis, Tsakos, & Petrakis, 2022) provide insights into the essential design characteristics and performance metrics of various open-source systems, including Apache Kafka and RabbitMQ. This current research contributes by conducting a thorough assessment of seven open-source systems, establishing standardized criteria for comparison, and offering insights into their functionality and performance in real-world scenarios. However, there is a need for further exploration of alternative systems such as ActiveMQ, Apache Pulsar, ZeroMQ, Redis, and others. Such exploration would contribute to a more comprehensive understanding of the capabilities and limitations of publish/subscribe systems.

In the field of fog computing, (Amanlou, Hasan, & Bakar, 2021) present a valuable lightweight authentication scheme designed specifically for resource-constrained IoT devices and fog gateways. Conversely, pre-shared key (PSK) authentication methods have been suggested as a solution for devices with limited resources, albeit facing scrutiny regarding their security robustness (Amnalou & Azmi, 2020). The study introduces an innovative approach by combining the Elliptic Curve Diffie–Hellman Ephemeral (ECDHE) key exchange algorithm with PSK authentication within a Message Queuing Telemetry Transport (MQTT) publish/subscribe framework in the context of distributed fog computing. However, the study's

focus on ECDHE-PSK authentication within the MQTT publish/subscribe architecture might constrain its applicability to other fog computing and IoT contexts.

The work presented by (Rahman & Das, 2022) introduces the RTID framework, employing Apache Spark for streamlined data processing, RESTful API for efficient data access, and OAuth 2.0 for secure access management. In the domain of real-time IoT data processing, previous research has focused on addressing the intricate challenges associated with the effective management of massive amount of data (Malek et al., 2017). Traditional methods, such as 6LoWPAN (Jenzeri & Chehri, 2022) and the IoT with cloud system (Dawaliby, Bradai, & Pousset, 2016), have been integral in shaping the IoT landscape but have encountered limitations in terms of adaptability and data management efficiency. While the RTID framework contributes to the improvement of real-time MIIOT data processing, conducting more comprehensive comparative studies with existing frameworks is imperative to firmly establish the suitability and feasibility of the RTID framework in practical deployment scenarios.

In contrast to prior endeavors in IoT data processing, which frequently addressed isolated challenges utilizing diverse methods and architectures such as publish/subscribe and microservice-based approaches, some of which overlooked the substantial scale in MIIOT, our research adopts a holistic approach. Our principal aim is to introduce a comprehensive framework that consolidates all these challenges into a unified architecture. Our thesis seeks to offer a detailed exploration of real-time data processing in MIIOT and aims to collectively tackle these formidable challenges.

CHAPTER 2

BACKGROUND

2.1. Real-time Data Processing in MIIoT

As previously mentioned, data volumes are increasing quickly as a result of the proliferation of connected IoT devices which referred as MIIoT. This exponential increase is fueling the need for real-time analytics on the massive amounts of IoT data. Delivering business knowledge and actionable insights as soon as possible is the main objective. Regretfully, the unique requirements of real-time IoT analytics are frequently not met by the analytics architecture of contemporary IoT systems. Unfortunately, a lot of academics have neglected to include in their studies the need for a well-designed IoT framework.

It is specifically necessary to have a robust infrastructure in order to process and analyze the massive amounts of data generated by the IoT (Ge et al., 2016). As indicated in Figure 2.1, when categorizing the entire IoT network into three key segments—namely, the sensing, delivery, and analytics networks—our emphasis is on the analytical component (Shikhar Verma et al., 2017).

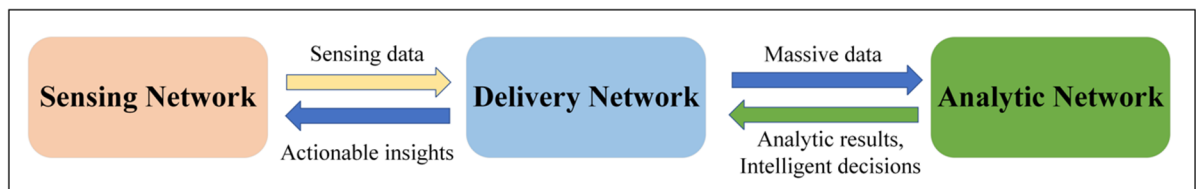


Figure 2.1 IoT network architecture

Adapted from Shikhar Verma et al., (2017)

In our research, we aimed to align with this architecture by employing suitable platforms for each phase. Specifically, we utilized Apache Kafka and Apache Druid within our analytics network. Apache Kafka, employing a publish/subscribe approach, addresses the limitations of conventional request/response messaging systems. Concurrently, Apache Druid enhances real-

time data processing capabilities, thereby improving the framework's ability to conduct instantaneous analytics and supporting statistical visualizations.

2.1.1. Data Pipeline Tools for Data Retention & Processing

Today's businesses produce enormous amounts of data. Unbelievably large amounts of data are being generated by our internet usage. This massive data is dispersed among the various business-used systems, including cloud applications, databases, SDKs, etc. Deep analysis is needed in order to extract useful information from this data. Companies would initially like to bring this data to a single location for straightforward access and further analysis (Tawakuli & Engel, 2022).

The mechanism by which ETL (Extract, Transform and Load) activities are carried out is a data pipeline. We would need to execute ETL in order to be able to derive true insights from the data:

- **Extract** information that matters to you from a variety of data sources.
- To make this data ready for analysis, **transform** and enrich it.
- **Load** this data more frequently to a Data Lake or Data Warehouse than to a single source of truth (Hanlin, Xianzhen, & Xianrong, 2012).

There are various Data Pipeline tools available depending on the goal. The following are the most common types:

- Real-time vs. Batch Data Pipeline Tools
- Open Source vs. Proprietary Data Pipeline Tools
- Cloud-native vs. On-premise Data Pipeline Tools

2.1.1.1. Batch vs Real-time Data Pipeline Tools

Utilizing solutions for batch data pipelines involves moving data at regular intervals in high volumes, sacrificing real-time functionality for efficiency. These solutions are commonly employed for on-premise data sources or in scenarios where real-time processing may disrupt

regular business operations due to resource constraints. Some prominent Batch Data Pipeline tools include Informatica PowerCenter, IBM InfoSphere DataStage, Talend, and Pentaho.

On the other hand, real-time ETL solutions are specifically designed for processing data in real-time. These tools are ideal for situations where instant analysis is needed, such as extracting data from user interactions on a website or mobile application from a streaming source. Some well-known real-time data pipeline tools in this category include Hevo Data, Apache Kafka, Estuary Flow, and StreamSets.

2.1.1.2. Open-source vs Proprietary Data Pipeline Tools

Open-source tools, being publicly accessible, require customization for each use case. These data pipeline tools are typically free or have minimal costs, but their growth and enhancement depend on the user's knowledge. Popular open-source data pipeline tools include Talend, Apache Kafka, and Apache Airflow.

In contrast, proprietary data pipeline tools are designed for specific business applications, eliminating the need for customization or technical expertise from the user. These tools operate effectively out of the box. Some notable proprietary data pipeline tools include Hevo Data, Blendo, and Fly Data.

2.1.1.3. Cloud-native vs On-premise Data Pipeline Tools

In the past, businesses stored all their data on on-premise systems, requiring the establishment of on-premise data warehouses or data lakes. These on-premise data pipeline technologies offer enhanced security as they are deployed on the customer's local infrastructure. Platforms such as Informatica PowerCenter, Talend, and Oracle Data Integrator can support on-premise data pipelines.

Alternatively, data from cloud-based sources can be processed and transferred to cloud-stored data warehouses through technologies known as "cloud-native data pipelines." In this case, the data pipeline is hosted by the vendor, relieving the client of infrastructure-related resource

burdens. Security remains a top priority for cloud-based service providers. Platforms that support cloud data pipelines include Hevo Data, Blendo, and Apache Kafka.

2.1.1.4. Why Apache Kafka for Real-time Data Processing in MIIoT?

As it is elaborated earlier, there are many data pipeline platforms which we could integrate in our MIIoT platform. In Figure 2.2, three criteria profoundly influence our decision-making process. Firstly, real-time data processing capability emerges as paramount, as our analysis demands instantaneous insights. Unlike batch data processing, which adheres to regular time intervals (more efficiency), real-time processing ensures timely responses. Secondly, the preference for open-source solutions arises from their accessibility to the public and minimal associated costs. Moreover, the flexibility to customize solutions to align with our specific requirements underscores the significance of open-source frameworks. Lastly, prioritizing cloud-native deployment holds strategic importance, alleviating infrastructure-related burdens for clients. By leveraging cloud-native solutions, the need for individualized deployment infrastructures is obviated, streamlining operations and enhancing accessibility.

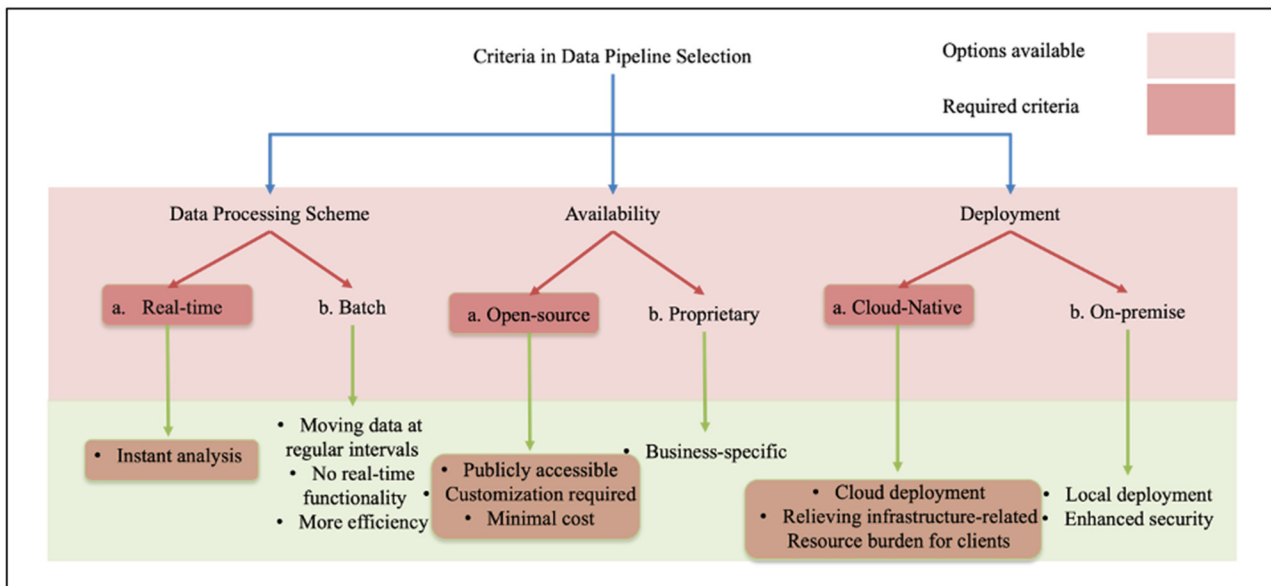


Figure 2.2 Apache Kafka's required criteria

According to the findings outlined in Table 2.1, Apache Kafka emerges as the sole Event Streaming Engine that satisfactorily fulfills all the specified criteria.

Table 2.1 Event Streaming platforms comparison

	Real-time	Batch	Open-Source	Proprietary	Cloud-Native	On-Premise
Kafka						
Airflow						
PowerCenter						
IBM						
Talend						
Pentaho						
Hevo Data						
Blendo						

However, what makes Apache Kafka a solid choice among the many publish/subscribe messaging systems available? The supportive reasons for such choice is explained in detail in the following sections:

- **Multiple producer applications:** Multiple producers can be handled by Kafka without any issues, regardless of whether they are utilizing different topics or the same topic. This makes the solution perfect for combining and standardizing data from various frontend systems. For instance, IoT sensors that provides us with content via several microservices may have a single topic for temperature that all services can write to in a standard style. Without having to coordinate consuming from many topics, one for each application, consumer applications can then receive a single stream of temperature for all applications on the site.
- **Multiple consumer applications:** Numerous consumers can read any single stream of messages without interfering with one another according to Kafka's design, which also allows for multiple producers. This is in contrast to many queuing systems, where a message is no longer accessible to other clients once it has been digested by one client.

One or more Kafka consumers may decide to work together as a group and share a stream in order to ensure that a message is only processed once by the group as a whole.

- **Scalability:** Any amount of data can be easily handled with Kafka thanks to its adaptable scalability. Users can begin with one broker as a proof-of-concept, grow to a modest development cluster of three brokers, and then transition to a bigger cluster of tens or even hundreds of brokers that expands over time as the data scales up, the ability which is absolutely requires in MIIoT. The availability of the cluster as a whole is unaffected by expansions that are carried out while it is online. This implies that a cluster of several brokers may manage the failure of a single broker and go on providing client service. Higher replication factors can be configured in clusters to handle more simultaneous failures.
- **Integrated Regulation Engine:** As mentioned earlier, Kafka has the capacity to manage multiple consumers, and its persistent message retention feature eliminates the need for immediate responses from consumers. Messages are stored with specific retention criteria on disk after being written. This retention flexibility can be customized for each topic, allowing different message streams to have varying retention levels based on consumer requirements. With durable retention, there is no risk of data loss if a consumer falls behind due to slow processing or a sudden surge in traffic. This feature also facilitates consumer maintenance, allowing temporary application shutdowns without concerns about message backlog or loss on the producer side. Even when consumers are stopped, Kafka retains the messages, enabling them to resume processing communications without data loss (Seymour, 2021).

2.1.1.5. Kafka Communication Model

The client/server, synchronous model of communication between systems may be the most typical. Applications, microservices, databases, and anything else that reads and publishes data over a network are all considered systems in this context. The client/server approach, as

depicted in Figure 2.3, is initially straightforward and involves direct system-to-system communication.

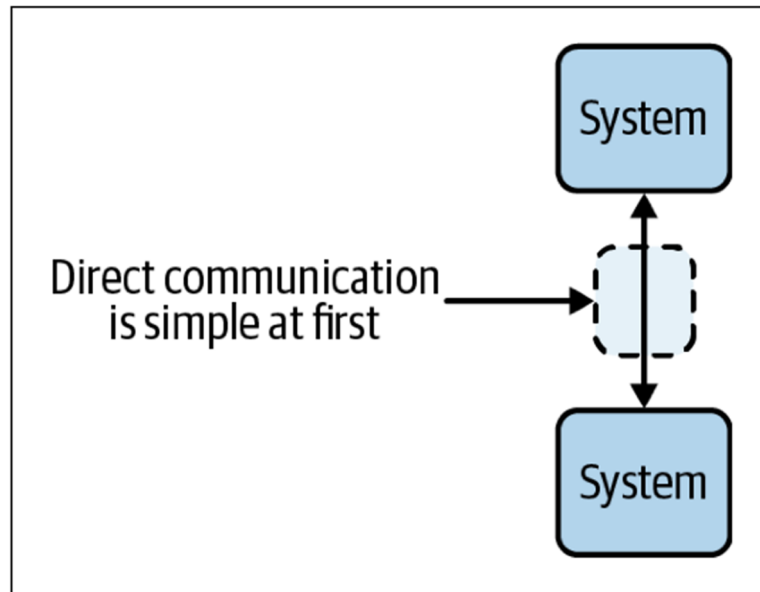


Figure 2.3 Client-server communication model
Taken from Shapira, Palino, Sivaram, & Petty, (2022)

Nevertheless, scaling point-to-point communication becomes problematic as more systems, particularly within a complex ecosystem like MIoT, need to communicate. This leads to the creation of a complex network of communication channels that can be difficult to comprehend and maintain. Figure 2.4 illustrates the challenges that can arise, even when dealing with a minimal number of systems.

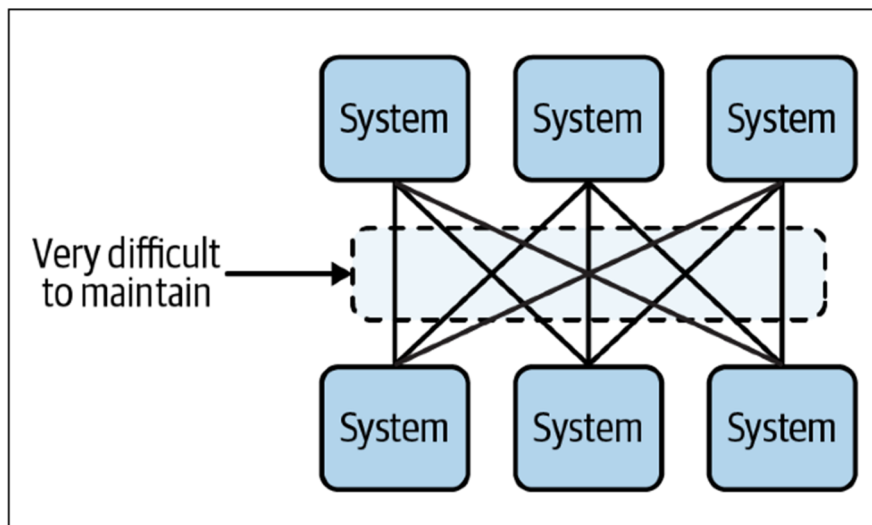


Figure 2.4 Complex client-server communication

Taken from Shapira et al., (2022)

To refer to the drawbacks of client/server model, systems become strongly connected because communication between them rely on mutual understanding. This makes upkeep and updating these systems more challenging than it should be. Moreover, since receiving systems have little control over how quickly new requests or data arrive, they are readily overloaded. They are subject to the whims of the programs making requests if there is no request buffer. Also, communication cannot be relived. As a result, reconstructing a system's state is challenging.

By serving as a centralized communication hub (sometimes compared to a central nervous system), which allows systems to send and receive data without being aware of one another, Kafka makes the communication between systems smoother. It uses the publish/subscribe communication pattern (also known as pub/sub), which results in a significantly more straightforward communication model, as shown in Figure 2.5 (Shapira et al., 2022)

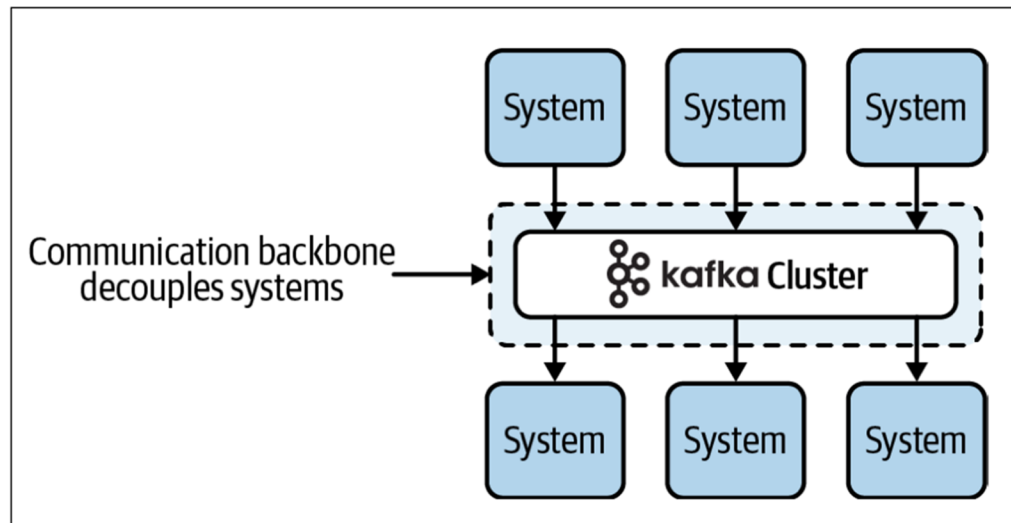


Figure 2.5 Kafka communication model

Taken from Shapira et al., (2022)

2.1.1.6. Kafka's Architecture Reflecting Publish/Subscribe Feature

The major elements of Kafka's communication model can be identified by adding more specifics to the previous diagram, as illustrated in Figure 2.6. Utilizing Kafka's architecture, which prioritizes the flow of data streams that are easily accessible for multiple processes to read from and write to, offers several advantages. For instance, due to the capability of systems to independently produce and consume data, there is increased decoupling, making systems simpler to maintain. Additionally, with asynchronous communication, delivery assurances are more robust. In the event of a consumer going offline, it can resume its operations upon reconnection, or in the case of multiple consumers, the workload is distributed among other members of the consumer group (Shapira et al., 2022).

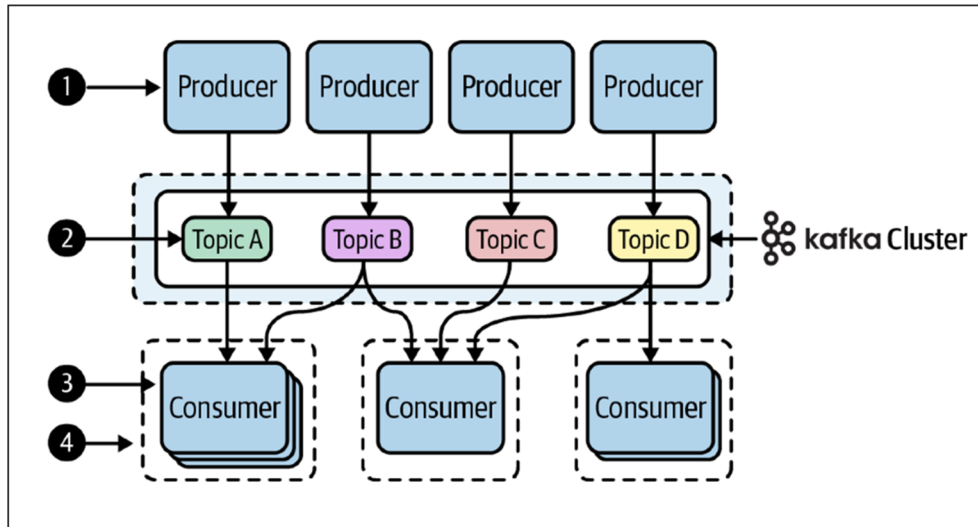


Figure 2.6 Kafka's architecture

Taken from Shapira et al., (2022)

2.1.1.7. Topics & Partitions

In Kafka, topics serve the purpose of grouping messages, resembling database tables or folders in a filesystem. Moreover, topics are subdivided into partitions, which can be similar to individual logs, adhering to the "commit log" definition. Each partition functions as a single log, receiving append-only messages that are sequentially read from start to finish. It is important to note that message ordering is guaranteed only within a single partition, not across the entire topic, as a topic may consist of multiple partitions. Figure 2.7 illustrates a topic with four partitions, each accepting appended writes. Kafka utilizes partitions for scalability and redundancy. Horizontal scaling is achieved by distributing a single topic across multiple servers, significantly enhancing performance beyond the capacity of a single server, as each partition can reside on a different server. Additionally, partitions can be duplicated, ensuring that in the event of a server failure, multiple servers retain a copy of the same partition (Shapira et al., 2022).

