A study of the naming convention used for topics in distributed MQTT brokers' environment

Sheetal Pravin Raut

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Intelligent Systems)

Supervisor: Prof. Dr Stefan Weber

August 2022

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Sheetal Pravin Raut

August 18, 2022

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Sheetal Pravin Raut

August 18, 2022

A study of the naming convention used for topics in distributed MQTT brokers' environment

Sheetal Pravin Raut, Master of Science in Computer Science

University of Dublin, Trinity College, 2022

Supervisor: Prof. Dr Stefan Weber

The world revolves around the Internet and accessing information through the Internet. The traditional Internet infrastructure uses a host-centric paradigm. There are various disadvantages that come into notice after different research has been conducted. So, a new paradigm for the future Internet has been introduced that is away from the host-centric paradigm and provides perpetual connectivity, an end-to-end principle. This paradigm is Information-centric networking (ICN). In this type of network architecture focal point is identified information or content or data.

In 2021, there were more than 10 billion IoT devices active. Also, It has been estimated that this number could grow up to 25.4 billion in 2030. These IoT devices use Message Queuing Telemetry Transport (MQTT) protocol for communication purposes. MQTT has been based on the Publish-Subscribe Internet Routing Paradigm (PSIRP), Which ideally a one of ICN architecture.

ICN architecture maps a name to content. That is why naming plays an important role in terms of routing. This research focuses on studying the advantages and disadvantages of different naming conventions in distributed MQTT broker architecture and also observing their effects on naming conventions in different topologies. It involves the implementation of the different naming conventions, such as a hierarchical naming convention and a hybrid naming convention which could be used in real-time scenarios in distributed MQTT broker. As mentioned earlier, It involves studying the effect of this naming convention on different topologies. In terms of topology, namely chaining, i.e. hierarchical topology and clustering, topology took into consideration. The evaluation has been conducted between the topology and naming convention with the help of different evaluation attributes.

Acknowledgments

I would like to thank Prof. Dr Stefan Weber for supervising and providing valuable guidance and motivation throughout the journey of the dissertation. I would also like to thank my father, Pravin, and mother, Anjali, for providing unconditional support and encouragement throughout my course at Trinity.

Sheetal Pravin Raut

University of Dublin, Trinity College
August 2022

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In today's world, every device around us is getting smarter by connecting them to the Internet and making them accessible from anywhere, by using their data in order to perform some action. For example, In the case of smart home automation use case, the light of the room gets turned on whenever a person enters the room, where the presence of any person is detected using the motion sensor that in turn sends the command to the light sensor to turn on. This is possible because of the Internet of Things(IoT). Because of that, devices could connect to the Internet, transmit their data, and can communicate with other sensors, and take action based on conditions met.

In 2021, there were more than 10 billion active IoT devices observed. It has been estimated that this number will grow exponentially and surpass 24.4 billion in 20230.Jovanovic (2022) As this number talks, the IoT is the future of the Internet and evolving domain.

1.1 Motivation and Aim

Current Internet architecture is based on a host-centric paradigm. It uses Transmission Control Protocol/Internet Protocol (TCP/IP) networking model. This paradigm has been successful in terms of addressing and naming schemes to date. Nowadays, smart devices and sensors get connected to the Internet and of which in the future, the number of devices connecting to the Internet will grow exponentially. The current network architecture will not be able to provide support for addressing billions of devices and contents since IPV4 has limited address space. Also, IPV6 addresses are too long and not suitable to use with resource-constrained devices.

The current network paradigm has some shortcomings, and it will not be suitable for the coming future of the Internet. A few of the shortcomings are discussed below.

Persistent, location-independent naming: Current Internet architecture lacks in terms of persistence and location-independent naming for the content. In the case of current Internet architecture, a URL is used to locate the service or resource. It uses the URL for IPV4 mapping. In short, it is mapping directly to a specific location. This does not help to achieve location-independent mapping, for example, moving a website to another domain. Then the original website will become unreachable.

Security: As current Internet architecture uses a host-centric paradigm, the security is based on securing channels between hosts via encryption such as transport layer security(TLS). Also, the client needs to rely on trusted third parties for data integration.

Mobility and multihoming: Current Internet architecture resolves around the host, which is suitable for static devices. But in terms of mobility and multihoming, this architecture has become more complex to deal with. Solutions such as mobile IP addresses have been designed to address this issue; however, this approach suffers from some disadvantages such as increased stretch and complexity due to problems inherited from the host-centric approach.