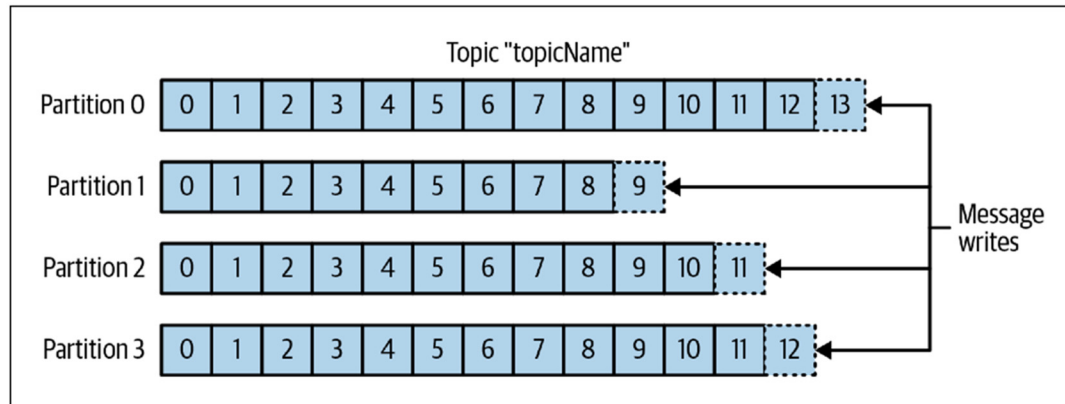


Figure 2.7 Topics and partitions in Kafka

Taken from Shapira et al., (2022)

2.1.1.8. Messages in Kafka

In Kafka, a message is the designated term for the data structure employed. For those with a database background, this can be similar to a row or record. Importantly, Kafka is indifferent to the format or meaning of the data within a message; Kafka perceives it solely as an array of bytes. Introducing a key, which is an optional metadata type within a message, is another aspect. Similar to the message, the key is a byte array and holds no intrinsic significance for Kafka. Keys come into play when a more controlled method is needed for writing messages to partitions. A common approach involves employing a consistent hash for the keys. This ensures that the same key consistently directs messages to the same partition, provided that the partition count remains unchanged. Figure 2.8 illustrates the key-value pairs of messages within Kafka (Shapira et al., 2022).

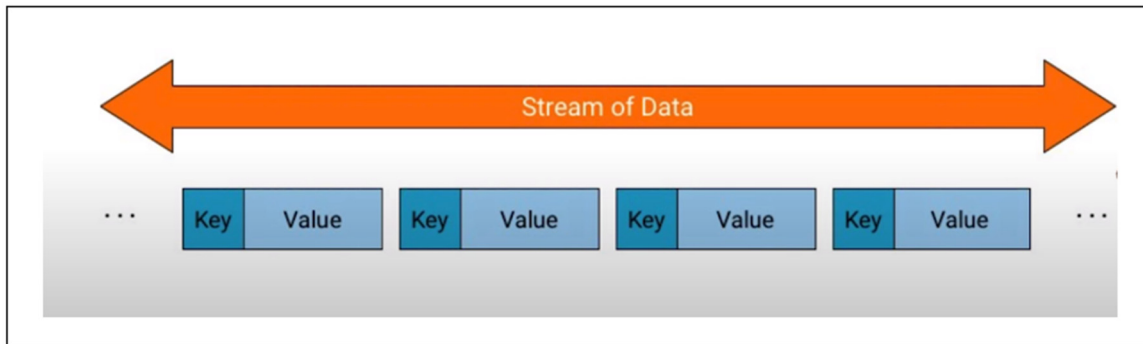


Figure 2.8 Messages in Kafka

Taken from Shapira et al., (2022)

2.1.1.9. Batches in Kafka

Kafka employs a batch approach for writing messages to enhance efficiency. As it can be seen in Figure 2.9, a batch refers to a collective set of messages produced to the same topic and partition. Aggregating messages into batches reduces the overhead associated with sending each message individually across the network. This introduces a trade-off between throughput and latency: larger batches enable processing more messages within a specific timeframe, but the time taken for a single message to propagate increases. Moreover, batches are often compressed to optimize data transfer and storage, albeit at a slight increase in computational overhead (Shapira et al., 2022).

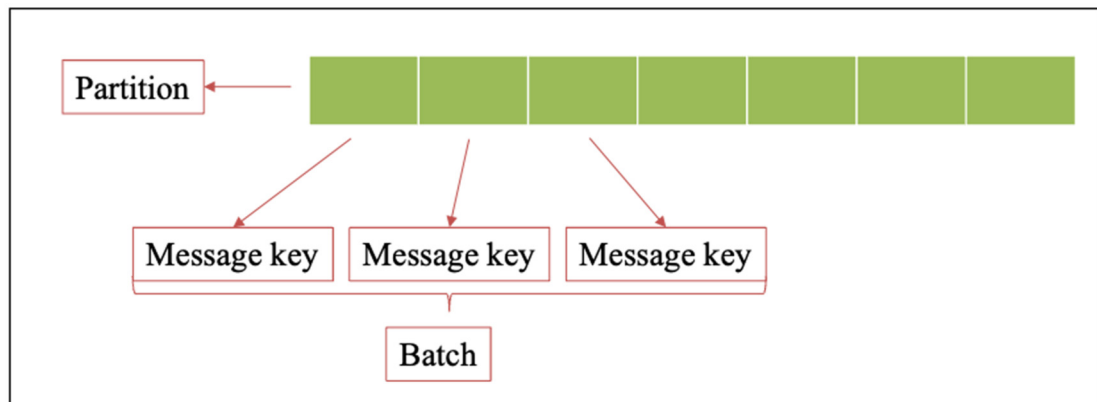


Figure 2.9 Batches in Kafka messages

2.1.1.10. Producers in Kafka

An application might have diverse requirements for writing messages to Kafka, such as monitoring user activity for auditing or analysis, saving log messages, storing log messages from smart appliances, asynchronous communication with other applications, buffering data before database writes, and various other purposes. While the producer API itself is relatively simple, it executes additional tasks when transmitting data. Figure 2.10 illustrates the fundamental steps involved in sending data to Kafka.

Initiating the process of sending messages to Kafka involves creating a *ProducerRecord*, which must include the topic and value for the intended message. Other available options include specifying a key, partition, date, and/or a set of headers. If a partition is not explicitly chosen, the data is directed to a partitioner. The partitioner, usually based on the key of the *ProducerRecord*, will autonomously select a partition. Once a partition is determined, the producer becomes aware of the destination topic and partition for the record. Subsequently, the record is included in a batch that will be sent to the same topic and partition.

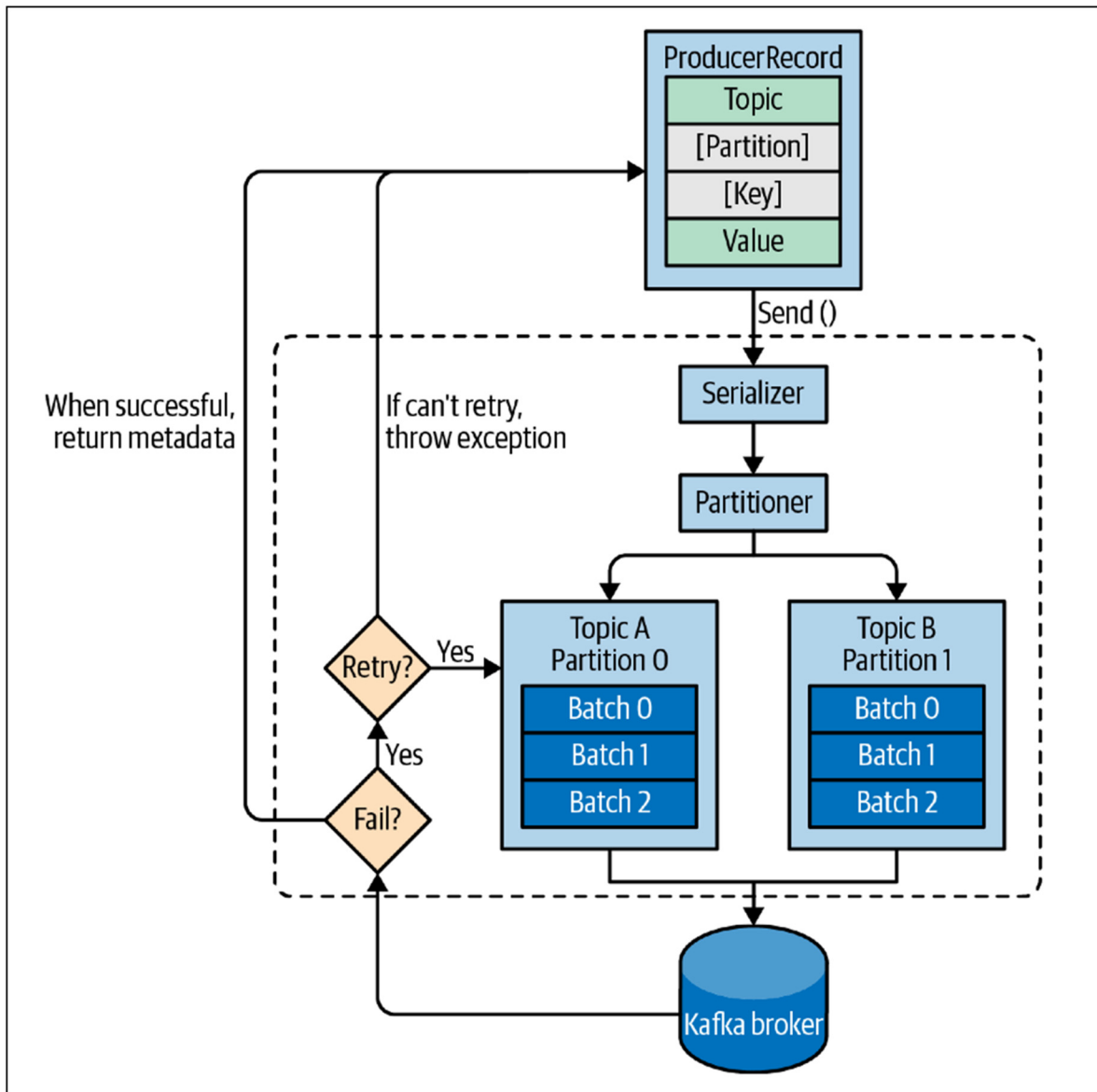


Figure 2.10 An overview of Kafka producer components

Taken from Shapira et al., (2022)

A separate thread is responsible for dispatching batches of records to the relevant Kafka brokers. Brokers, which will be detailed in subsequent sections, respond to these communications. Upon successful writing of the messages to Kafka, the broker returns a *RecordMetadata* object containing information about the topic, partition, and offset of the record within the partition. In cases where the messages cannot be written, the broker returns

an error. Following an error, the producer may make several additional attempts to send the message before ultimately giving up and returning an error (Shapira et al., 2022).

2.1.1.11. Consumers in Kafka

Applications requiring data retrieval from Kafka use a `KafkaConsumer` to subscribe to topics and receive messages from these subscribed topics. The process of reading data from Kafka involves certain unique concepts and ideas, distinguishing it from the approach used in other messaging systems. Kafka consumers are typically organized into consumer groups. Within a consumer group, each consumer subscribed to a specific topic receives messages from a unique subset of the partitions associated with that topic, as shown in Figure 2.11 (Shapira et al., 2022).

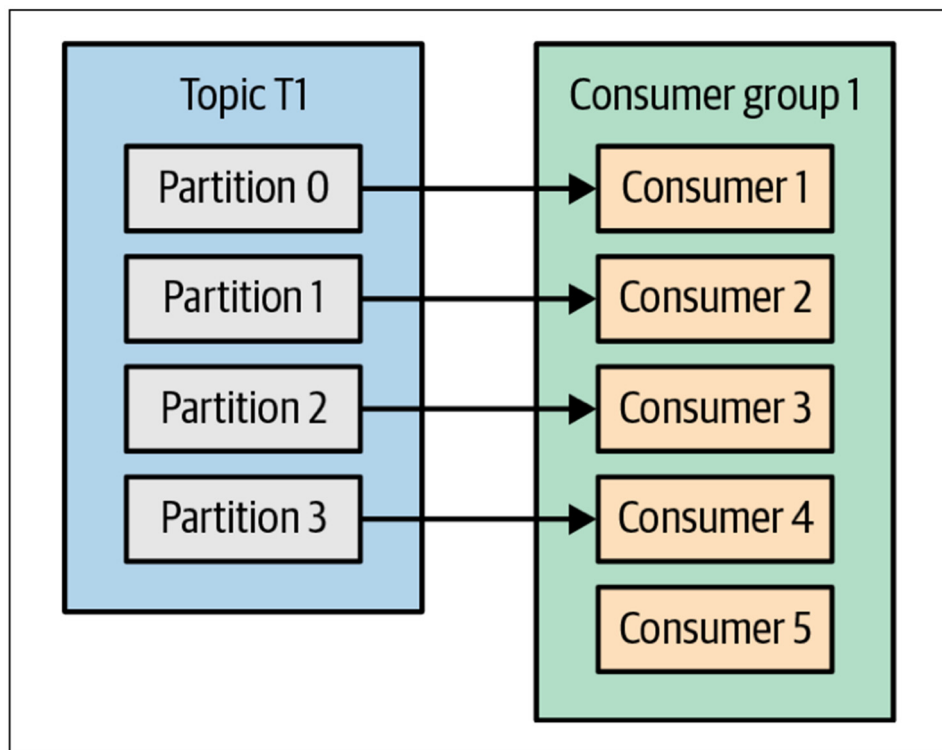


Figure 2.11 Consumer group consuming from topics

Taken from Shapira et al., (2022)

2.1.1.12. Brokers & Clusters in Kafka

A broker represents an individual Kafka server responsible for gathering messages from producers, assigning offsets to them, and storing the messages on disk storage. Moreover, it serves consumer requests by responding to partition fetch requests and delivering the published messages. Depending on the hardware's performance attributes, a single broker has the capability to efficiently handle millions of messages per second and manage thousands of partitions.

Kafka brokers are designed to operate within a cluster, where one broker assumes the role of the cluster controller, automatically selected from the active members. The controller is responsible for managing administrative functions, including partition allocation to brokers and monitoring for broker failures. The broker in the cluster that controls a specific partition is termed the partition leader. Additional brokers assigned to a replicated partition are known as followers of the partition, as illustrated in Figure 2.12. Replication ensures redundancy of messages within the partition, allowing a follower to assume the leader's role in the event of a broker failure. Consumers have the option to fetch data from either the leader or one of the followers, but all producers must connect to the leader when publishing messages.

Retention, encompassing the persistent, prolonged storage of messages, constitutes an important aspect of Apache Kafka. By default, Kafka brokers are configured to retain topic messages for a specified duration (e.g., 7 days) or until a predetermined partition size is reached (e.g., 1 GB). Messages are automatically expired and deleted once these specified limits are met. Consequently, the retention setting dictates the minimum data that must always remain accessible (Shapira et al., 2022).

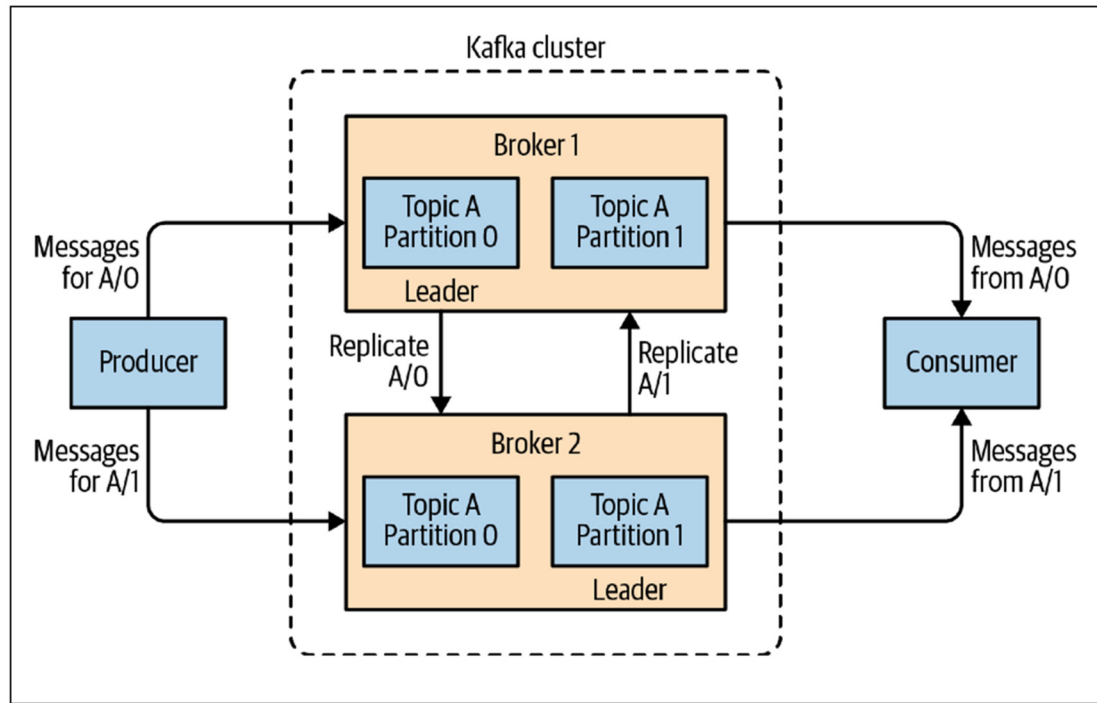


Figure 2.12 Brokers and clusters in Kafka

Taken from Shapira et al., (2022)

2.1.2. Analytical Database Software (Apache Druid)

Apache Druid plays a significant role in enabling real-time data processing, augmenting the framework's capabilities for immediate analytics. The choice between a data warehouse and a real-time analytics database in data analytics is contingent upon the specific task at hand. Noteworthy as a data warehouse, Snowflake excels in reporting and data consolidation. Its cost-effective architecture emphasizes economical storage and supports concurrency through clustering, making it well-suited for periodic reports and insights. Conversely, Apache Druid belongs to the realm of real-time analytics databases, purposefully designed for interactive data interactions with sub-second response times, high concurrency, and adaptability. Druid's dynamic schema approach accommodates evolving data, and its emphasis on low-latency querying and real-time analytics positions it as a robust solution for rapidly processing massive volumes of streaming data. Refer to Table 2.2 for a detailed comparison in this context.

Furthermore, we employ Apache Druid to visualize statistics across diverse scenarios, as it offers a range of statistical insights.

Table 2.2 Summary of Apache Druid advantages vs data warehouse

Data warehouse	Apache Druid as a real-time analytic tool
Provides reporting without considering performance	Provides reporting while performance is important (interactive data conversation)
Focuses on low cost	Focuses on low latency (fast)
Cluster concurrency	High concurrency
Inflexible, locking (fixed schema)	Flexible in heterogenous data types (fixed schema, flexible schema)

2.2. Security and Privacy

Numerous investigations and services have been undertaken to explore the recent developments in IoT security (Makhdoom, Abolhasan, Lipman, Liu, & Ni, 2019). The integrity of the system administrator is purportedly at risk due to various issues, including blocking, spoofed breakouts, and other unwarranted intrusions, as asserted by the researchers. As depicted in Figure 2.13, research on IoT security has progressed rapidly, benefiting from a variety of simulation models and modelers. It is reasonable to affirm that severe consequences may arise in the event of malfunctions in IoT devices (Beltran et al., 2022).

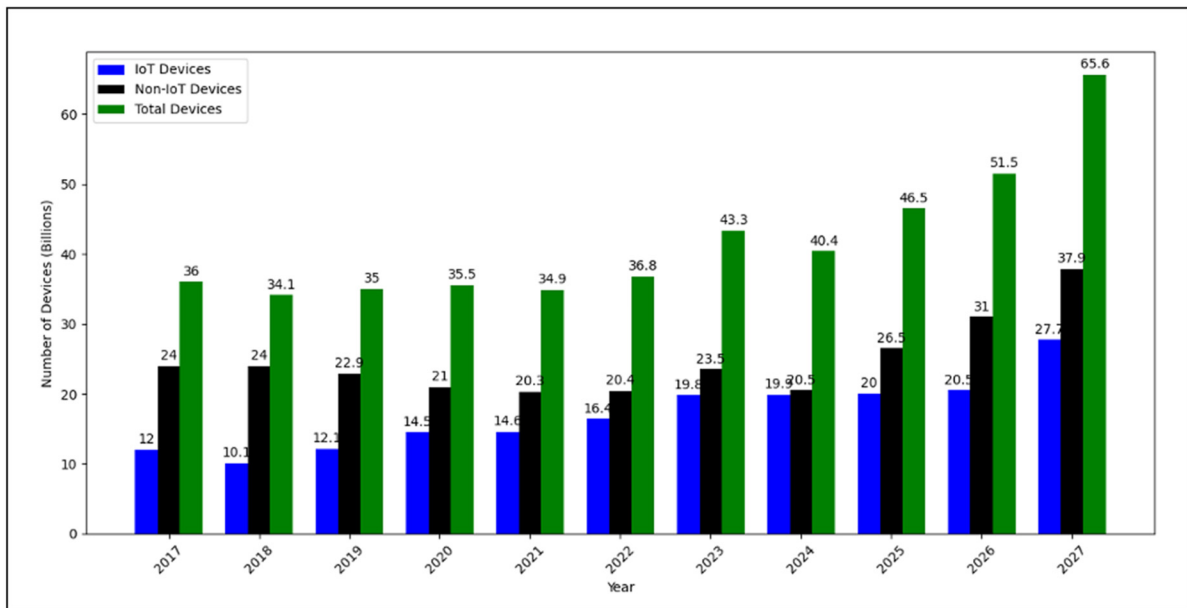


Figure 2.13 Number of IoT devices

Adapted from Beltran et al., (2022)

Privacy, stands out as a critical concern that enables consumers to feel secure and confident when utilizing IoT solutions. Consequently, asserting permission and authentication within a secure network is imperative to validate interactions between trustworthy organizations (Pei, Tschofenig, Atyeo, Cook, & Yoo, 2016). Actually, the widespread deployment of IoT devices, possessing the ability to collect and disseminate data nearly everywhere, intensifies issues about privacy. This is particularly significant considering the ease with which private data can be globally accessed without specific protective measures in position.

2.3. Network Access Technologies

In recent decades, there has been significant progress and improvement in wireless communication technologies and equipment. Figure 2.14 illustrates the evolution of wireless network access technology, progressing from 1G systems to the current 4G systems. The initial wireless networks, emerging in the 1980s, were voice-centric cellular systems known as

Advanced Mobile Phone Services (AMPS) and IS-95, representing 1G and 2G networks. Upgrades were implemented in 1G and 2G networks to accommodate data-centric devices with low-to-medium data speeds, such as EDGE (e.g., 9.6 Kbps). The establishment of Wideband Code Division Multiple Access (WCDMA), a 3G wireless network, was facilitated by the formation of 3GPP. Introduced in 1999, WCDMA and CDMA2000, two 3G systems, aimed to provide diverse voice, video, and data services. Releases 99 through 7 of the 3GPP's WCDMA standard, released during this period, enabled genuine mobile broadband access with speeds reaching several megabits per second. The advent of Long Term Evolution (LTE), marking the beginning of 3GPP's 4G initiative, commenced with Release 8 in 2009. LTE underwent subsequent updates, ranging from Release 8 to Release 15. The latter release signaled the initiation of standardization efforts for 5G.

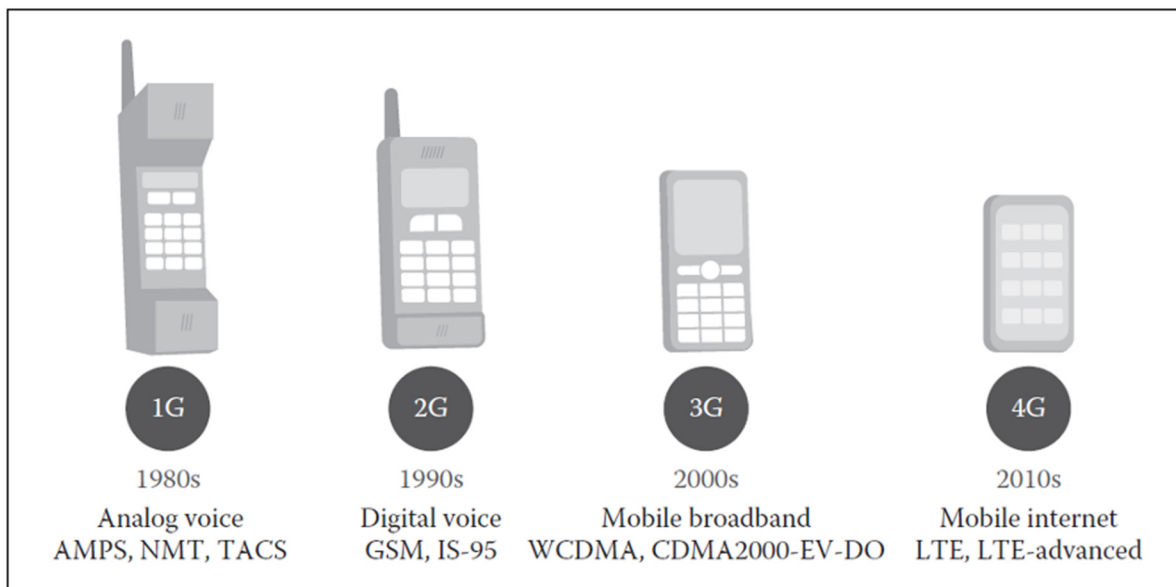


Figure 2.14 Wireless network access technologies

Taken from Fattah, (2018)

It is important to note that the network layer assumes the responsibility of processing data transmitted by the perception layer. Furthermore, it is tasked with forwarding data to the application layer (cloud) using a diverse range of network technologies, such as wireless and

wired networks as well as Local Area Networks (LAN). FTTx, 3G, 4G, LTE for Machine Type Communications (LTE-M), 5G, WiFi, Bluetooth, Zigbee, NarrowBand IoT (NB-IoT), Enhanced Coverage Global System for Mobile Communication (GSM) for IoT (EC-GSM-IoT), and other communication mediums. Given the substantial data volumes in Massive IoT, the network must be capable of efficiently handling this extensive data load. Consequently, it is imperative to provide a reliable middleware for the storage and analysis of such massive amounts of data (Laroui et al., 2021). Table 2.3 presents a comparison of the key characteristics of significant technologies in MIoT.

2.3.1. LTE as a Network Access Technology

LTE-M, a candidate for facilitating M2M communications within Long Term Evolution (LTE) cellular networks, emerges as a solution for MIoT. In the initial introduction of M2M devices, such as sensors and smart meters, LTE as a platform lacked optimization for low data rate requirements. LTE is designed for swift data transfer at high rates with minimal latency. LTE-MTC, also referred to as LTE-M, originated from the EXALTED Project, introducing a scalable end-to-end network architecture to extend LTE for machine devices and cater to M2M communications for low-end devices (Dawaliby et al., 2016). Several cellular service providers and companies, such as Nokia, Ericsson, and Qualcomm, have introduced LTE-M as a feasible solution to improve LTE for the Internet of Things (Gündoğan, Amsüss, Schmidt, & Wählich, 2020).

2.3.2. NB-IoT as a Network Access Technology

NarrowBand-IoT, introduced in 3GPP Release 13 as a Frequency Division Duplex (FDD) only radio access technology, offers expansive coverage for IoT applications. It is designed to fulfill IoT requirements, providing flexibility in deployment, high connection density, extended coverage, and prolonged battery life by leveraging LTE integration extensively. With a lower

device complexity compared to LTE-MTC (LTE-M) modules, NB-IoT is well-suited for mMTC applications.

Table 2.3 Main characteristic of proposed technologies

Adapted from Laroui et al., (2021)

Characteristics	LTE-M	EC-GSM-IoT	NB-IoT
Frequency bands	LTE 1-5, 7, 8, 11-14, 18-21, 25-28, 31, 39-41, 66, 71-75, 85, 87, 88	GSM/GPRS (850, 900, 1800, 1900 MHz)	LTE 1-5, 8, 11-14, 17-20, 25, 26, 28, 31, 66, 70-74, 85
Coverage	155, 7 dB	164 dB (33 dBm) 154 dB (23 dBm)	164 dB
Bandwidth	1.08 MHz	200 kHz	180 kHz
Downlink (DL) and uplink (UL) throughput	1 Mbps	489.9 kbps-GMSK, 153.2 kbps-8PSK	250 kbps
DL Modulation and MAC	16 QAM & OFDMA	GMSK, 8PSK & TDMA/FDMA	OFDMA
UL modulation and MAC	16 QAM & SC-FDMA		SC-FDMA
Duplex	HD, FD TDD and FDD	HD FDD	HD FDD
Power saving	Power saving mode (PSM) & Extended Discontinuous Reception (eDRx)		
Power classes	20 dBm, 23 dBm	23 dBm, 33 dBm	20 dBm, 23 dBm

Utilizing licensed airwaves ensures its safety, reliability, and superior Quality of Service (QoS). Many service providers have already endorsed NB-IoT as their preferred access

method, placing it within the Low Power Wide Area (LPWA) technology category and making it suitable for IoT applications with limited mobility (Annamalai, Bapat, & Das, 2018).

NB-IoT offers added advantages by utilizing shared spectrum within existing LTE networks, operating on a reframed 200 KHz carrier from GSM. Physically, it utilizes 12 subcarriers of a single Physical Resource Block (PRB) with a 180 KHz bandwidth in both the uplink and downlink. The reduced bandwidth contributes to increased device durability and significantly reduced complexity. To align with MIoT requirements, constraints on Transport Block Size (TBS) and modulation strategy are implemented, reducing the peak data rate and further enhancing device simplicity. NB-IoT provides three deployment options: 1) Stand Alone, utilizing a single 200 KHz GSM carrier without LTE resources; 2) In Band, allowing flexible LTE capacity trading for NB-IoT applications without additional spectrum; and 3) Guard Band, utilizing unused frequencies between LTE carrier guard bands, which is highly distinctive and doesn't utilize LTE resources. Table 2.4 summarizes key enhancements introduced in NB-IoT over LTE (Annamalai et al., 2018).

Table 2.4 NB-IoT Enhancements over LTE

Key Enhancement Items	IoT Impact
200 KHz Narrow Band Carrier Bandwidth	<ul style="list-style-type: none"> □ Helps to improve the network capacity to accommodate more number of devices and can be scaled up by adding multiple carriers □ Reduces noise level due to narrow band operation and helps in enhancing the coverage by 20dB □ Transceiver design becomes less complex and inexpensive to bring down the cost and power consumption
Single Hybrid Automatic Repeat Request (HARQ) Process	<ul style="list-style-type: none"> □ Implementation is simplified to bring down the complexity and memory footprint

Key Enhancement Items	IoT Impact
Modulation Support Limited to QPSK	<input type="checkbox"/> Makes the system very robust in poor signal conditions <input type="checkbox"/> Helps to extend the coverage for better reachability of devices

Table 2.4 NB-IoT Enhancements over LTE

Key Enhancement Items	IoT Impact
Half Duplex Transmission and FDD only Support	<input type="checkbox"/> Brings down the complexity, size, components and its associated cost of IoT devices for massive deployment
Maximum Transport Block Size (TBS) is capped to 680 bits in Downlink and 1000 bits in Uplink	<input type="checkbox"/> Makes them suitable for small data transmission
Single PRB Operation and Single-Layer Transmission	<input type="checkbox"/> Reduces the throughput to facilitate small data transmission <input type="checkbox"/> Limits processing and memory requirements to minimize size and cost
Idle Mode eDRX up to 3 hours and Connected Mode eDRX up to 1024 Secs	<input type="checkbox"/> Achieves long years of battery life for unattended services
Single-Tone (15 KHZ or 3.75 KHZ) or Multi-Tone (n*15KHz, n up to 12) Narrow Band Physical Uplink Channel	<input type="checkbox"/> Enables multiplexing of massive number of devices in the available bandwidth to increase the connection density <input type="checkbox"/> Minimizes PAPR resulting in lesser power consumption leading to long battery life
No Turbo Coding in Downlink	<input type="checkbox"/> Minimizes the device receiver complexity that further reduces size, power consumption, cost, etc

Lesser than a PRB Bandwidth Allocation in Uplink	<input type="checkbox"/> Facilitates lesser peak throughput to accommodate large number of devices to enhance connection density
--	--

While the NB-IoT OSI reference model, also known as the protocol stack, is relatively recent, it stands out for its capacity to connect with over five billion devices through the 5G platform (Gbadamosi, Hancke, & Abu-Mahfouz, 2020). The NB-IoT layer design encompasses the data plane and the control plane. Unlike the control plane, which manages protocols governing radio-access carriers and the link between the network and the user equipment (UE), the data plane outlines the flow of user data between two nodes. The 3GPP has recognized the NB-IoT protocol stack as a value-added solution for cellular communication networks. Comprising six layers, namely the physical layer (PHY), media access control (MAC), radio link control (RLC), packet data convergence protocol (PDCP), radio resource control (RRC), and non-access stratum (NAS), the NB-IoT protocol stack aligns with the OSI reference model's structure developed by the International Standard Organization (ISO), except for the five upper layers (Anamuro, 2020). The difference between the OSI reference model and the NB-IoT protocol stack is illustrated in Figure 2.15 (Wang & Ma, 2019b).

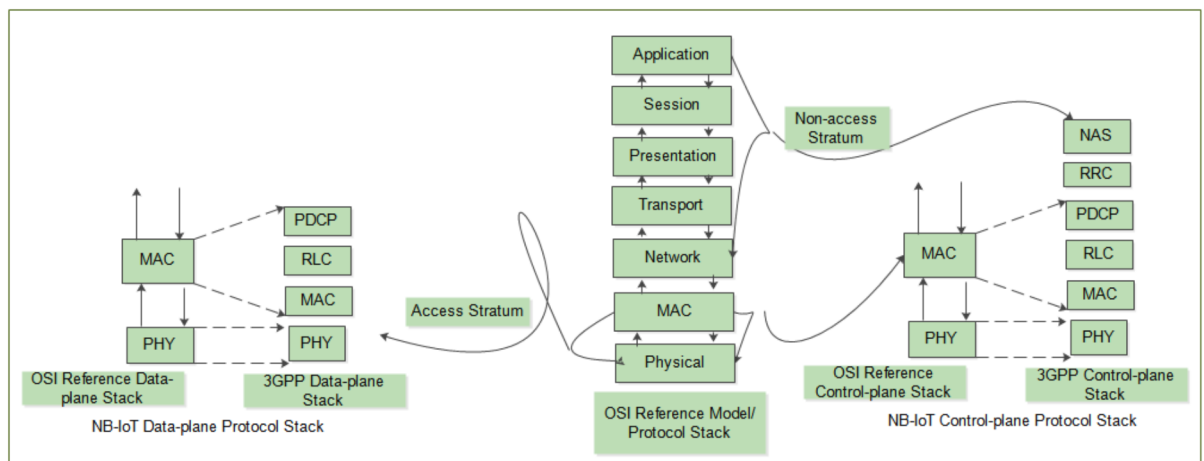


Figure 2.15 The NB-IoT Protocol Stack Architecture vs. OSI Reference Model

Taken from Gbadamosi et al., (2020)

2.3.3. 5G as a Network Access Technology

The 5G system is expected to provide at least three access technology types, each specifically designed for different service types, including Enhanced Mobile Broadband (eMBB), Ultra-Reliable Low Latency Communications (URLLC), and massive Machine Type Communications (mMTC). Figure 2.16 illustrates specific challenges in these service sectors related to crucial access technology system design criteria. The initial iteration of 5G in Release 15 New Radio (NR) specifications by 3GPP incorporated fundamental changes to frame numerology to potentially address other IoT requirements, such as MIoT. While this version provides a platform capable of handling IoT services related to URLLC or mMTC, the challenge remains in developing real-world IoT systems that precisely meet their specifications. All envisioned smart and connected applications are categorized into three service verticals for 5G (Wang & Ma, 2019).

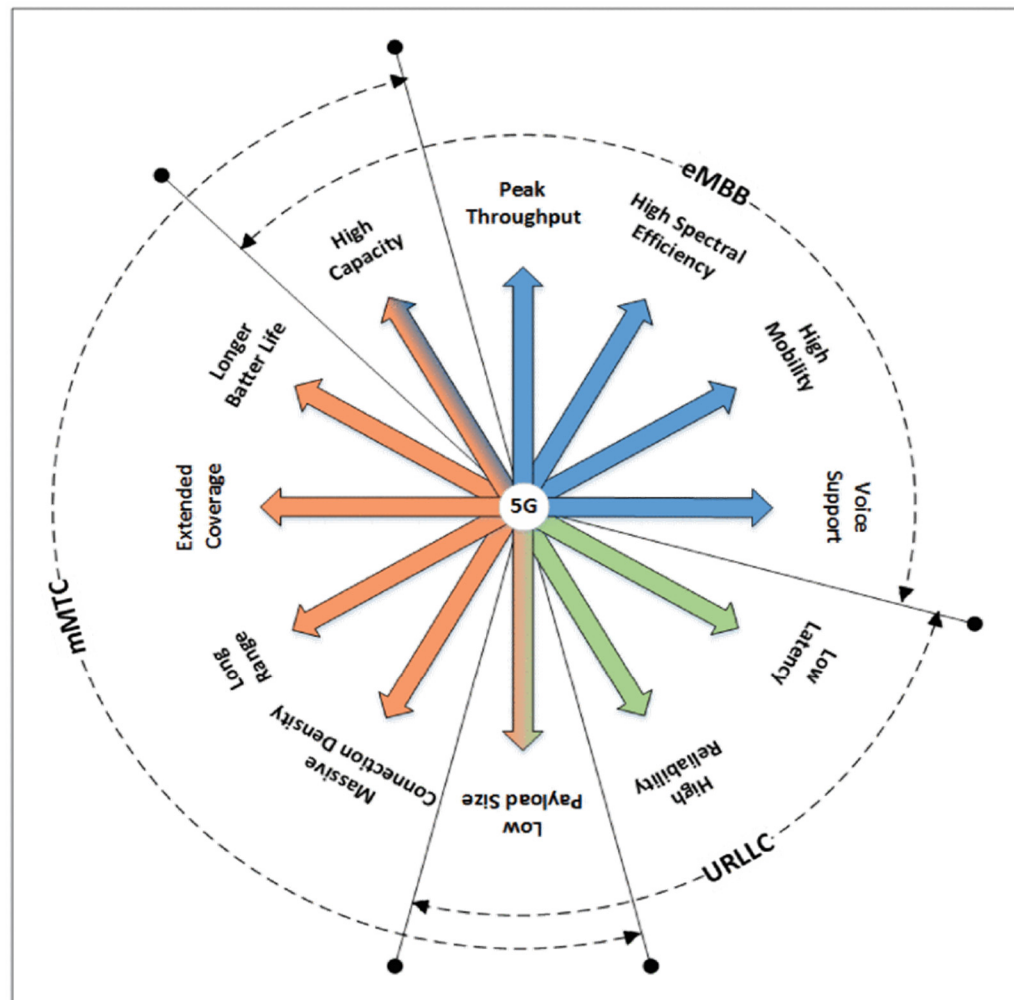


Figure 2.16 5G three service sectors

Taken from Annamalai et al., (2018)

- eMBB: Extremely high data rates, such as those required for applications like virtual reality (VR), augmented reality (AR), live streaming, high definition (HD)/4K video download, etc., should be supported by eMBB with guaranteed quality of service (QoS).
- URLLC: For time-sensitive and control applications such as industrial automation, Vehicle-to-X (V2X) communications, and other mission-critical ones, high reliability, strict latency specifications, and guaranteed availability are essential.

- One of the main goals of mMTC is to connect billions of terminal devices for years of unattended operation. Other significant criteria include higher connection density, energy efficiency, cost reduction, and coverage improvement. Even if some of these needs have already been met by existing technologies, there is still much work to be done in the next generation of technologies (Abdmeziem et al., 2015).

2.4. Communication Protocols

IoT oversees each phase of message transport through seven interrelated layers (refer to Figure 2.17). Among these seven layers, the IoT application layer and transport layer are integral parts of IoT application protocols.

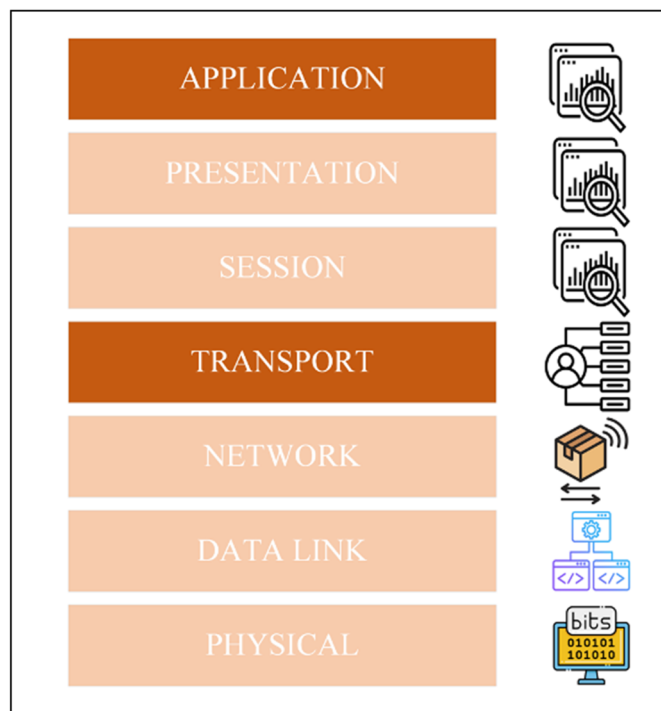


Figure 2.17 Application & transport layers in IoT

The transport layer, which employs protocols like HTTP, TCP, or UDP, is responsible for facilitating the transmission of data. On the other hand, the application layer serves as the

interface between an IoT device and the communication network it utilizes. Handling data formatting and presentation, it acts as the intermediary between the actions of the IoT device and the handoff of the data to the network (Dave, Doshi, & Arolkar, 2020). In the next sections, we will discuss various communication protocols.

2.4.1. MQTT

The MQTT protocol initially introduced in 1999 as a publish/subscribe messaging technology, was standardized by the Open Standards Open Source (OASIS) committee in 2013. Developed to provide reliable communication for embedded devices with resource and power constraints, the protocol has undergone two major releases: (a) MQTT 3.1.1 (with version 5.0 standardized in March 2019) and (b) MQTT for Sensor Networks (MQTT-SN). While MQTT-SN utilizes UDP, MQTT 3.1.1 employs TCP as the transport layer bearer (Stusek, Zeman, Masek, Sedova, & Hosek, 2020).

As depicted in Figure 2.18, the MQTT mechanism comprises four components: publish/subscribe, message, topic, and broker:

- **Publish/Subscribe:** Clients engage with brokers during the publish process, and brokers interact with clients during subscription.
- **Message:** The actual data that needs to be transmitted.
- **Topic:** Serves as the access point for both the publisher and subscriber to connect with the message.
- **Broker:** Facilitates communication between clients.

This protocol supports both the transmission of sensor-generated data to clients and the issuance of commands for output control. Key advantages of the MQTT protocol include its high throughput and robust delivery guarantee (Jaikumar, Brindha, Deepalakshmi, & Gomathi, 2020).

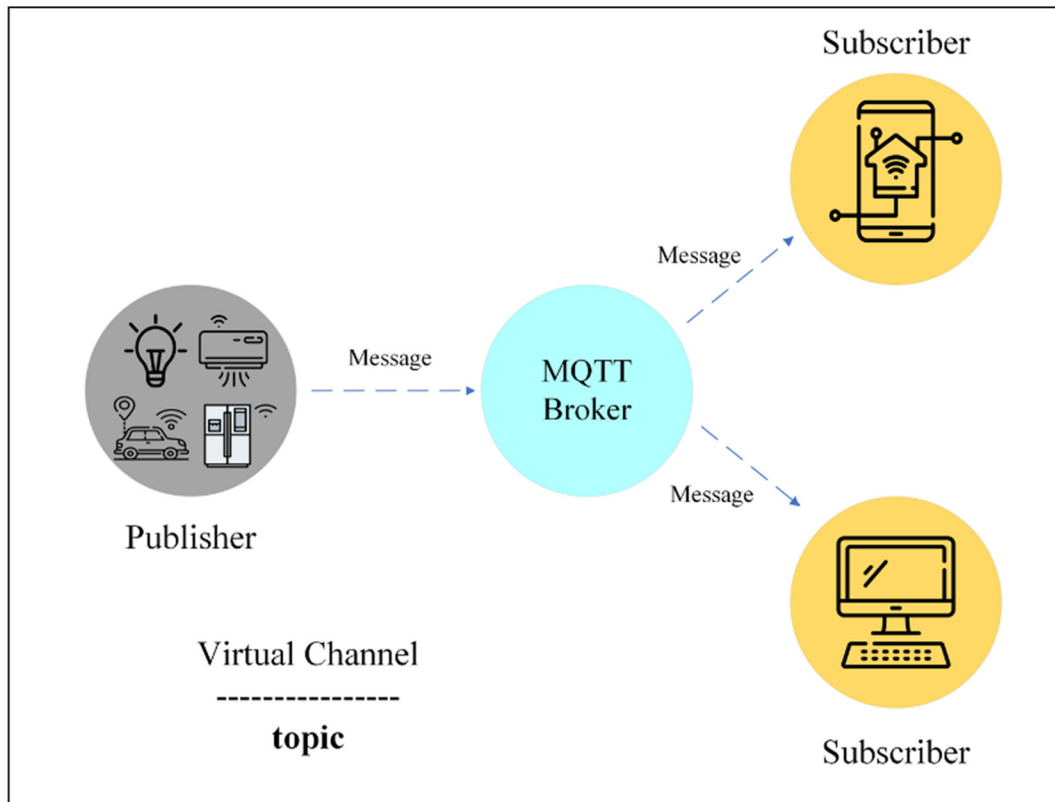


Figure 2.18 MQTT mechanism
Adapted from Jaikumar et al., (2020)

2.4.2. LwM2M

The Lightweight M2M (LwM2M) standard, established by the Open Mobile Alliance (OMA), is designed to create client-server specifications for swift deployment, specifically targeting machine-to-machine (M2M) services. It functions as the M2M counterpart to OMA device management (OMA-DM), providing efficient device management and security protocols applicable to IoT applications using the same protocol, enhancing user-friendliness. LwM2M addresses the service and administrative needs of constrained M2M devices across various carriers and transports. Notably, it incorporates a built-in binding for the widely used Constrained Application Protocol (CoAP), making it particularly attractive for IoT applications.

The LwM2M 1.0 version supported SMS and UDP. In the updated LwM2M 1.1 protocol stack depicted in Figure 2.19, additional protocols such as TCP, CIoT, and LoRaWAN are now supported. Transports like UDP, SMS, and CIoT can be employed with or without DTLS, while TCP and TLS can be used in combination. LwM2M 1.1 also integrates OSCORE, an application layer security protocol that provides proxy operations and end-to-end security independently of the underlying transport layer protocols. OSCORE can be utilized with any of the transport bindings, including UDP, SMS, and TCP, with or without DTLS or TLS.

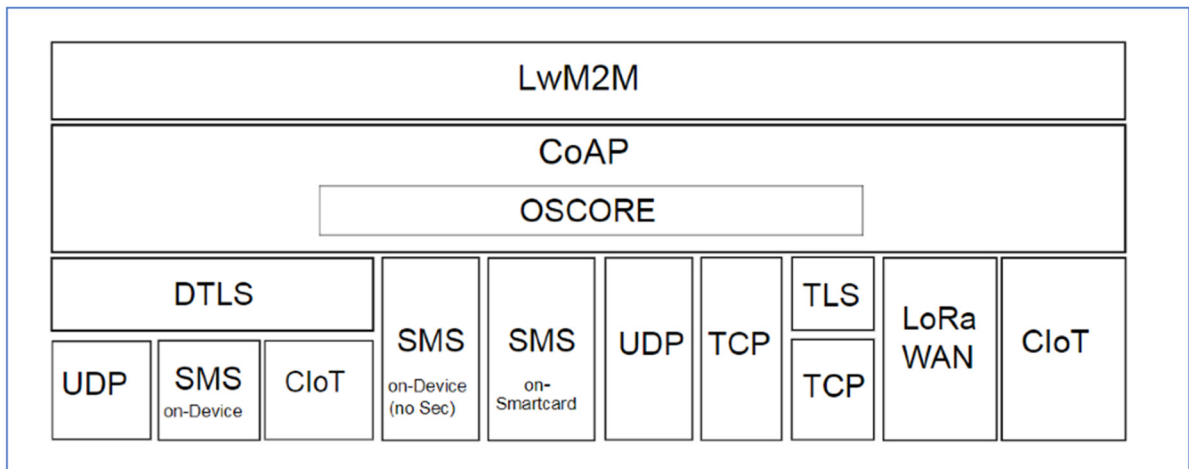


Figure 2.19 LwM2M protocol stack

CHAPTER 3

DESCRIPTION OF METHODOLOGY USED

3.1. The Proposed MIoT Solution

To meet the needs of the MIoT platform, an extensive five-layer architectural framework has been developed and implemented, as shown in Figure 3.1. This arrangement exhibits remarkable efficiency by facilitating communication for a wide variety of IoT devices, such as actuators and sensors. The emphasis in designing this architecture was on ensuring robust capabilities in data retention, processing, and secure storage, which are respectively delivered through Apache Kafka, Apache Druid, and blockchain technology.

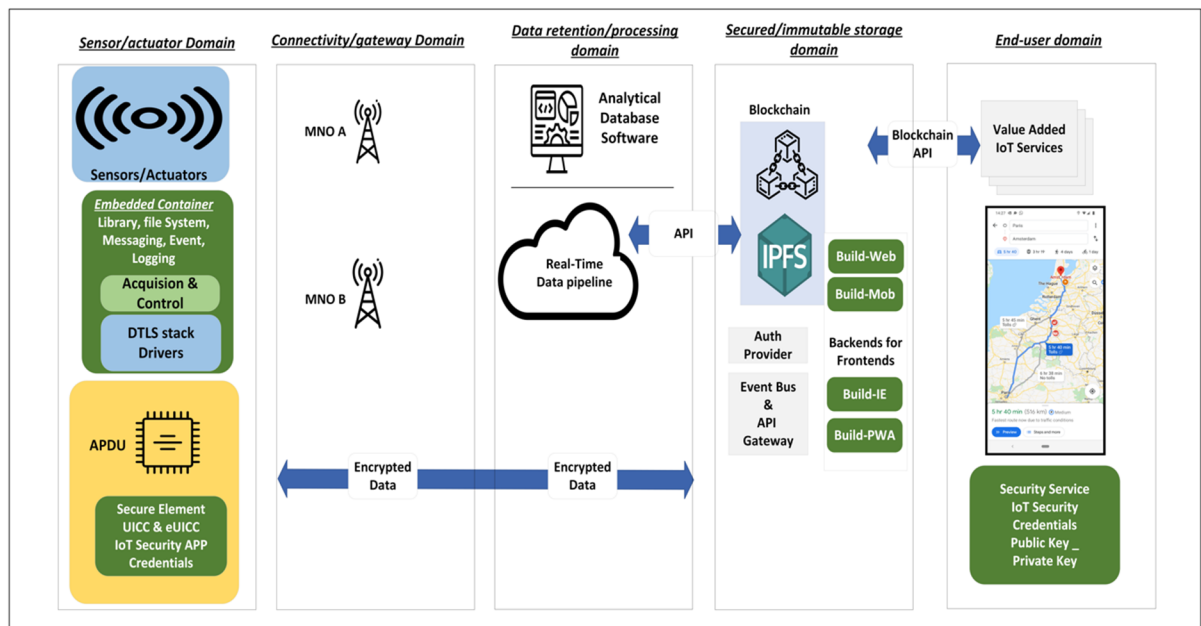


Figure 3.1 Architecture for MIoT with distinct domains

Acknowledging the importance of addressing a spectrum of challenges, we made a strategic decision to structure our model into five distinct domains. This approach was selected with the aim of optimizing our solution. The primary domain, centered on sensing and actuating

devices, serves the crucial function of data acquisition. Ensuring availability, the second domain (connectivity and gateway) utilizes dual mobile network operators (MNOs) and automatic switchover, providing a failover mechanism to uphold framework availability in case of failure. As mentioned earlier, we employ two LTE-based Software-Defined Radios (SDRs). The subsequent domain not only ensures scalability, real-time data processing, and minimal latency but also integrates data retention capabilities. Storage security measures are implemented within the fourth domain. A succinct overview of these domains and their respective functionalities is presented in Table 3.1.

Table 3.1 Brief description of the domains

Domains	Description
1st Domain—Sensor/gateway	Includes devices to collect various data, e.g., temperature, humidity, pressure, position, etc.
2nd Domain—Connectivity	Includes Pico LTE, which is an SDR to simulate MNOs in order to provide various connectivity protocol, e.g., LTE Cat 0, Cat 1, Cat M1, Cat NB1, and Cat NB2
3rd Domain—Data retention & processing	Includes Apache Kafka and Apache Druid for data retention and real-time processing
4th Domain—Secured/immutable storage	Includes IPFS to supply with storage security
5th Domain—End-user	Includes smart home application in order to manage and track IoT devices

To explain the complexities of data flow, a three-zone framework is illustrated as Figure 3.2, which includes data collection to its secure storage on the blockchain. The process commences

with sensors collecting data, followed by its transmission to gateway devices and subsequently network simulators. These network simulators, embodied by our simulated dual Software-Defined Radios (SDRs), specifically the Pico LTE functioning as a Network-in-a-Box, establish connectivity to Kafka. Operating with protocols delineated in the picture, they act as publishers or producers, facilitating data transfer to Kafka. Simultaneously, Hyperledger Fabric subscribes to specified Kafka topics via socket.io, ensuring secure access for end-users. Furthermore, Apache Druid, serving as a real-time data analytics tool, functions as another consumer in the chain.

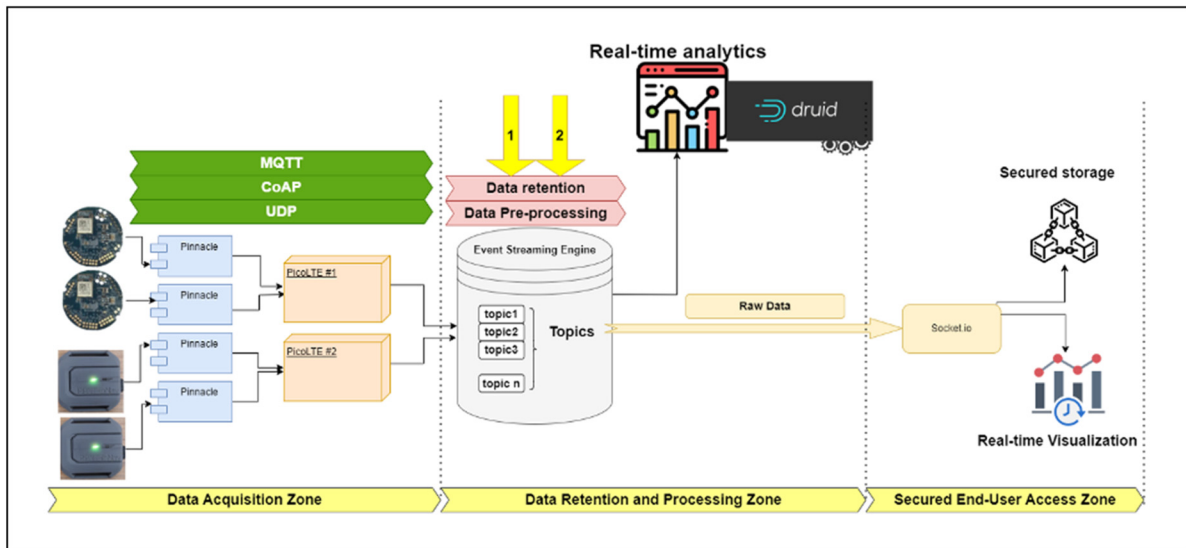


Figure 3.2 Data flow featured with three zones

3.1.1. Sensors/gateway Domain

As indicated by the domain's name and Figure 3.3, it accommodates a diverse array of devices, including sensors, actuators, and board computers, tasked with capturing environmental data. In our investigation, we employed BL654 and ibNav sensors for the data collection process. The gateway elements utilized in this study were Pinnacle™ 100 DVK devices from Laird Connectivity, a manufacturing company based in Akron, OH, USA, equipped with a SIM card for connectivity. It is noteworthy that Wi-Fi was utilized for ibNav, while Bluetooth was

employed for Pinnacle, facilitating the transmission of data from sensors to gateways. This arrangement allowed for onward transmission to the network layer, where two Software-Defined Radio (SDR) devices from Nutaq were stationed. Leveraging BL654 sensors, we effectively captured readings for temperature, humidity, and pressure. With ibNav, our data collection scope expanded to include position data in addition to the aforementioned metrics. The devices in use are illustrated in Figure 3.4.

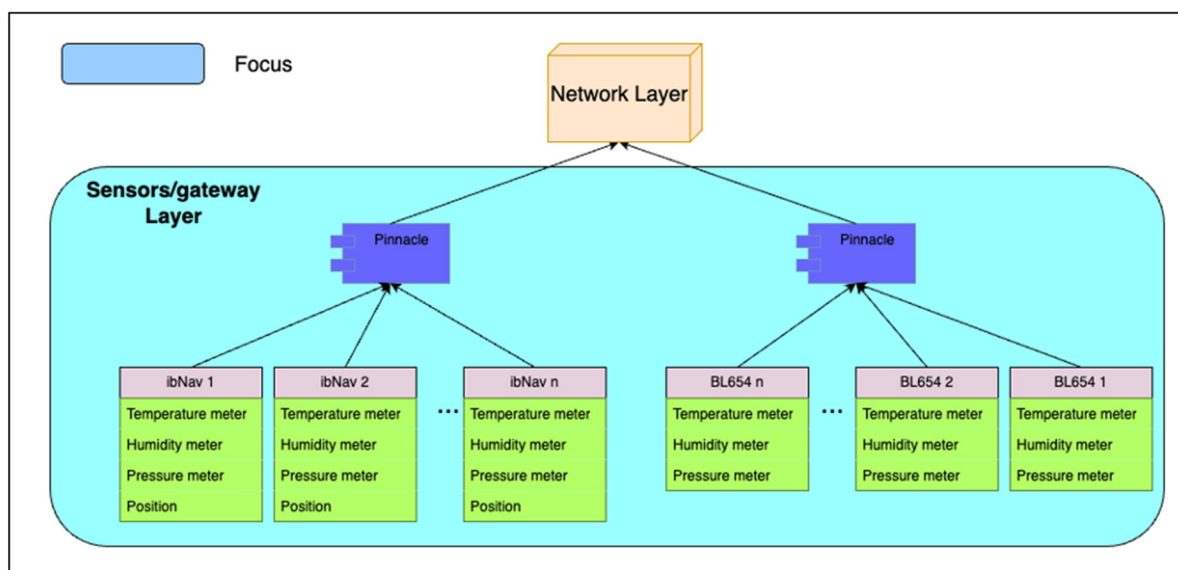


Figure 3.3 Sensor/Gateway domain

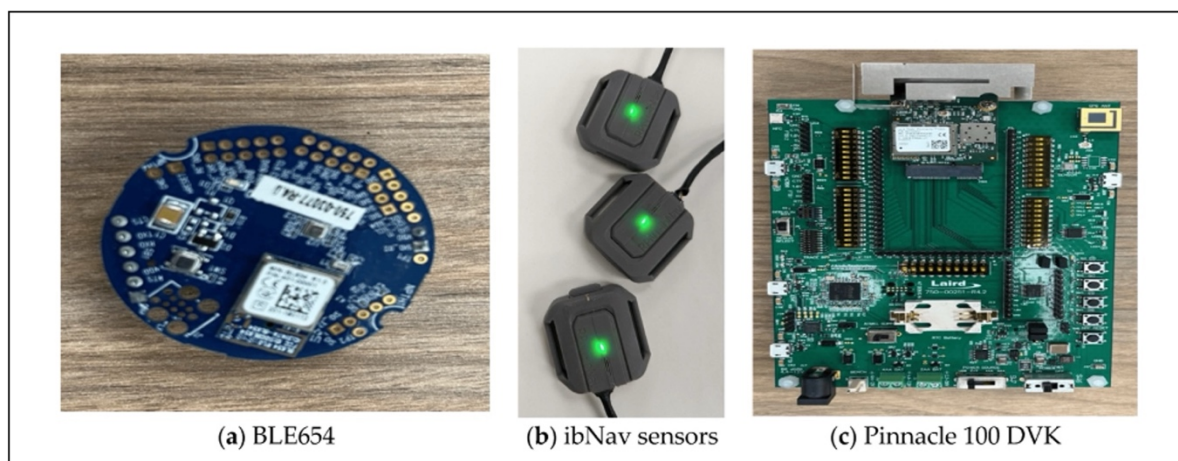


Figure 3.4 (a) BL654; (b) ibNav sensors; (c) Pinnacle 100 DVK

3.1.2. Connectivity Domain

By incorporating network simulators, we can replicate LTE-M, an efficient connectivity protocol, to realize swift, high-rate data transfer with minimal latency (Dawaliby et al., 2016). The network domain guarantees transfer of data from the sensor/gateway domain to the storage and processing domain. In our experimental setup, we integrated two Nutaq PicoLTE second-generation devices as Figure 3.5, adeptly emulating two mobile network operators (MNOs), which will be elaborated in the next section.

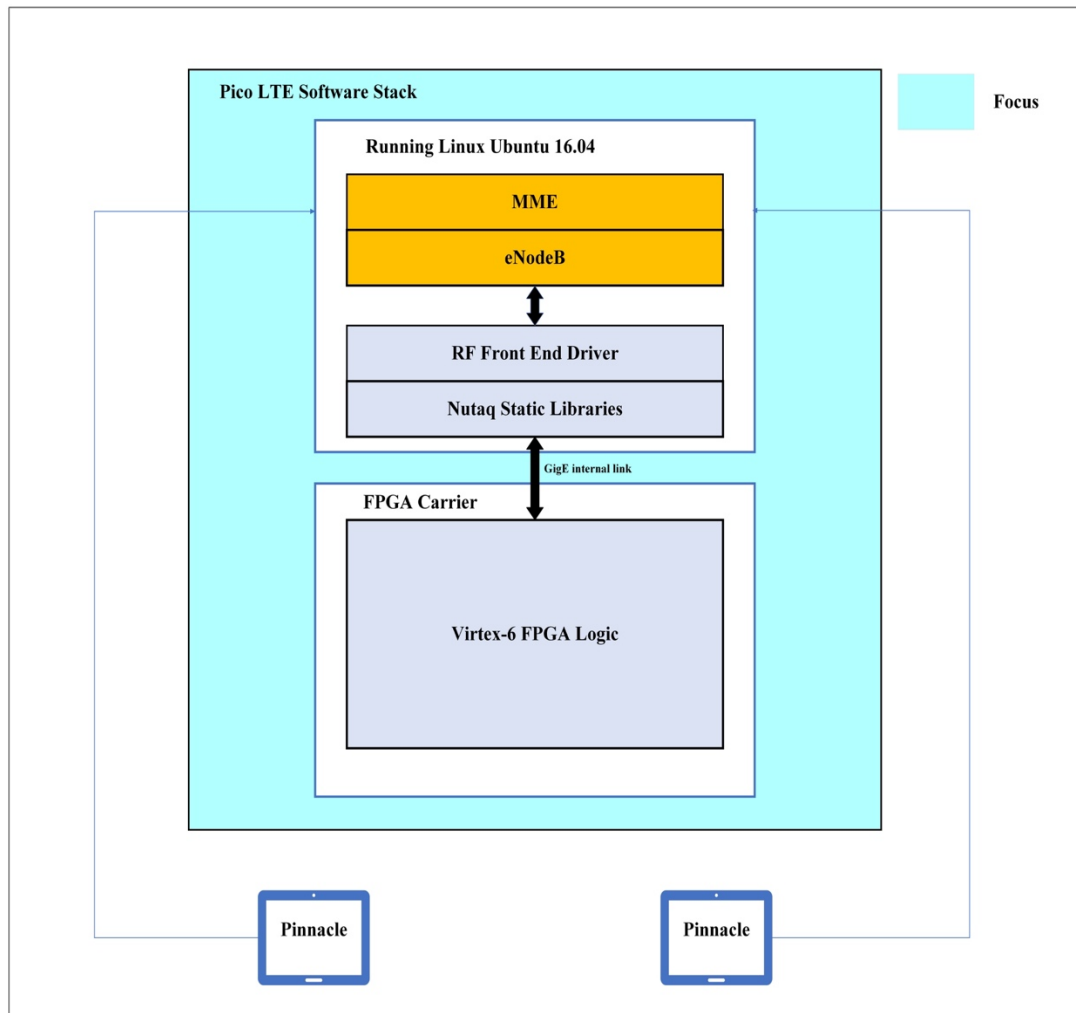


Figure 3.5 Connectivity domain

3.1.2.1. SDR for Automatic Switchover

By taking advantage of the functionalities of PicoLTE, we implemented a dynamic and resilient simulation platform, as illustrated in Figure 3.6. This allowed us to construct an LTE-centric network environment for our MIoT experiments, facilitating the evaluation of our MIoT system's performance and behavior. Nutaq's first generation of PicoLTE exhibits a range of features, establishing it as a compelling solution for various LTE requirements. With its compact and portable design, this integrated solution supports LTE Cat 0, Cat 1, Cat M1, Cat

NB1, and Cat NB2, ensuring compatibility across multiple LTE generations. Its cost-effectiveness and affordability make it a standout choice, providing an accessible option for diverse deployment scenarios. Configuration support is extended to all LTE bands, encompassing both the Frequency Division Duplex (FDD) and Time Division Duplex (TDD) modes.

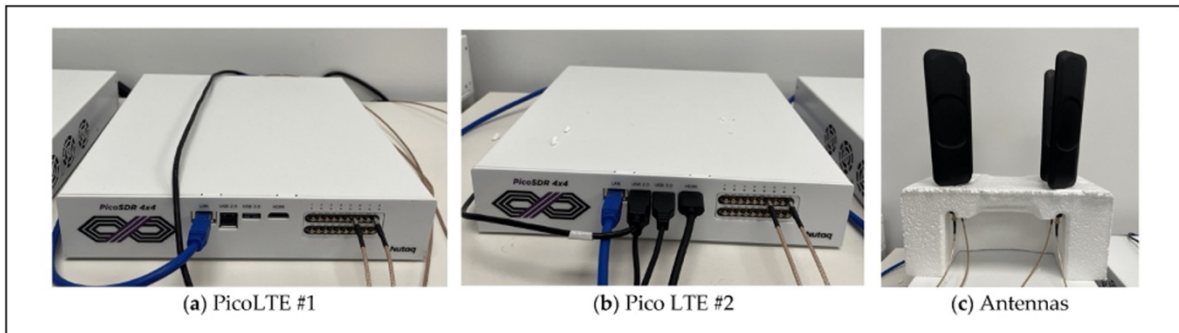


Figure 3.6 Two PicoLTEs and their antennas

3.1.3. Data Retention & Processing Domain

All IoT infrastructures and applications rely on data. They receive, analyze, manipulate, and generate output based on information. Each byte of data carries a meaningful narrative, providing insights to guide subsequent actions. The quicker this process occurs, the more adaptable and responsive organizations become. This underscores the role of data pipelines such as Kafka in data-driven IoT applications. As shown in Figure 3.7, we intend to utilize Kafka as our Event Streaming Engine in our solution. In addition to Kafka, our approach incorporates the utilization of an analytical database software named as Apache Druid to facilitate real-time analysis. This strategic integration underscores our commitment to achieving comprehensive and timely insights.

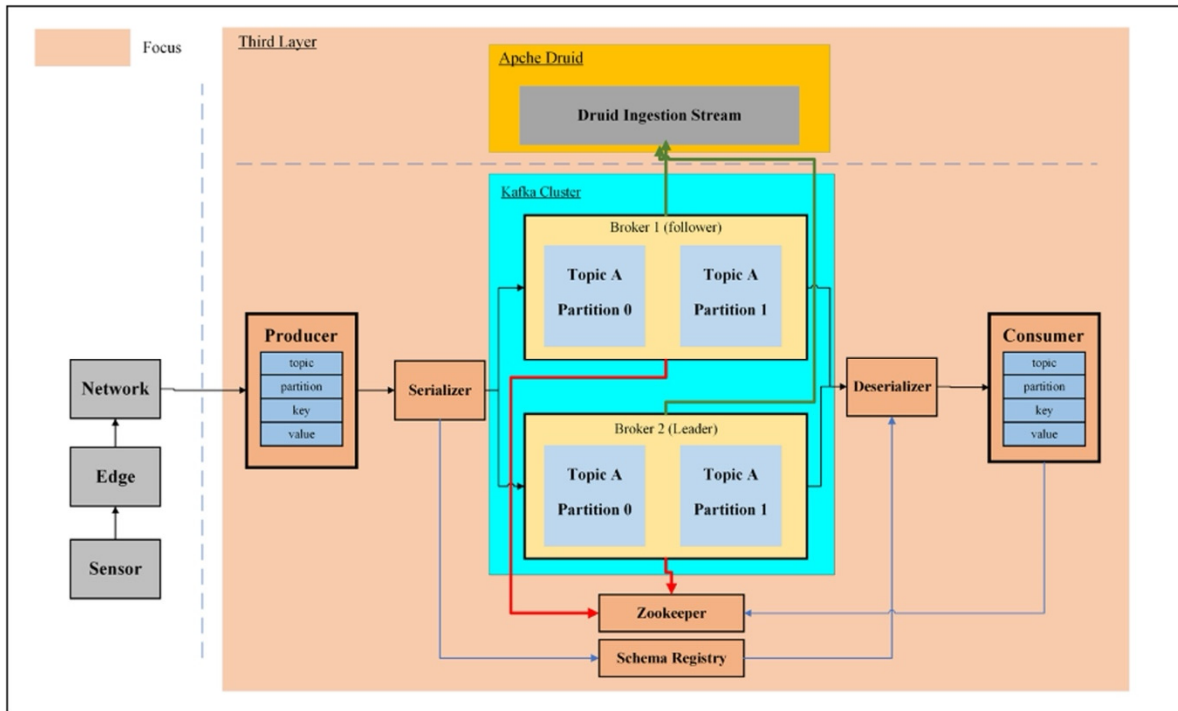


Figure 3.7 Data retention & processing domain

3.1.3.1. Kafka for Data Retention

To define such tool in a sentence, Kafka functions as a commit log for databases, ensuring the consistent reading of data by maintaining customized durations. The present status of Apache Kafka is undeniably promising, with widespread adoption by numerous global organizations. This open-source platform has swiftly become one of the most rapidly advancing projects within the open-source community (Garg, 2013). Additionally, Kafka enables the distribution of data throughout the system, enhancing resilience against failures (availability) and presenting significant opportunities for performance enhancement.

In our conceptual framework, this particular domain performs the role of preserving and conducting real-time analytics on the massive volume of data produced by IoT devices. Through the implementation of data retention policies within Apache Kafka, we gain a dual benefit of tailored data storage durations and the ability to perform diverse data manipulations.

These manipulations include aggregation, filtering, data-type transformation, field redaction, and more (Shapira et al., 2016.).

In our system, IoT devices act as producers, generating data that is initially directed to Kafka brokers. Within the broker infrastructure, the data is channeled to a queue and then persisted by assigning it to a specific partition within an assigned topic. On the consumer side, represented by the blockchain component in our project, subscriptions are established to either the broker's partition or topic, and data is asynchronously consumed from the queue. Crucial metadata, indicating the processed and pending data, is managed in Kafka Zookeeper. Any required actions are communicated back to the broker, eventually reaching the IoT devices. In this context, the blockchain functions as the producer, while IoT devices operate as consumers in our system. Latency is calculated using the following equation:

$$T_{\text{total}} = T_{DB} + W_{TDB} + T_B + T_{BC} + W_{TBC} + R_{TBC} + R_{TBD}$$

This equation incorporates various time components associated with data processing latency, encompassing the time for IoT devices to publish data (T_{DB}), the wait time in the queue for data processing at the broker (W_{TDB}), the processing time in the broker layer (T_B), the time needed to transfer the broker-processed data to the blockchain component (T_{BC}), the wait time in the queue for data processing at the blockchain component (W_{TBC}), the time required to transfer the processed data back to the broker by the blockchain component (R_{TBC}), and the time needed to transfer the information back to the IoT devices through the broker (R_{TBD}). The latency metrics observed within the queue are elaborated in Table 3.2.

3.1.3.2. Druid for Real-time Data analytic

Apache Druid serves as a powerful analytical database software designed for real-time data analytics in our MIIoT solution. This section explores the capabilities of Druid as a pivotal component within the MIIoT architecture.

Table 3.2 Broker's queue properties and description

Queue Properties	Description
Response Send	Measures the time taken by the broker to send a response to the client after receiving a request
Request Queue	Measures the time a client request spent waiting in the broker's request queue to be processed
Request Local	Measures the time taken by the broker to process a client request that was directed to the same broker
Response Remote	Measures the time taken by the broker to receive a response from another broker in the Kafka cluster
Response Queue	Measures the time a response spent waiting in the broker's response queue to be sent back to the client
Median	Represents the 50th percentile of the distribution of the response time for requests in this broker

Druid is uniquely designed to handle large volumes of streaming data, making it an ideal choice for applications demanding real-time insights and analytics. It operates on the principle of data ingestion, segmentation, and indexing, allowing for rapid querying and retrieval of information. Druid integrates with Kafka, subscribing to relevant Kafka topics, thereby establishing a robust connection between the data streaming in from MIIOT devices and the analytics engine.

As data flows from Kafka topics to Druid, the software conducts real-time statistical analysis on the ingested information. This includes but is not limited to the calculation of aggregates, grouping of data based on specified dimensions, and the generation of meaningful visualizations for users. By subscribing to Kafka topics, Druid ensures that it stays synchronized with the latest data, allowing MIIOT applications to derive actionable insights without delays. The statistical analysis performed by Druid is integral in empowering users and decision-makers with up-to-the-moment information, enhancing the overall efficiency and responsiveness of the MIIOT ecosystem.

3.1.4. Secured Storage Domain

Within this domain, blockchain and database work in tandem to establish a decentralized and secure ledger. This collaboration ensures the integrity and immutability of data by preserving hashed versions of the transmitted information. Acting as an intermediary, Kafka receives data from devices and conveys it through the pipeline to the database. Subsequently, the data undergo a secure hashing process, and the resulting hash is stored within the blockchain, creating an unalterable record of the original data.

In our study, we integrated the Inter Planetary File System (IPFS) to complement the blockchain, enhancing the hosting of generated MIoT data and improving the overall system's efficiency. IPFS serves as a valuable addition to the blockchain by functioning as a decentralized peer-to-peer network for file sharing and storage, utilizing unique resource addresses to locate and retrieve data through Distributed Hash Tables (DHTs). IPFS effectively reduces redundancies and conserves storage space by using file hashes. It is important to note that IPFS operates differently from blockchain; once a file is uploaded to the network, it cannot be intentionally removed. However, IPFS requires periodic cache clearing to gradually phase out less popular files from the network (W. Li, Wang, & Li, 2022).

CHAPTER 4

IMPLEMENTATION

In this section, we will discuss the practical implementation of the previously mentioned platforms and their integration into a comprehensive solution. Given the primary emphasis on real-time data streaming in this study, the focus will predominantly be on explaining the implementation processes associated with Apache Kafka and Apache Druid. Furthermore, we will explore the integration of Software-Defined Radios (SDRs) and sensors, elucidating the steps involved in sending and receiving data.

4.1. Navigating SDR Setup (1st PicoLTE)

This section focuses on the initial setup of the first PicoLTE as our Software Defined Radio (SDR). To accomplish this, we followed a structured process consisting of four essential steps.

4.1.1. Initial Hardware and Software Configuration of 1st PicoLTE

In the initial step of setting up the 1st PicoLTE, we connected computer peripherals to the device to establish a communication interface. This involved linking essential peripherals such as keyboards, mice, and monitors to the PicoLTE unit to enable interaction and configuration. Following this, the second step entailed establishing antenna connectivity. Specifically, we ensured proper positioning and alignment by connecting the Rx1 (Receiver 1) and Tx1 (Transmitter 1) antennas with a prescribed separation distance of 60cm as it can be seen from Figure 4.1. This separation distance was selected based on optimal performance considerations and the specific requirements of our deployment scenario.

In the software configuration phase, we set up the operating environment on Ubuntu 18.04 Long Term Support (LTS) to integrate with our system architecture. This involved a series of steps, starting with the configuration of Ubuntu software to ensure compatibility and optimal

performance. Additionally, we undertook the task of downloading and installing the necessary Software Development Kits (SDKs) and drivers essential for conducting comprehensive testing procedures.

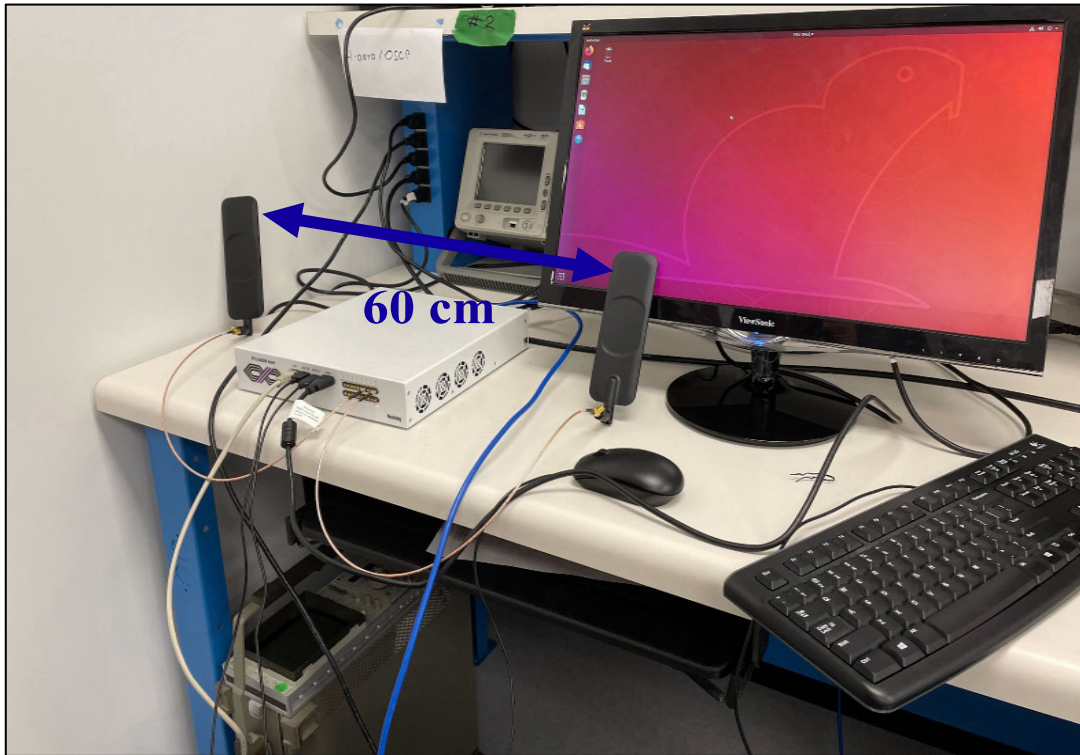


Figure 4.1 Hardware setup of the 1st PicoLTE

Furthermore, Ethernet connection configuration was another step of the setup process. We configured the Ethernet connection settings to establish reliable and high-speed connectivity between the PicoLTE unit and the Ubuntu workstation. This involved configuring network parameters such as IP addresses, subnet masks, and gateway settings to ensure communication over the Ethernet interface. You can see the software setup specifications in Figure 4.2.

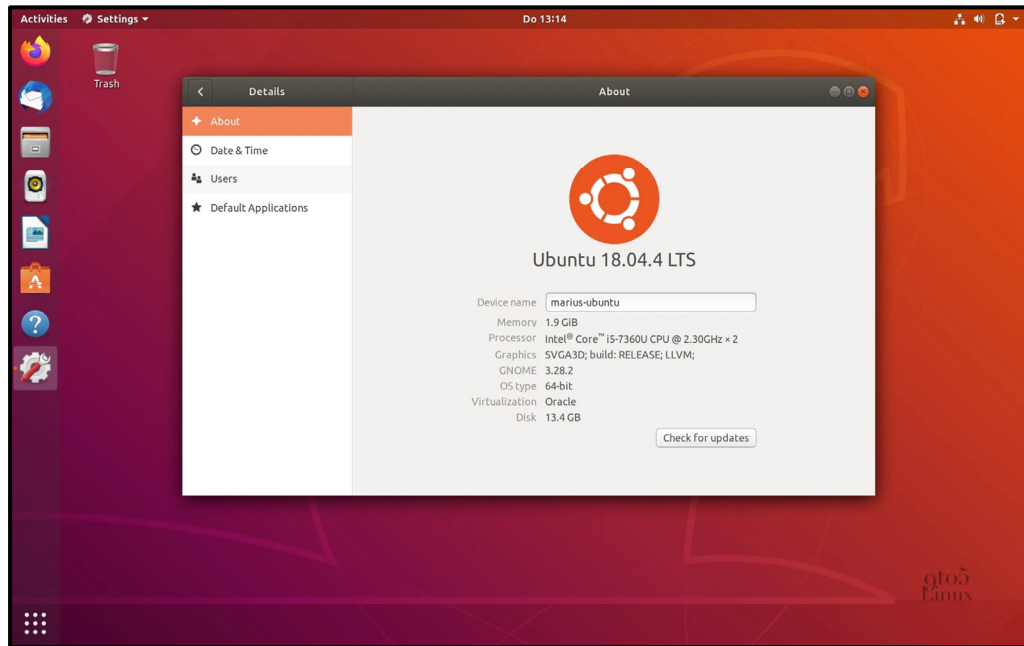


Figure 4.2 Software setup of the 1st PicoLTE

4.1.2 LTE Stack Launch

The second step of our deployment process focuses on launching the LTE stack. Initially, we commenced by mounting the PCIe (Peripheral Component Interconnect Express) driver of the PicoLTE unit to facilitate the establishment of communication between the hardware components and the underlying software infrastructure. This process involved configuring the necessary drivers to enable the PicoLTE unit to interface with the host system effectively which we set it on R2001. Following this, we focused on configuring the Evolved Packet Core (EPC), responsible for managing user sessions, mobility, and connectivity. By configuring the EPC, we ensured reliable operation of the LTE network infrastructure. Moreover, we set the transmission mode on TM3 (Transmission Mode), configuring the downlink and uplink antenna configuration to optimize spectral efficiency and throughput. This involved configuring the system to utilize two downlink antennas and one uplink antenna, aligning with our deployment requirements and performance objectives. Additionally, we configured the system to operate within a bandwidth of 20 MHz. Furthermore, we proceeded with setting up

IP addresses to have the communication and data exchange between network nodes. Finally, with all configurations executed, we initialized and launched the LTE stack and the cellphone as User Equipment could be detected by Pico as shown in Figure 4.3.

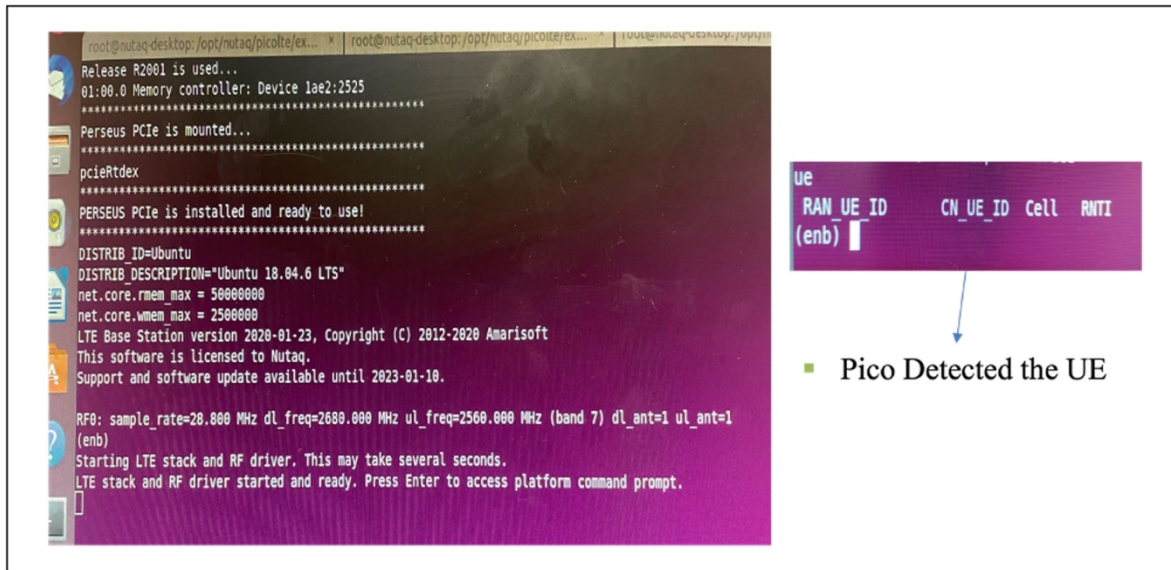


Figure 4.3 LTE Stack has been launched

4.1.3. Validating PicoLTE's Network Connectivity

During this phase, we conducted a validation of Pico LTE connectivity. After a minute of network scanning, the PicoLTE network became detectable on our User Equipment. As depicted in Figure 4.4, our connection to the Internet was established, allowing browsing of various websites.

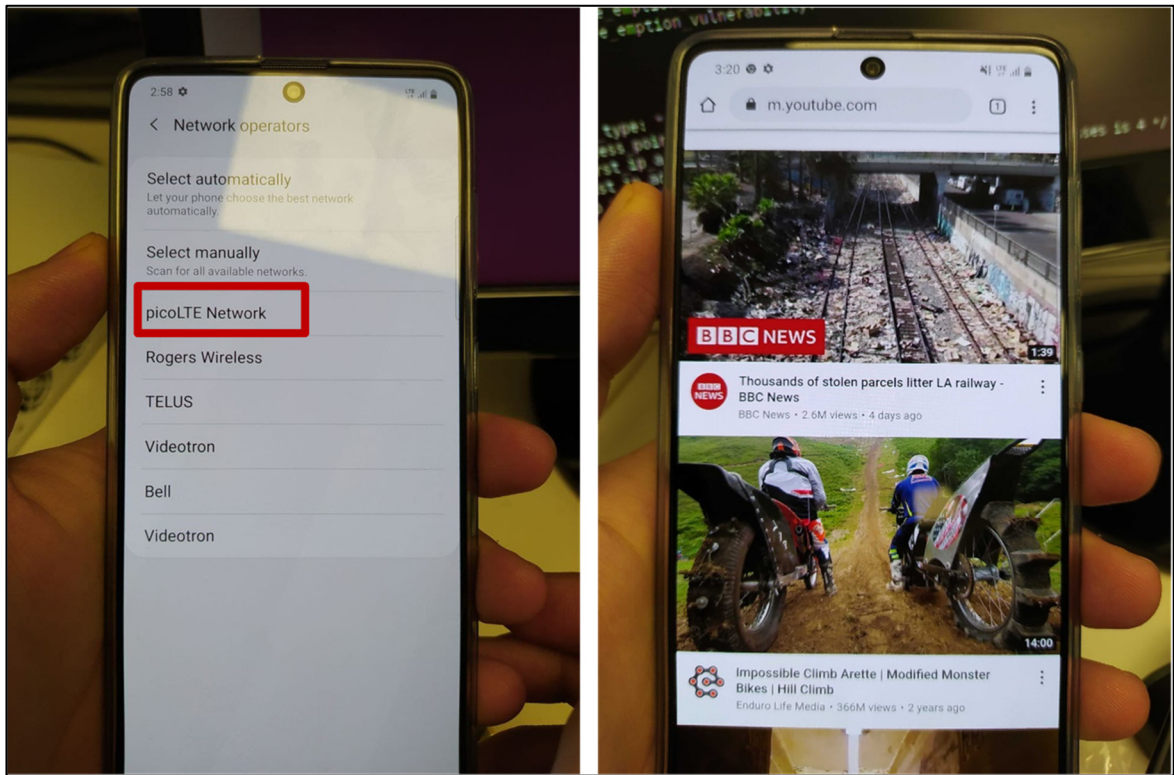


Figure 4.4 Validating PicoLTE's connectivity

4.1.4. Signal Receiver Power Testing

We assessed the LTE signal strength both within and outside the laboratory environment. As demonstrated in Figure 4.5, the results align with the acceptable range, typically falling between -50 and -85 dBm, signifying satisfactory signal strength.

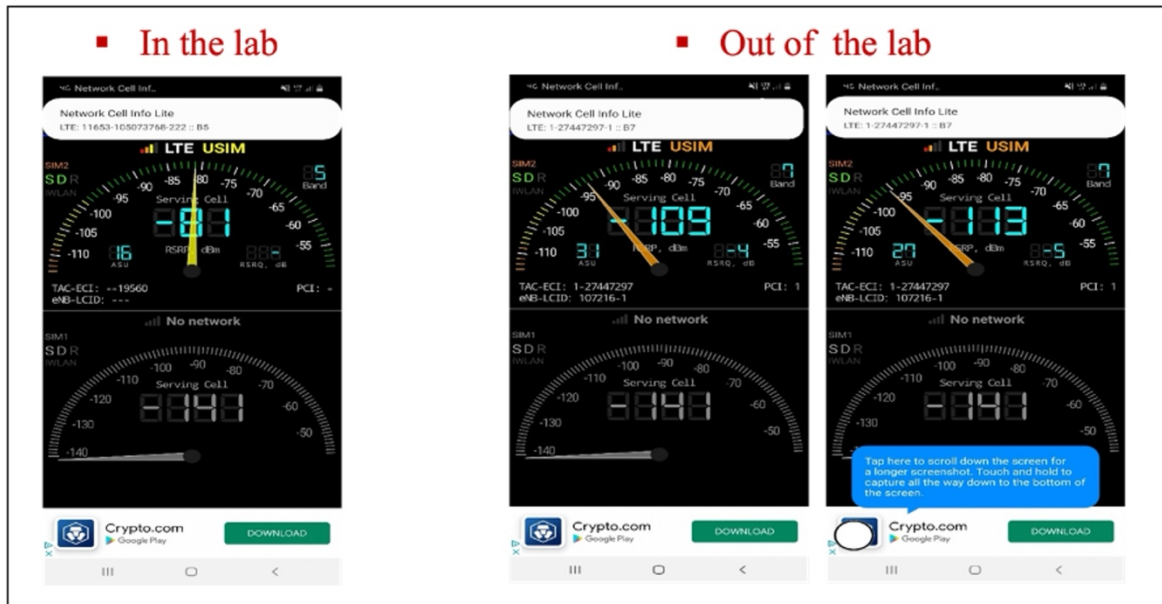


Figure 4.5 LTE Signal Strength

4.2. Real-time Data Streaming's Implementation Procedure

Throughout the implementation of our platform, we utilized Docker images for containerization. Containerization, a technology gaining prominence, serves as a lightweight form of virtualization for defining software infrastructure. These containers enable the packaging of an application along with its dependencies and execution environment into a standardized, self-contained unit. This unit can be utilized for both software development and running the application on diverse systems. Docker containers have become widely adopted as the de facto standard format, owing to their rapid proliferation in the software development community (Hummer, Rosenberg, Oliveira, & Eilam, 2013). The Docker container's contents are explicitly defined in a Dockerfile, serving as a set of instructions to achieve a specific infrastructure state. By storing Dockerfiles in source code repositories, the execution of program code in an isolated and efficient environment becomes achievable with a single command (Cito et al., 2017).

Figure 4.6 shows Kafka implementation procedure. Upon employing Docker images, it becomes essential to designate a specific port, either on a local machine or a server, to enable

access and interaction with the respective container. The initial phase of the implementation focused on deploying Kafka as our real-time data processing platform. Subsequently, a Dockerfile was created to establish a connection with ibNav sensors, facilitating the transmission of data readings to Kafka topics. Following this, the implementation included Apache Druid, intricately connected to Kafka and functioning as a statistical platform. Lastly, the integration of blockchain as a consumer and the incorporation of a web interface into our platform were executed.

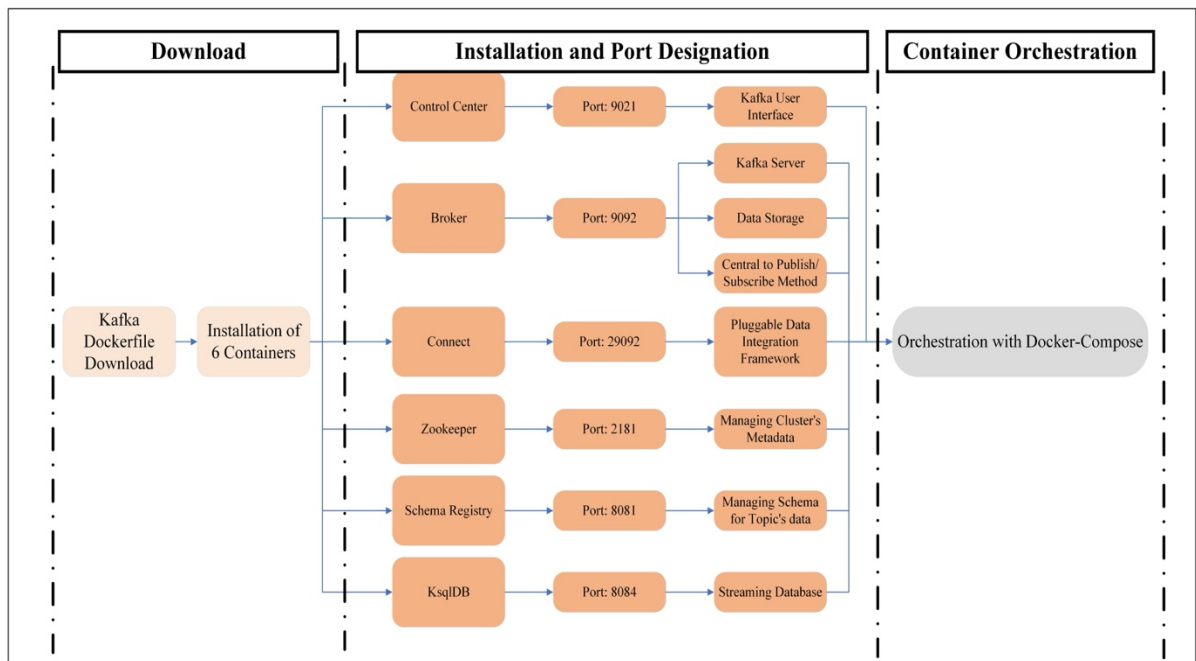


Figure 4.6 Kafka implementation procedure

4.3. Implementation of Real-time Data Streaming (Apache Kafka)

In this segment, we will provide a comprehensive overview of the detailed implementation of Kafka as the real-time data processing platform. We opted for a containerized implementation for optimal execution. To achieve this, we utilized a docker-compose file, a tool designed to streamline the orchestration of multi-container Docker applications. This tool ensures the

systematic orchestration of Docker applications. In our docker-compose file, we incorporated a total of 5 containers, as shown in Figure 4.7:

- *Zookeeper*: As previously indicated, Zookeeper plays an instrumental role in overseeing cluster metadata and preserving broker leader statuses. Kafka offers a default and straightforward Zookeeper configuration file designed for initiating a single local Zookeeper instance. In this context, Zookeeper functions as the coordination interface between the Kafka broker and consumers.
- *Broker*: As previously noted, a broker is an individual Kafka server tasked with collecting messages from producers, assigning offsets to them, and storing the messages on disk storage. Additionally, it addresses consumer requests by responding to partition fetch requests and delivering the published messages.
- *Connect*: Kafka Connect, an integral component of Apache Kafka, offers an efficient and scalable mechanism for transferring data between Kafka and various datastores. It supplies APIs and a runtime environment for the development and execution of connector plug-ins—libraries executed by Kafka Connect responsible for data movement. Kafka Connect operates as a cluster of worker processes, and connector plug-ins are installed on these workers. Configuration and management of connectors are performed using a REST API, allowing connectors to run with specific configurations. Connectors initiate additional tasks to parallelize the transfer of large data volumes, optimizing resource utilization on the worker nodes.
- *Control Center*: Confluent offers a commercial implementation of Kafka Control Center. This commercial tool serves as a monitoring solution, overseeing message counts, checksums, and addressing monitoring gaps in the system.
- *Schema Registry*: The Schema Registry maintains the complete schema within the data file. It is important to note that the Schema Registry is separate from Apache Kafka. The fundamental concept involves storing all the schemas employed for writing data to Kafka in the registry. Consequently, we only need to store the schema identifier in the record produced for Kafka. Subsequently, consumers can utilize this identifier to retrieve the record from the Schema Registry and deserialize the data. The tasks of storing the schema

in the registry and retrieving it when necessary are handled within the serializers and deserializers.

CONTAINER ID	IMAGE NAMES	COMMAND	CREATED	STATUS
7eddf3c8c01a	confluentinc/cp-enterprise-control-center:6.0.1 control-center	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes
77fcb9d4af57	confluentinc/cp-ksqldb-cli:6.0.1 ksqldb-cli	"/bin/sh"	4 minutes ago	Up 4 minutes
3af07f429297	confluentinc/ksqldb-examples:6.0.1 ksql-datagen	"bash -c 'echo Waiti..."	4 minutes ago	Up 4 minutes
ffa70e33f0d0	confluentinc/cp-ksqldb-server:6.0.1 ksqldb-server	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes
4b66b9339a13	confluentinc/cp-kafka-connect-base:6.0.1 kafka-connect	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes (healthy)
a104f652812e	confluentinc/cp-kafka-rest:6.0.1 rest-proxy	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes
6650814b6687	confluentinc/cp-schema-registry:6.0.1 schema-registry	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes
759cdb2c0c35	confluentinc/cp-server:6.0.1	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes
01->9101/tcp	broker			
f36f25eb7e23	confluentinc/cp-zookeeper:6.0.1 zookeeper	"/etc/confluent/dock..."	4 minutes ago	Up 4 minutes

Figure 4.7 Kafka Containers are up and running

The Control Center serves as a monitoring tool, enabling us to oversee data within topics, brokers, clusters, Kafka Connect, and Zookeeper. As depicted in Figure 4.8, it provides a comprehensive overview of the cluster under consideration.

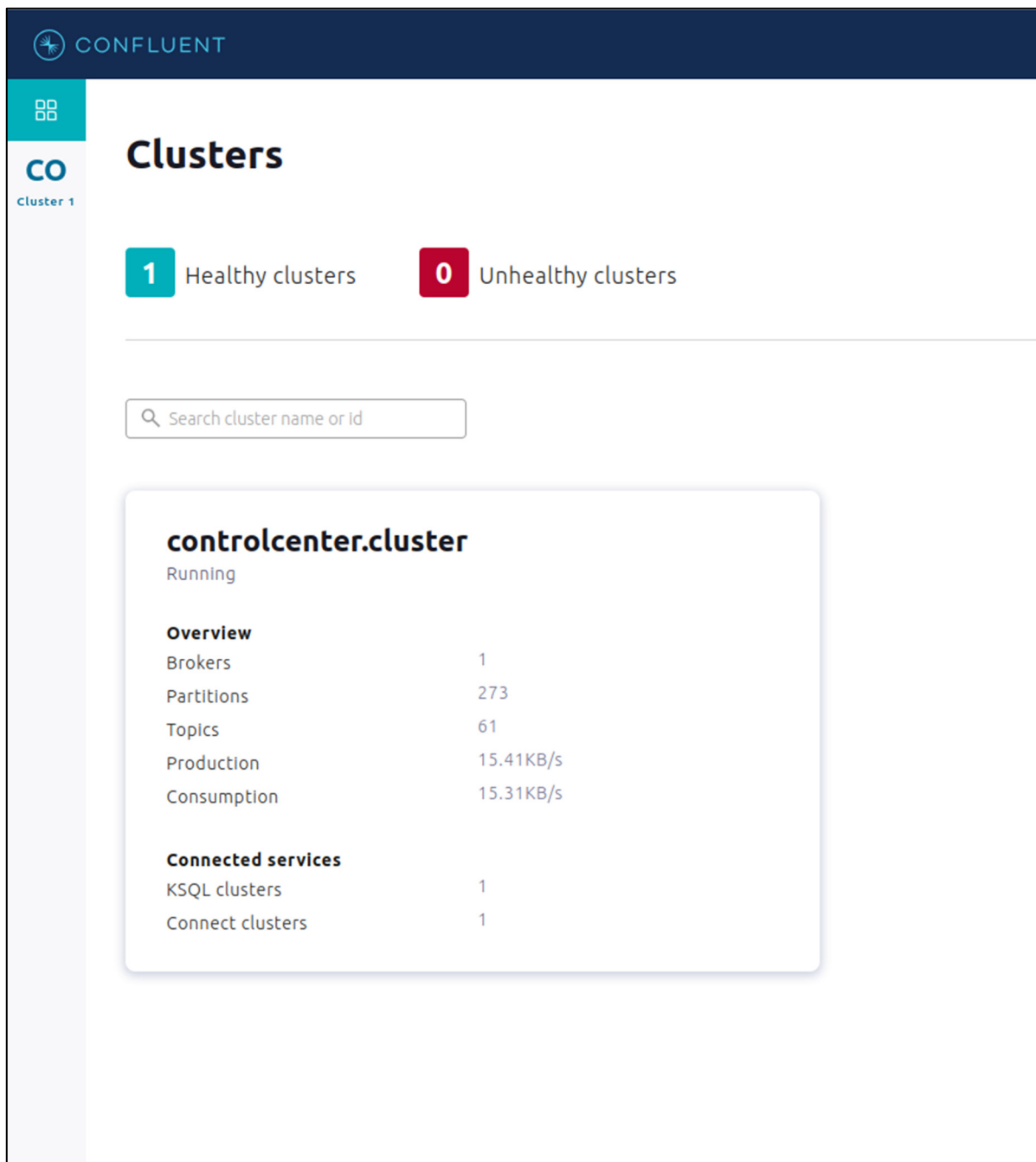
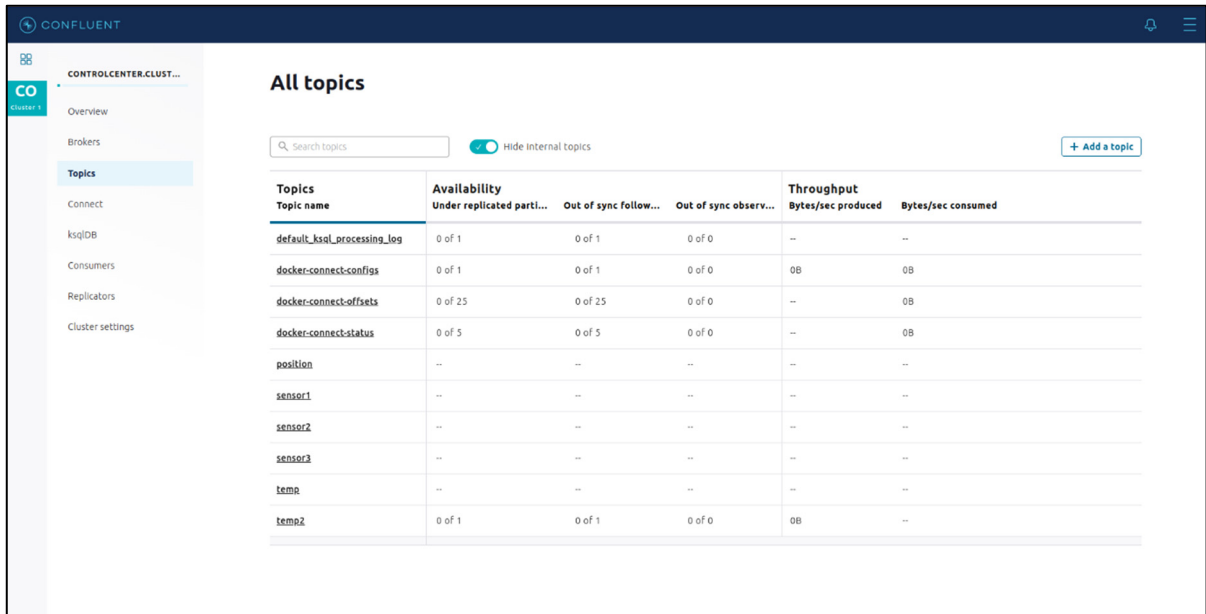


Figure 4.8 Cluster's overview

The topic management functionality, accessible from the menu, is crucial within the Control Center, given that it is where we receive all data within topics, as we see in Figure 4.9.



Topics	Availability			Throughput	
Topic name	Under replicated parti...	Out of sync follow...	Out of sync observ...	Bytes/sec produced	Bytes/sec consumed
default_ksql_processing_log	0 of 1	0 of 1	0 of 0	--	--
docker-connect-configs	0 of 1	0 of 1	0 of 0	0B	0B
docker-connect-offsets	0 of 25	0 of 25	0 of 0	--	0B
docker-connect-status	0 of 5	0 of 5	0 of 0	--	0B
position	--	--	--	--	--
sensor1	--	--	--	--	--
sensor2	--	--	--	--	--
sensor3	--	--	--	--	--
temp	--	--	--	--	--
temp2	0 of 1	0 of 1	0 of 0	0B	--

Figure 4.9 List of topics

4.3.1. Sensors as Producers

We employ two Software-Defined Radios (SDRs) which utilizes LTE-M and 5G as connectivity protocols, and MQTT serves as our communication protocol. The Kafka producer component utilizes MQTT and PicoLTE to establish a connection with ibNav sensors for data reception. Essentially, the producer subscribes to an MQTT topic, preparing itself for data ingestion. The received data undergoes serialization, encoded in the "utf-8" format. Following this, the KafkaProducer command transmits the serialized data to the Kafka cluster, associating it with a specified topic as shown in Figure 4.10. In Figure 4.11, three sensors are illustrated as connected to a pinnacle.

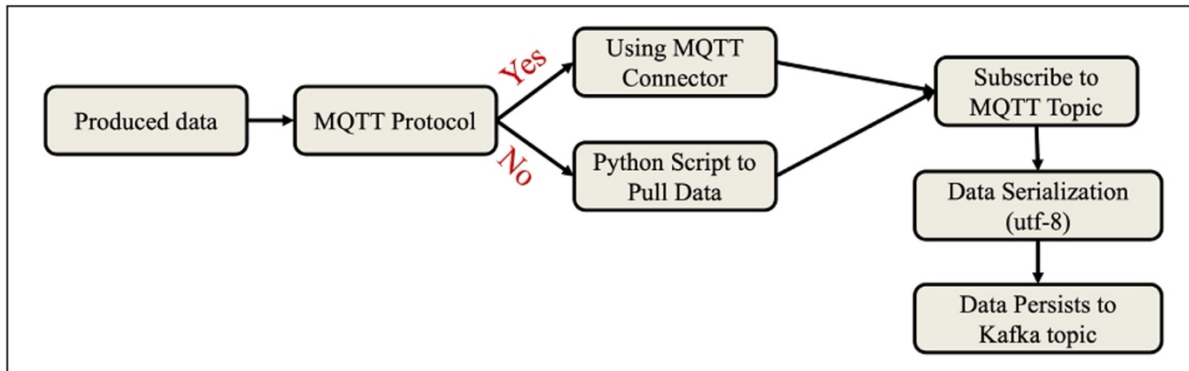


Figure 4.10 Producer procedure

```

Sending the data related to the sensor3 is: {'humidity': 52, 'temperature': 35, 'pressure': 1143, 'time': 'Fri Aug 25 16:35:20 2023'}
Sending the data related to the sensor1 is: {'humidity': 80, 'temperature': 21, 'pressure': 898, 'time': 'Fri Aug 25 16:35:23 2023'}
Sending the data related to the sensor2 is: {'humidity': 37, 'temperature': 15, 'pressure': 1019, 'time': 'Fri Aug 25 16:35:23 2023'}
Sending the data related to the sensor3 is: {'humidity': 69, 'temperature': 13, 'pressure': 1057, 'time': 'Fri Aug 25 16:35:23 2023'}
Sending the data related to the sensor1 is: {'humidity': 38, 'temperature': 8, 'pressure': 948, 'time': 'Fri Aug 25 16:35:26 2023'}
Sending the data related to the sensor2 is: {'humidity': 67, 'temperature': 12, 'pressure': 992, 'time': 'Fri Aug 25 16:35:26 2023'}
Sending the data related to the sensor3 is: {'humidity': 89, 'temperature': 24, 'pressure': 1018, 'time': 'Fri Aug 25 16:35:26 2023'}
Sending the data related to the sensor1 is: {'humidity': 99, 'temperature': 2, 'pressure': 802, 'time': 'Fri Aug 25 16:35:29 2023'}
Sending the data related to the sensor2 is: {'humidity': 24, 'temperature': 22, 'pressure': 1002, 'time': 'Fri Aug 25 16:35:29 2023'}
Sending the data related to the sensor3 is: {'humidity': 86, 'temperature': 9, 'pressure': 1195, 'time': 'Fri Aug 25 16:35:29 2023'}
Sending the data related to the sensor1 is: {'humidity': 7, 'temperature': 6, 'pressure': 800, 'time': 'Fri Aug 25 16:35:32 2023'}
Sending the data related to the sensor2 is: {'humidity': 42, 'temperature': 30, 'pressure': 1101, 'time': 'Fri Aug 25 16:35:32 2023'}
Sending the data related to the sensor3 is: {'humidity': 75, 'temperature': 34, 'pressure': 966, 'time': 'Fri Aug 25 16:35:32 2023'}
Sending the data related to the sensor1 is: {'humidity': 62, 'temperature': 31, 'pressure': 1112, 'time': 'Fri Aug 25 16:35:35 2023'}
Sending the data related to the sensor2 is: {'humidity': 25, 'temperature': 22, 'pressure': 1003, 'time': 'Fri Aug 25 16:35:35 2023'}
Sending the data related to the sensor3 is: {'humidity': 15, 'temperature': 28, 'pressure': 1155, 'time': 'Fri Aug 25 16:35:35 2023'}
Sending the data related to the sensor1 is: {'humidity': 47, 'temperature': 2, 'pressure': 970, 'time': 'Fri Aug 25 16:35:38 2023'}
Sending the data related to the sensor2 is: {'humidity': 48, 'temperature': 39, 'pressure': 1065, 'time': 'Fri Aug 25 16:35:38 2023'}
Sending the data related to the sensor3 is: {'humidity': 3, 'temperature': 27, 'pressure': 1149, 'time': 'Fri Aug 25 16:35:38 2023'}
Sending the data related to the sensor1 is: {'humidity': 47, 'temperature': 32, 'pressure': 1138, 'time': 'Fri Aug 25 16:35:41 2023'}
Sending the data related to the sensor2 is: {'humidity': 87, 'temperature': 22, 'pressure': 812, 'time': 'Fri Aug 25 16:35:41 2023'}
  
```

Figure 4.11 Topics are receiving data

4.3.2. Blockchain as Consumers

In this section, we explore the deployment of the consumer component in the Apache Kafka framework. When the Kafka producer sends data or information to the Kafka cluster under a designated topic, it becomes the responsibility of the consumer to retrieve this data. The primary function of the consumer is to receive and then transmit this data to the necessary storage system. For our study, we have opted for the InterPlanetary File System (IPFS) as our decentralized database, as shown in Figure 4.12.

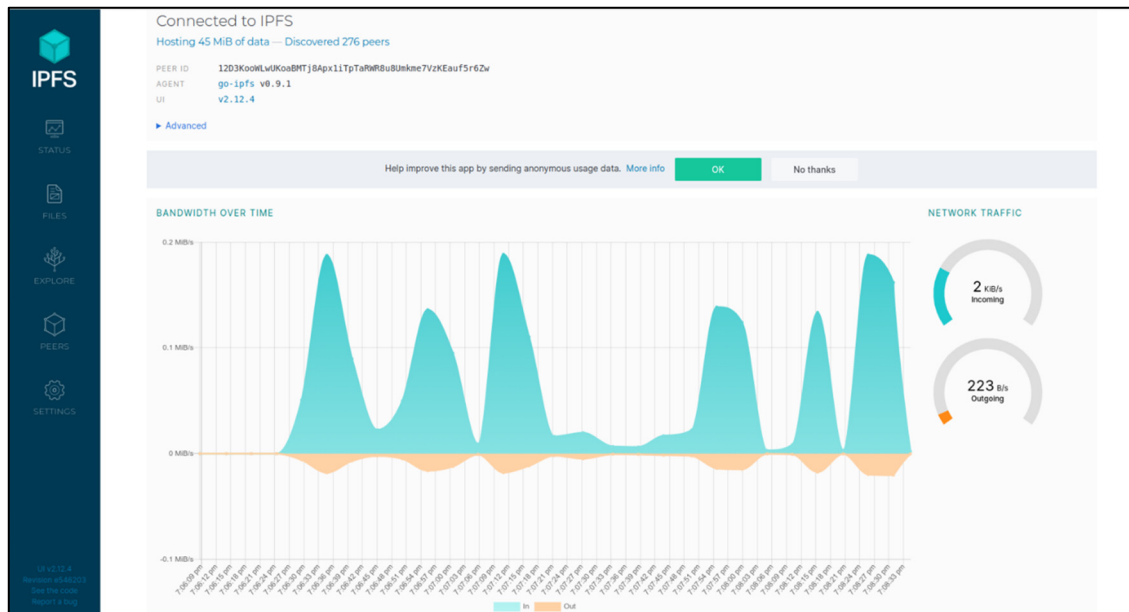


Figure 4.12 IPFS user interface for monitoring

4.4. Druid on Top of Kafka

As previously mentioned, we employed Apache Druid as our data analytics software layered on Kafka. To achieve this, we downloaded Druid and installed the Java Development Kit (JDK). After configuring the environment variables, we initiated the Apache Druid services, enabling the execution of Apache Druid on localhost:8888. The dashboard of Apache Druid is illustrated in Figure 4.13:

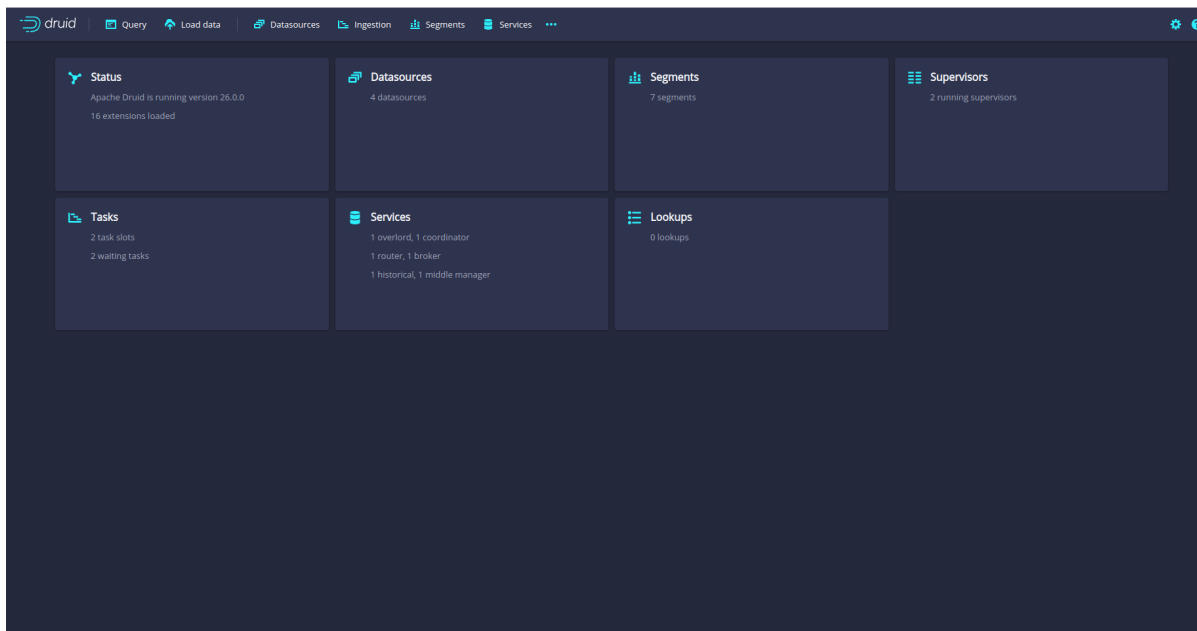


Figure 4.13 Apache Druid's dashboard

As evident from the dashboard of Druid, we have the capability to ingest data in real-time into Druid for monitoring and analytical purposes. Figure 4.14 depicts the dashboard where data can be loaded from various sources, including Kafka, Azure, and others:

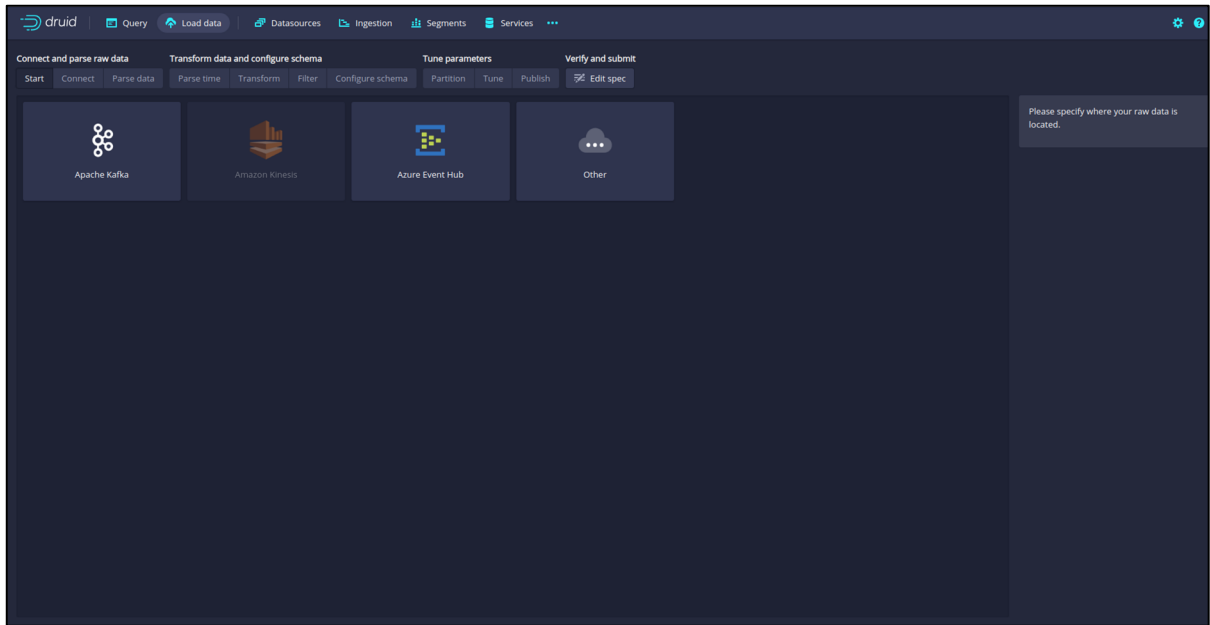


Figure 4.14 Loading data from different sources into Apache Druid

For instance, to ingest data from Kafka, we need to input Kafka's broker address and the topic's name. This enables us to parse the data in the subsequent stage and establish a supervision in Druid for analytical purposes (Figure 4.15).

Druid ingests raw data and converts it into a custom, [indexed format](#) that is optimized for analytic queries.

To get started, please specify what data you want to ingest.

[Learn more](#)

Bootstrap servers ⓘ

localhost:9092

Topic

topic_name

Consumer properties ⓘ

```
{  
  "bootstrap.servers": "localhost:9092"  
}
```

Where should the data be sampled from?

☒ Start of stream

☐ End of stream

Cancel Apply

Next: Parse data →

Figure 4.15 Ingestion from Kafka

4.5. User Interface

Our web interface provides authenticated users with the ability to interact with the platform. The administrator is responsible for incorporating both users and devices into the system.

ReactJS, a well-known JavaScript library with a focus on component-based architecture, was employed for interface development.

As it can be seen in Figure 4.16, the interface includes a dashboard that allows users to connect and engage within their designated space. Device registration is facilitated through a smart contract designed for efficient device and user management, resulting in the generation of a unique UUID for each device upon successful registration.

This section also underscores the significance of data integrity. A meticulous examination of each block ensures that its hashed data is correctly linked and chained, preserving the consistency and reliability of the entire dataset. Visualization techniques, utilizing various libraries such as react-plotly.js, have been implemented for effective data representation.

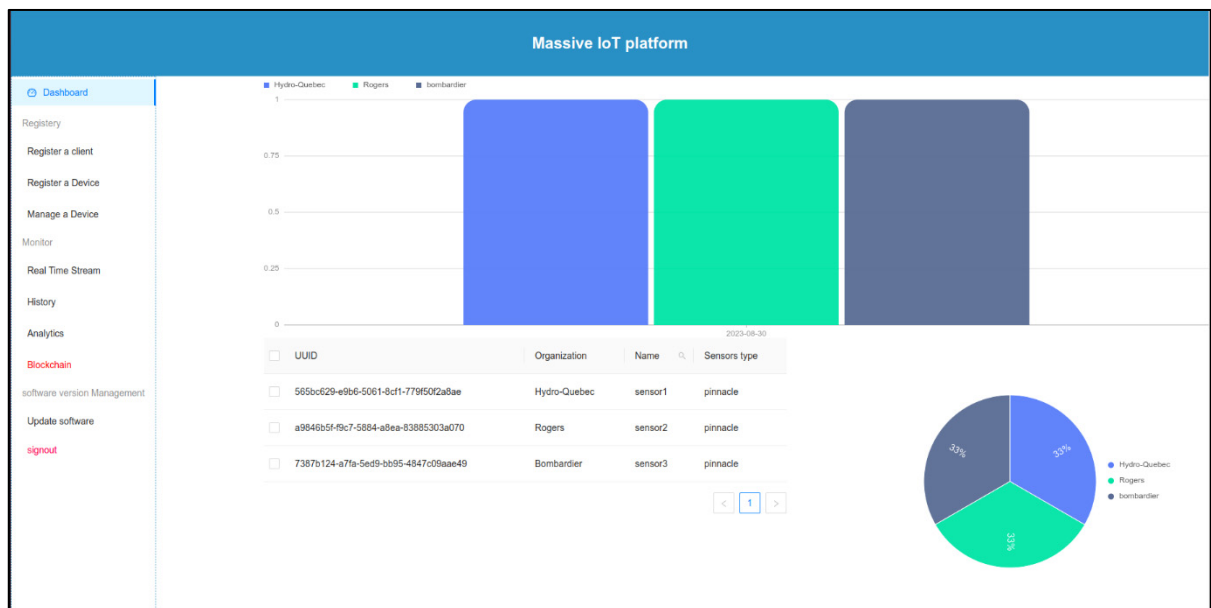


Figure 4.16 User interface dashboard

CHAPTER 5

RESULTS AND ANALYSIS

5.1. Simulation Parameters

The study outlined in this thesis concentrated on tackling the data processing and communication challenges in MIIoT through the publish/subscribe approach, coupled with the utilization of blockchain for secure data storage, with a specific focus on latency. Throughout the experimentation and analysis, several insights into latency were discovered, providing insights into the system's performance and efficacy. To measure latency, we utilized Kafka's Confluent Center and Apache Druid, leveraging the built-in feature for displaying statistics alongside predefined use cases. Regarding hardware, we employed an Ubuntu machine with configurations outlined in Table 5.1.

Table 5.1 Simulation parameters

Parameters	Value
CPU	Intel(R) Core(TM) i7-9750U CPU @ 2.60 GHz
RAM	16 GB
Storage	512 GB
No. of Kafka clusters	1
No. of Kafka brokers	1 and 2
No. of Kafka topics	50 to 500
No. of messages per second	1000 to 100,000

5.2. Latency of Druid without Kafka

In Figure 5.1, we display the latency measurements for Apache Druid in a scenario where Apache Kafka is not active. The graph depicts the fluctuation in latency across different message rates. Notably, a substantial rise in latency is observed when Apache Druid contends

with a message rate of 100,000 messages per second. It is crucial to highlight that the messages in this context include temperature, humidity, and pressure values, which are not typically characterized by high volume.

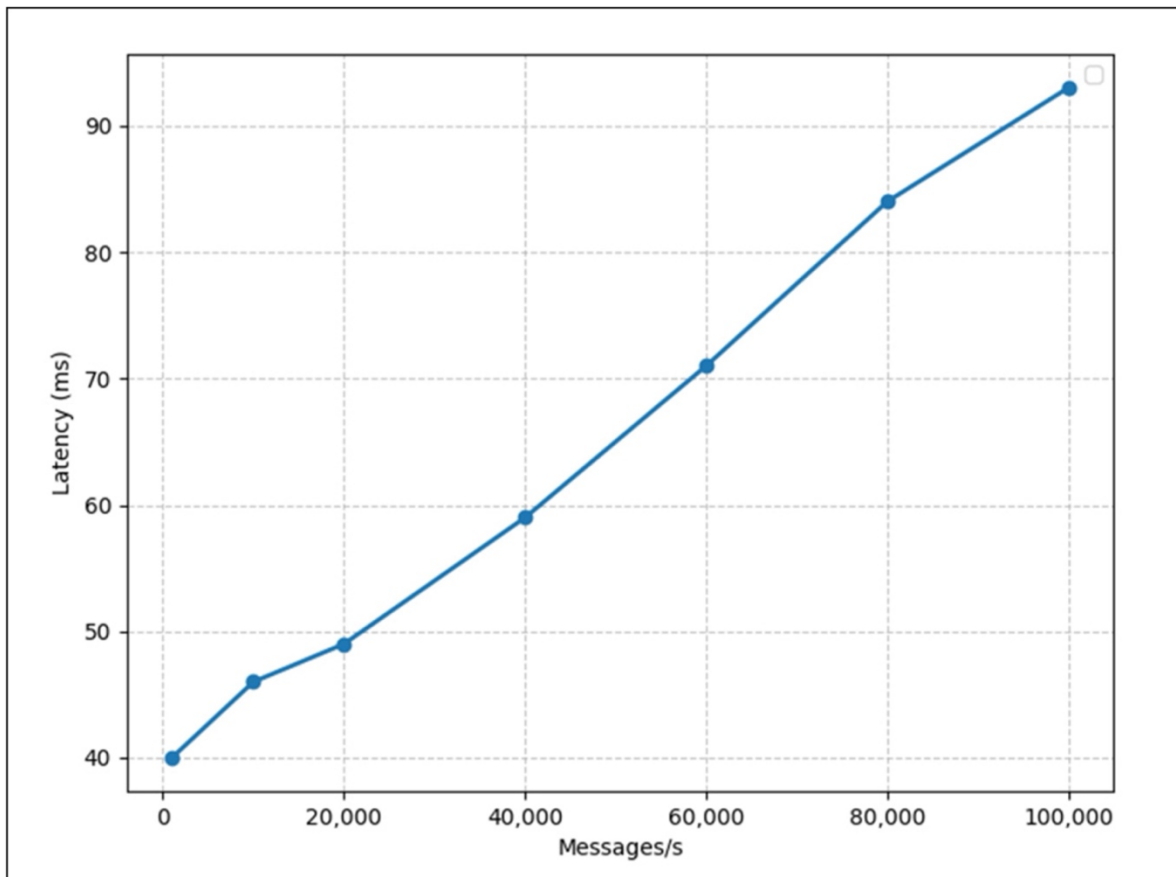


Figure 5.1 Apache Druid's latency without Kafka

5.3. Latency with Kafka in Operation

As it can be seen in Figure 5.2, we investigate the influence of integrating Apache Kafka with varying numbers of topics in a single-broker configuration. In this scenario, we observe consistently low latency, reaching below 25 milliseconds, even when processing 100,000 messages per second. In comparison to the findings in (Sai Lohitha & Pounambal, 2023), where they report an average latency of around 25 ms for a workload of 60 messages per

second with a message size of 10,000 KB, it is important to note that their scale may not be representative of MIIoT. This underscores the difference in performance evaluation between the two studies, highlighting the efficiency and reliability of our use of Kafka for managing data streams with diverse topics, ensuring minimal latency for real-time data processing.

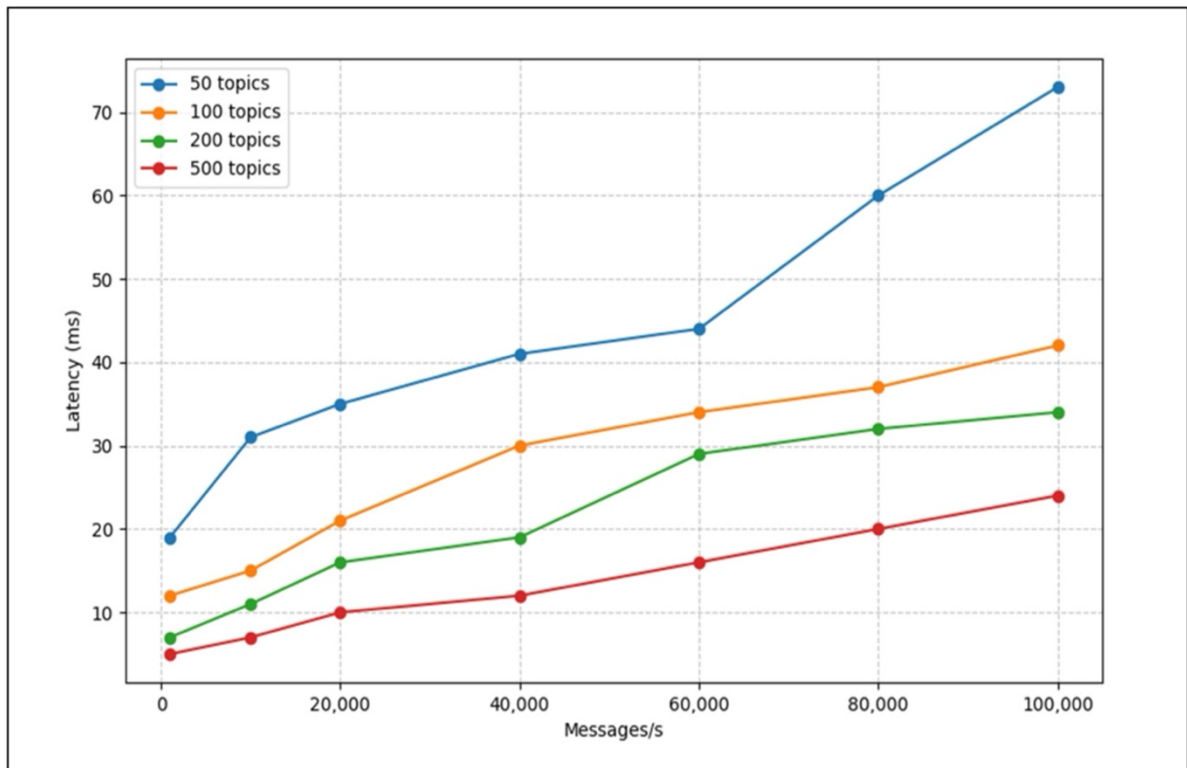


Figure 5.2 Apache Druid latency with Kafka

In conclusion, our systems are structured to effectively handle a network of up to 100,000 sensors, each capable of transmitting low-volume measurements every second, while supporting 500 topics. These specifications hold true within a framework of consistently low latency.

5.4. Resource Utilization

As mentioned earlier, each broker has a dedicated queue. When a producer sends a message, it enters the queue, where it primarily stays in an idle state, occasionally transitioning to an active state. The request pool usage metric reflects the average capacity utilization of request handlers across all brokers. Simply put, it measures the percentage of time that the request handler threads are actively occupied and not in an idle state. Figure 5.3 demonstrates that, at a data rate of 100,000 messages per second, the request handler is active for only 2.06 percent of the time, indicating an exceptionally efficient use of resources.

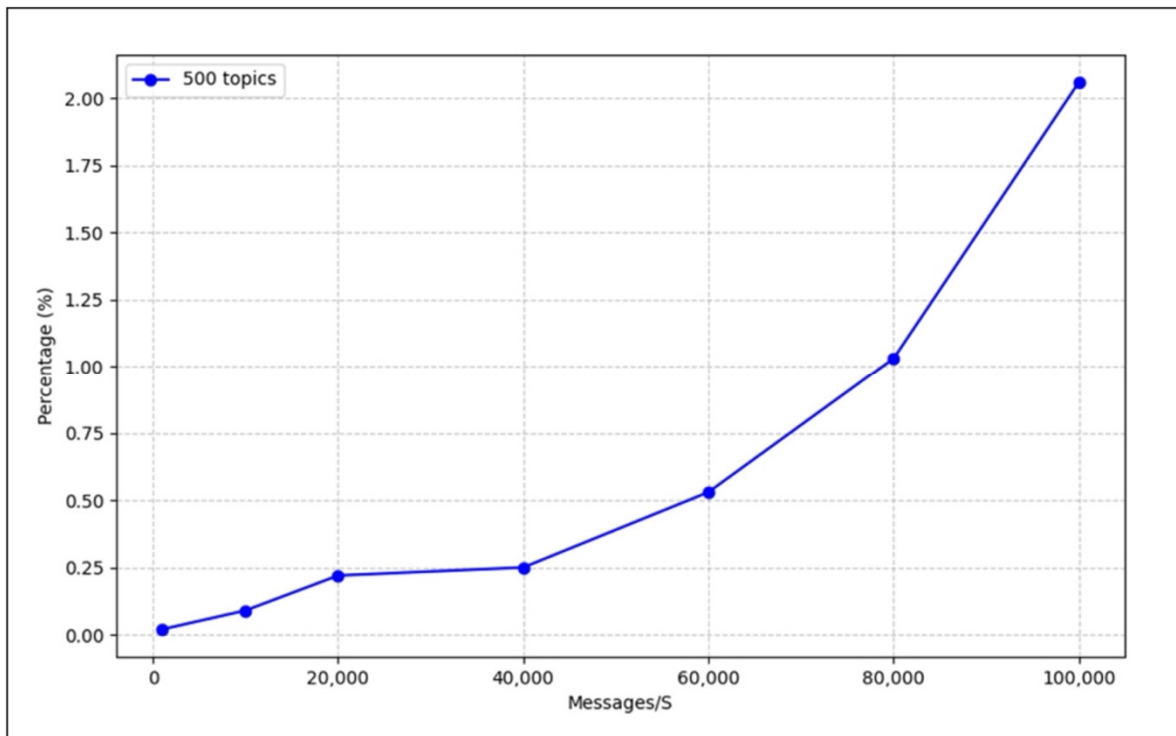


Figure 5.3 Request pool usage

5.5. Throughput and Availability Trade-off

In this section, we dive into the essential trade-off between throughput and availability within our system. Our investigation reveals an interesting dynamic: a single-broker configuration

yields higher throughput, while the introduction of two brokers may lead to a slight degradation. In a dual-broker configuration, one broker takes on the role of the leader, while the other functions as the follower. This arrangement ensures the replication of partitions from the leader broker to the follower broker in the event of a broker failure, ensuring continuous data availability. It is worth noting that the introduction of two brokers results in a marginal, though not significant, decrease in throughput, as illustrated in Figure 5.4.

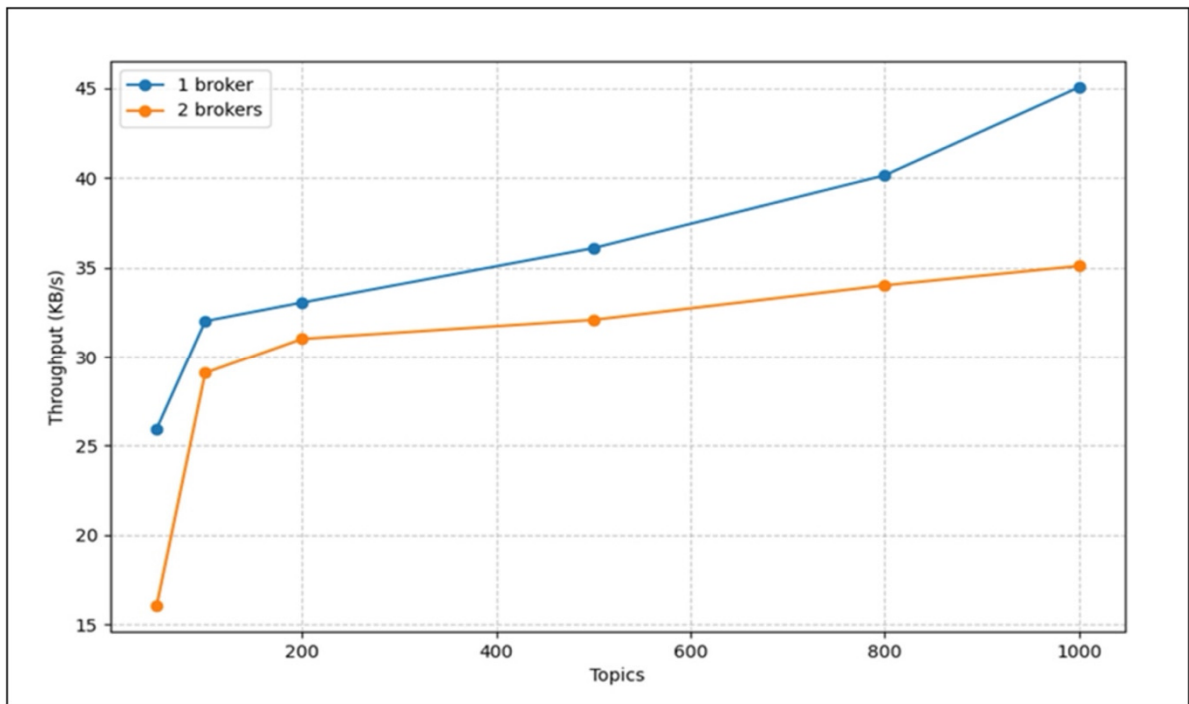


Figure 5.4 Throughput comparison with one and two brokers

5.6. Blockchain's Performance

We employed Hyperledger Caliper, an open-source tool freely available, to evaluate the performance of our blockchain platform. Our testing focused on Hyperledger Fabric, utilizing hardware identical to that used in the Kafka test. Figure 5.5 displays the performance results for writing and reading transactions during our evaluation. The figure clearly illustrates a direct

correlation between Transactions Per Second (TPS) and the number of transactions, emphasizing the effectiveness of the proposed platform.

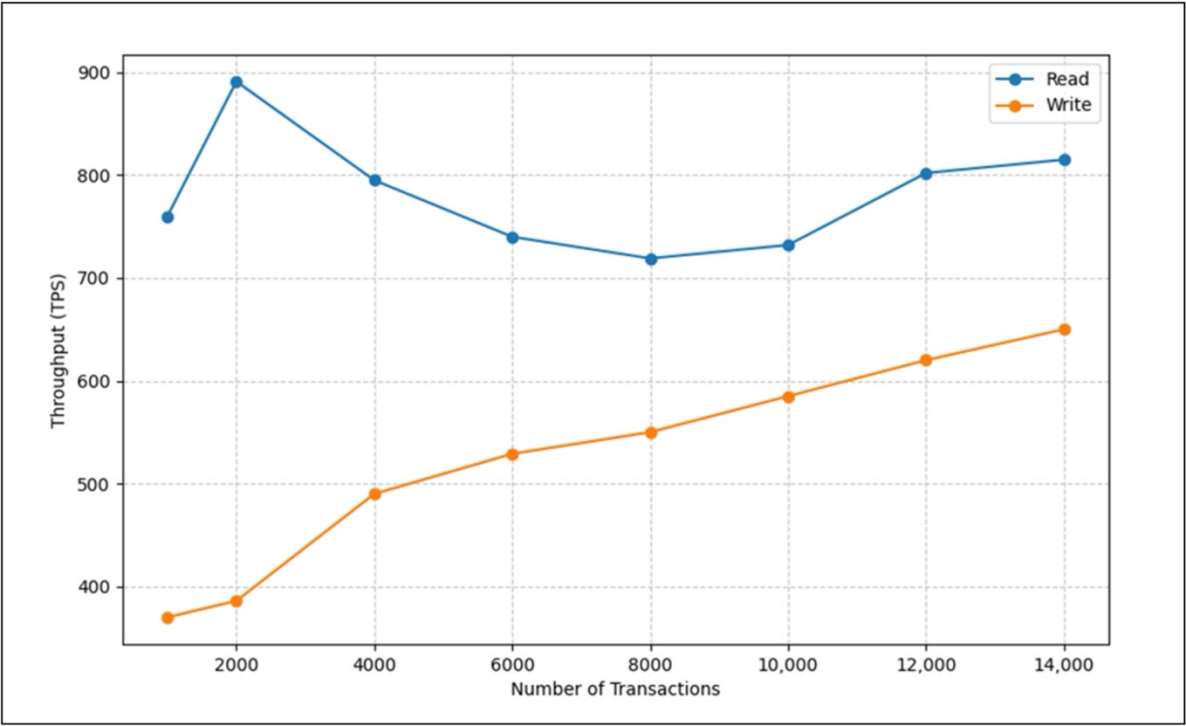


Figure 5.5 Performance of Hyperledger

Moreover, our results, illustrated in Figure 5.5, indicate that as the number of transactions increases, the success rate of accepting read transactions remains consistently high. This observation emphasizes the network's robust ability to manage massive volume of read requests while ensuring the accuracy and reliability of the data retrieval process.

CONCLUSION

As the Internet of Things landscape undergoes continuous transformation, the rise of Machine-Type Devices (MTDs) has given rise to the emergence of MIoT, a significant trend in the realm of connected devices. Our thesis utilizes the publish/subscribe approach for data storage, introducing noteworthy enhancements in scalability and latency for the interconnected world of IoT. Integration of Apache Druid, an analytical database software, into our framework empowers real-time data processing, elevating the capabilities of MIoT applications. The strategic selection of Apache Kafka as the data transmission platform streamlines the data flow from sensors to brokers, creating an efficient subscription ecosystem that caters to the dynamic needs of IoT applications. Our proposed architecture envisions the system effectively supporting an IoT network comprising a minimum of 100,000 sensors, each transmitting low-volume messages per second, across a network featuring 1 broker and 500 topics.

Furthermore, we enhance connectivity resilience by simulating the presence of two mobile network operators (MNOs) through Software-Defined Radio (SDR) devices, providing a reliable backup for uninterrupted data transmission, even in the face of connection failures.

Our unwavering commitment to data security is evident in the deployment of Hyperledger Fabric, a cutting-edge blockchain technology known for its robust security and data integrity features. The results of our comprehensive analysis underscore the model's core attributes, highlighting its remarkable scalability, unwavering availability, robust security protocols, and minimal latency, positioning it as a formidable solution in the ever-evolving area of MIoT.

6.1. Discussion

In this section, we will dive into the favorable attributes associated with the adoption of the proposed MIoT framework, enabling enterprises to access a variety of valuable outcomes and enhance their overall operations. As mentioned earlier, our thesis takes advantage of publish/subscribe messaging method, integrating Apache Druid and Apache Kafka to achieve heightened scalability and low latency in MIoT applications. Notably, our latency results,

reaching less than 25 ms for 100,000 sensors, surpass existing works reporting a similar latency for considerably fewer devices. With support for a massive number of sensors, fortified by dual mobile network operators to ensure connectivity resilience, and the utilization of Hyperledger Fabric for storage security, achieving over 800 successful TPS in 14,000 transactions, our model represents a breakthrough in MIIoT innovation, paving the way for future research and advancements. The positive aspects can be categorized into six sections, as follows:

6.1.1. Low Latency

Leveraging Kafka's publish/subscribe architecture obviates the necessity for real-time application control and data allocation, as data are asynchronously available for subscription from any IoT application in Kafka topics. This characteristic notably enhances the low-latency performance of our framework.

6.1.2. Data Retention with Kafka

Kafka's data retention functionality plays a fundamental role in our framework, ensuring data accessibility in scenarios involving latency or disruptions. The customized retention of data by Kafka enhances the system's reliability and resilience, providing the capability for historical data analysis and auditing.

6.1.3. Scalability

The framework's scalability is rooted in Kafka's capability to distribute brokers across multiple clusters within different cloud services. Partitioning is a key factor in this scalability, as scaling a partition across multiple brokers allows different consumers to simultaneously access the same partition across various clusters. This becomes particularly advantageous during periods of high message traffic for a specific Kafka topic.

6.1.4. Availability

Our framework prioritizes maintaining continuous service availability, and Kafka's built-in fault tolerance, achieved through data replication and a distributed architecture, is instrumental in achieving this goal. The incorporation of multiple brokers and leader-follower configurations, as explained earlier, further enhances availability by ensuring data continuity even in the event of a broker failure. Moreover, the presence of two Mobile Network Operators (MNOs) ensures connectivity availability in the face of connection failures.

6.1.5. Integrity and Security

By utilizing blockchain technology, our framework places a strong emphasis on data security and integrity. The inherent immutability of blockchain ensures that data remains resistant to tampering, thereby preserving the trustworthiness of stored information. Our system's security is strengthened by robust measures, such as multi-layered authentication protocols and encryption, creating a protective shield against potential threats and safeguarding sensitive data.

6.2. Research Limitation

It is crucial to acknowledge that while the publish/subscribe approach demonstrates resilience in various MIIOT scenarios, its efficacy in synchronous communication scenarios, such as media streaming, may encounter limitations. These considerations, among others, present opportunities for future research and innovation, encouraging further investigation into the dynamic and diverse realm of MIIOT.

LIST OF REFERENCES

- Abdmeziem, M. R., Tandjaoui, D., & Romdhani, I. (2015). Architecting the Internet of Things: State of the Art. *Architecting the Internet of Things*, 21.
- Agrawal, R., Verma, P., Sonanis, R., Goel, U., De, A., Kondaveeti, S. A., & Shekhar, S. (2018). Continuous Security in IoT Using Blockchain. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6423–6427. Calgary, AB: IEEE. <https://doi.org/10.1109/ICASSP.2018.8462513>
- Alotaibi, A., & Barnawi, A. (2023). Securing massive IoT in 6G: Recent solutions, architectures, future directions. *Internet of Things*, 22, 100715. <https://doi.org/10.1016/j.iot.2023.100715>
- Al-Qaseemi, S. A., Almulhim, H. A., Almulhim, M. F., & Chaudhry, S. R. (2016). IoT architecture challenges and issues: Lack of standardization. *2016 Future Technologies Conference (FTC)*, 731–738. San Francisco, CA, USA: IEEE. <https://doi.org/10.1109/FTC.2016.7821686>
- Alshaikhli, M., Elfouly, T., Elharrouss, O., Mohamed, A., & Ottakath, N. (2022). Evolution of Internet of Things From Blockchain to IOTA: A Survey. *IEEE Access*, 10, 844–866. <https://doi.org/10.1109/ACCESS.2021.3138353>
- Amanlou, S., Hasan, M. K., & Bakar, K. A. A. (2021). Lightweight and secure authentication scheme for IoT network based on publish–subscribe fog computing model. *Computer Networks*, 199, 108465. <https://doi.org/10.1016/j.comnet.2021.108465>
- Amnalou, S., & Azmi, K. (2020). Lightweight Security Mechanism over MQTT Protocol for IoT Devices. *International Journal of Advanced Computer Science and Applications*, 11(7). <https://doi.org/10.14569/IJACSA.2020.0110726>
- Anamuro, C. A. V. (n.d.). *Opportunity and challenges of Device-to-Device relaying for IoT connectivity in cellular networks*. 175.
- Ande, R., Adebisi, B., Hammoudeh, M., & Saleem, J. (2020). Internet of Things: Evolution and technologies from a security perspective. *Sustainable Cities and Society*, 54, 101728. <https://doi.org/10.1016/j.scs.2019.101728>
- Annamalai, P., Bapat, J., & Das, D. (2018). Emerging Access Technologies and Open Challenges in 5G IoT: From Physical Layer Perspective. *IEEE*. <https://doi.org/10.1109/ANTS.2018.8710133>

- Banafaa, M., Shayea, I., Din, J., Hadri Azmi, M., Alashbi, A., Ibrahim Daradkeh, Y., & Alhammadi, A. (2023). 6G Mobile Communication Technology: Requirements, Targets, Applications, Challenges, Advantages, and Opportunities. *Alexandria Engineering Journal*, 64, 245–274. <https://doi.org/10.1016/j.aej.2022.08.017>
- Beale, M., Uchiyama, H., & Clifton, J. C. (2021). IoT Evolution: What's Next? *IEEE Wireless Communications*, 28(5), 5–7. <https://doi.org/10.1109/MWC.2021.9615126>
- Beltran, J. A., Mudholkar, P., Mudholkar, M., Tripathi, V., Valderrama-Zapata, C., & Lourense, M. (2022). Security Issues and Challenges in Internet of Things (IoT) System. *2022 5th International Conference on Contemporary Computing and Informatics (IC3I)*, 57–60. Uttar Pradesh, India: IEEE. <https://doi.org/10.1109/IC3I56241.2022.10072600>
- Cito, J., Schermann, G., Wittern, J. E., Leitner, P., Zumberi, S., & Gall, H. C. (2017). An Empirical Analysis of the Docker Container Ecosystem on GitHub. *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, 323–333. Buenos Aires, Argentina: IEEE. <https://doi.org/10.1109/MSR.2017.67>
- Dave, M., Doshi, J., & Arolkar, H. (2020). MQTT- CoAP Interconnector: IoT Interoperability Solution for Application Layer Protocols. *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, 122–127. Palladam, India: IEEE. <https://doi.org/10.1109/I-SMAC49090.2020.9243377>
- Dawaliby, S., Bradai, A., & Pousset, Y. (2016). *In depth performance evaluation of LTE-M for M2M communications*. <https://doi.org/10.1109/WiMOB.2016.7763264>
- Díaz, M., Martín, C., & Rubio, B. (2016). State-of-the-art, challenges, and open issues in the integration of Internet of things and cloud computing. *Journal of Network and Computer Applications*, 67, 99–117. <https://doi.org/10.1016/j.jnca.2016.01.010>
- Garg, N. (2013). *Apache Kafka: Set up Apache Kafka clusters and develop custom message producers and consumers using practical, hands-on examples*. Birmingham: Packt Publishing.
- Gbadamosi, S. A., Hancke, G. P., & Abu-Mahfouz, A. M. (2020). Building Upon NB-IoT Networks: A Roadmap Towards 5G New Radio Networks. *IEEE Access*, 8, 188641–188672. <https://doi.org/10.1109/ACCESS.2020.3030653>
- Ge, Y., Liang, X., Zhou, Y. C., Pan, Z., Zhao, G. T., & Zheng, Y. L. (2016). Adaptive Analytic Service for Real-Time Internet of Things Applications. *2016 IEEE International Conference on Web Services (ICWS)*, 484–491. San Francisco, CA, USA: IEEE. <https://doi.org/10.1109/ICWS.2016.69>

- Global IoT and non-IoT connections 2010-2025 | Statista. (2022). Retrieved November 22, 2023, from <https://www.statista.com/statistics/1101442/iot-number-of-connected-devices-worldwide/>
- Gubbi, J., Buyya, R., Marusic, S., & Palaniswami, M. (2013). Internet of Things (IoT): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7), 1645–1660. <https://doi.org/10.1016/j.future.2013.01.010>
- Gugueoth, V., Safavat, S., Shetty, S., & Rawat, D. (2023). A review of IoT security and privacy using decentralized blockchain techniques. *Computer Science Review*, 50, 100585. <https://doi.org/10.1016/j.cosrev.2023.100585>
- Gündoğan, C., Amsüss, C., Schmidt, T. C., & Wählisch, M. (2020, June 16). *IoT Content Object Security with OSCORE and NDN: A First Experimental Comparison*. arXiv. Retrieved from <http://arxiv.org/abs/2001.08023>
- Hanlin, Q., Xianzhen, J., & Xianrong, Z. (2012). Research on Extract, Transform and Load(ETL) in Land and Resources Star Schema Data Warehouse. *2012 Fifth International Symposium on Computational Intelligence and Design*, 120–123. Hangzhou, China: IEEE. <https://doi.org/10.1109/ISCID.2012.38>
- Henneke, D., Elattar, M., & Jasperneite, J. (2015). Communication patterns for Cyber-Physical Systems. *2015 IEEE 20th Conference on Emerging Technologies & Factory Automation (ETFA)*, 1–4. Luxembourg, Luxembourg: IEEE. <https://doi.org/10.1109/ETFA.2015.7301623>
- Hummer, W., Rosenberg, F., Oliveira, F., & Eilam, T. (2013). Testing Idempotence for Infrastructure as Code. In D. Eyers & K. Schwan (Eds.), *Middleware 2013* (pp. 368–388). Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-45065-5_19
- Jaikumar, K., Brindha, T., Deepalakshmi, T. K., & Gomathi, S. (2020). IOT Assisted MQTT for Segregation and Monitoring of Waste for Smart Cities. *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 887–891. Coimbatore, India: IEEE. <https://doi.org/10.1109/ICACCS48705.2020.9074399>
- Jenzeri, D., & Chehri, A. (2022). Data Analysis for IoT System Using 6LoWPAN and Constrained Application Protocol for Environmental Monitoring. *Procedia Computer Science*, 207, 1472–1481. <https://doi.org/10.1016/j.procs.2022.09.204>
- Johnsen, F. T. (2018, September 26). *Using Publish/Subscribe for Short-lived IoT Data*. 645–

649. <https://doi.org/10.15439/2018F232>

- Jouhari, M., Saeed, N., Alouini, M.-S., & Amhoud, E. M. (2023). A Survey on Scalable LoRaWAN for Massive IoT: Recent Advances, Potentials, and Challenges. *IEEE Communications Surveys & Tutorials*, 25(3), 1841–1876. <https://doi.org/10.1109/COMST.2023.3274934>
- Khashan, O. A., & Khafajah, N. M. (2023). Efficient hybrid centralized and blockchain-based authentication architecture for heterogeneous IoT systems. *Journal of King Saud University - Computer and Information Sciences*, 35(2), 726–739. <https://doi.org/10.1016/j.jksuci.2023.01.011>
- Kumari, P., & Jain, A. K. (2023). A comprehensive study of DDoS attacks over IoT network and their countermeasures. *Computers & Security*, 127, 103096. <https://doi.org/10.1016/j.cose.2023.103096>
- Laroui, M., Nour, B., Mounsla, H., Cherif, M. A., Afifi, H., & Guizani, M. (2021). Edge and fog computing for IoT: A survey on current research activities & future directions. *Computer Communications*, 180, 210–231. <https://doi.org/10.1016/j.comcom.2021.09.003>
- Lazidis, A., Tsakos, K., & Petrakis, E. G. M. (2022). Publish–Subscribe approaches for the IoT and the cloud: Functional and performance evaluation of open-source systems. *Internet of Things*, 19, 100538. <https://doi.org/10.1016/j.iot.2022.100538>
- Li, C., Wang, J., Wang, S., & Zhang, Y. (2024). A review of IoT applications in healthcare. *Neurocomputing*, 565, 127017. <https://doi.org/10.1016/j.neucom.2023.127017>
- Li, W., Wang, Y., & Li, J. (2022). Enhancing blockchain-based filtration mechanism via IPFS for collaborative intrusion detection in IoT networks. *Journal of Systems Architecture*, 127, 102510. <https://doi.org/10.1016/j.sysarc.2022.102510>
- Makhdoom, I., Abolhasan, M., Lipman, J., Liu, R. P., & Ni, W. (2019). Anatomy of Threats to the Internet of Things. *IEEE Communications Surveys & Tutorials*, 21(2), 1636–1675. <https://doi.org/10.1109/COMST.2018.2874978>
- Malek, Y. N., Kharbouch, A., Khoukhi, H. E., Bakhouya, M., Florio, V. D., Ouadghiri, D. E., ... Blondia, C. (2017). On the use of IoT and Big Data Technologies for Real-time Monitoring and Data Processing. *Procedia Computer Science*, 113, 429–434. <https://doi.org/10.1016/j.procs.2017.08.281>

- Mammela, O., Karhula, P., & Makela, J. (2019). Scalability Analysis of Data Transfer in Massive Internet of Things Applications. *2019 IEEE Symposium on Computers and Communications (ISCC)*, 1–7. Barcelona, Spain: IEEE. <https://doi.org/10.1109/ISCC47284.2019.8969722>
- Marcozzi, M., Gemikonakli, O., Gemikonakli, E., Ever, E., & Mostarda, L. (2023). Availability evaluation of IoT systems with Byzantine fault-tolerance for mission-critical applications. *Internet of Things*, 23, 100889. <https://doi.org/10.1016/j.iot.2023.100889>
- Medileh, S., Laouid, A., Nagoudi, E. M. B., Euler, R., Bounceur, A., Hammoudeh, M., ... Khashan, O. A. (2020). A flexible encryption technique for the internet of things environment. *Ad Hoc Networks*, 106, 102240. <https://doi.org/10.1016/j.adhoc.2020.102240>
- Pei, M., Tschofenig, H., Atyeo, A., Cook, N., & Yoo, M. (2016). *Open Trust Protocol (OTrP)*.
- P.K., G., S.S., K., A.M., K., S., R., A.H., M. R., & S., M. (2022). *Human-Machine Interaction and IoT Applications for a Smarter World* (First edition). CRC Press.
- Pokhrel, S. R., Ding, J., Park, J., Park, O.-S., & Choi, J. (2020). Towards Enabling Critical mMTC: A Review of URLLC Within mMTC. *IEEE Access*, 8, 131796–131813. <https://doi.org/10.1109/ACCESS.2020.3010271>
- Rahim, Md. A., Rahman, Md. A., Rahman, M. M., Asyhari, A. T., Bhuiyan, Md. Z. A., & Ramasamy, D. (2021). Evolution of IoT-enabled connectivity and applications in automotive industry: A review. *Vehicular Communications*, 27, 100285. <https://doi.org/10.1016/j.vehcom.2020.100285>
- Rahman, M. S., & Das, R. K. (2022). RTID: On-demand real-time data processing for IoT network. *Materials Today: Proceedings*, 62, 4721–4725. <https://doi.org/10.1016/j.matpr.2022.03.168>
- Sai Lohitha, N., & Pounambal, M. (2023). Integrated publish/subscribe and push-pull method for cloud based IoT framework for real time data processing. *Measurement: Sensors*, 27, 100699. <https://doi.org/10.1016/j.measen.2023.100699>
- Seymour, M. (2021). *Mastering Kafka Streams and ksqlDB*.
- Shapira, G., Palino, T., Sivaram, R., & Petty, K. (2022). *Kafka: The definitive guide: real-time data and stream processing at scale* (Second edition). Beijing Boston Farnham

Sebastopol Tokyo: O'Reilly.

- Shapira, G., Palino, T., Sivaram, R., & Petty, K. (n.d.). *Kafka: The Definitive Guide* (Second). O'Reilly Media, Inc.
- Shehada, D., Gawanmeh, A., Yeun, C. Y., & Jamal Zemerly, M. (2022). Fog-based distributed trust and reputation management system for internet of things. *Journal of King Saud University - Computer and Information Sciences*, 34(10), 8637–8646. <https://doi.org/10.1016/j.jksuci.2021.10.006>
- Shi, Y., Zhang, Y., & Chen, J. (2020). Cross-layer QoS enabled SDN-like publish/subscribe communication infrastructure for IoT. *China Communications*, 17(3), 149–167. <https://doi.org/10.23919/JCC.2020.03.013>
- Siddiqui, H., Khendek, F., & Toeroe, M. (2023). Microservices based architectures for IoT systems—State-of-the-art review. *Internet of Things*, 23, 100854. <https://doi.org/10.1016/j.iot.2023.100854>
- Stusek, M., Zeman, K., Masek, P., Sedova, J., & Hosek, J. (2020). IoT Protocols for Low-power Massive IoT: A Communication Perspective. *IEEE*. <https://doi.org/10.1109/ICUMT48472.2019.8970868>
- Taj, S., Imran, A. S., Kastrati, Z., Daudpota, S. M., Memon, R. A., & Ahmed, J. (2023). IoT-based supply chain management: A systematic literature review. *Internet of Things*, 24, 100982. <https://doi.org/10.1016/j.iot.2023.100982>
- Tawakuli, A., & Engel, T. (2022). Towards Normalizing The Design Phase of Data Preprocessing Pipelines For IoT Data. *2022 IEEE International Conference on Big Data (Big Data)*, 4589–4594. Osaka, Japan: IEEE. <https://doi.org/10.1109/BigData55660.2022.10020312>
- Verma, Shikhar, Kawamoto, Y., Fadlullah, Z. Md., Nishiyama, H., & Kato, N. (2017). A Survey on Network Methodologies for Real-Time Analytics of Massive IoT Data and Open Research Issues. *IEEE Communications Surveys & Tutorials*, 19(3), 1457–1477. <https://doi.org/10.1109/COMST.2017.2694469>
- Verma, Shobhit, Sharma, N., Singh, A., Alharbi, A., Alosaimi, W., Alyami, H., ... Goyal, N. (2022). DNNBoT: Deep Neural Network-Based Botnet Detection and Classification. *Computers, Materials & Continua*, 71(1), 1729–1750. <https://doi.org/10.32604/cmc.2022.020938>
- Wang, F., & Ma, G. (2019). *Massive Machine Type Communications: Multiple Access Schemes*. Cham: Springer International Publishing. <https://doi.org/10.1007/978-3-030->

13574-4

- Yang, C., Puthal, D., Mohanty, S. P., & Kougianos, E. (2017). Big-Sensing-Data Curation for the Cloud is Coming: A Promise of Scalable Cloud-Data-Center Mitigation for Next-Generation IoT and Wireless Sensor Networks. *IEEE Consumer Electronics Magazine*, 6(4), 48–56. <https://doi.org/10.1109/MCE.2017.2714695>
- Youn, J., Park, J., Kim, S., You, C., & Cho, S. (2021). A Study of Random Access for Massive Machine-type Communications: Limitations and Solutions. *2021 IEEE Region 10 Symposium (TENSYP)*, 1–4. Jeju, Korea, Republic of: IEEE. <https://doi.org/10.1109/TENSYP52854.2021.955099>