To address these issues listed above new paradigm has been researched and proposed. This approach is known as Information-Centric Network (ICN) paradigm. As the name implies, this approach resolves around content naming. It uses content names for Network data object mapping.

As mentioned, this architecture revolves around naming the content used in the network. There are various naming schemes available such as hierarchical, flat-based, attribute-value- based, and hybrid naming schemes. Each naming scheme has its own benefits and drawbacks in terms of security, scalability, reliability, and aggregation. ICN approach provides various architectures, which most commonly known are Data-Oriented Network Architecture (DONA), Named Data Networking (NDN), Content-Centric Networking (CCN) and Publish-Subscribe Internet Technology (PURSUIT). In terms of IoT publisher-subscribe model is most commonly used in order to transmit data between devices. MQTT uses the same model for transmitting data.

It is the point of interest and research to study various naming schemes in detail about their advantages and disadvantages over other naming schemes in terms of security, scalability, aggregation, and lookup performance. Also, for IoT networks, how these naming schemes would perform better in order to provide a more secure approach to name data as well as work well with resource-constrained devices?

The aim of this dissertation is

- To study the different naming schemes, their benefits, and drawbacks.
- To design the naming schemes for useful real-time IoT use-case.
- Implement the designed naming schemes in the simulated environment of distributed MQTT brokers.
- Evaluate the performance of those naming schemes in various network topologies of distributed MQTT brokers.

1.2 Map

This dissertation is structured into six sections, each having its own focus area as described below.

Chapter 1: Introduction This chapter gives an introduction to the Information-Centric Network (ICN). It outlines the motivation and the aim behind this research and the map for the dissertation.

Chapter 2: State Of Art This chapter gives an overview of background research conducted as well as critiques of previously published and implemented related work reviewed in order to design and implement the solution for the problem statement of the dissertation.

Chapter 3: Problem Statement This chapter gives a description of the problem statement for which the research is done as a part of the thesis.

Chapter 4: Design This chapter gives a detailed description of which designs are used to research the problem statement. It also provides the diagram of different distributed MQTT brokers topologies that are implemented further.

Chapter 5: Implementation This chapter talks about the list of tools and their specification used for the implementation of the design discussed in the design chapter. It also specifies the challenges faced during the implementation.

Chapter 6: Evaluation This chapter explains the experiments conducted to evaluate the designs implemented as a part of the thesis. Results of those experiments and detailed evaluation with appropriate evaluation attributes are presented in this chapter.

Chapter 7: Conclusion and Future Work Lastly, this chapter gives an insight into the conclusions made from this thesis as well as the scope to which it can be extended further.

Chapter 2

State of the Art

This Section provides a detailed description of background research conducted related to ICN, MQTT, and many more. It also includes the details of related work been critiqued that has paved the way to the problem statement of this dissertation.

2.1 Background

This Section describes in detail the background information about the Information-centric networking, their architectures, MQTT protocol, its packet format, and QoS levels supported. It also summarizes the comparison of MQTT with other similar protocols in terms of benefits and drawbacks. Also, the detailed description of various naming schemes, their benefits, and drawbacks.

2.1.1 Information-Centric Networking

In recent times, more smart devices and sensors are getting connected to the Internet. This number of devices is growing exponentially day by day. These devices could be a static or mobile device that senses the data from their surrounding and forward it in order to communicate with other devices present in the network. This network ecosystem is referred to as the IoT.Lin et al. (2017) Most IoT applications have seen used a content-oriented naming approach. Sensor or devices publishes data, and the user receives data as per the subscription to data, user requests sent for specific data of interest, or actuators perform an action as per the received command.

ICN is a new communication paradigm that has been researched to replace the current IP-based networking.Xylomenos et al. (2014) It uses data-oriented nature for communication. The main element which drives the communication is content names rather than the IP addresses of the content or service provider. Because of this, ICN supports location-

independent naming by decoupling it from content.Din et al. (2018) It also provides support for multihoming and mobility. ICN revolves around content naming approaches, and the efficiency, scalability, and security of this networking are directly linked to which naming approach has been used for content naming. In terms of having effective naming schemes, various factors of content naming should take into consideration, such as device naming, length of names, the structure of names, name aggregation, security, meta-data, and many more.

For the ICN concept, different architecture has been implemented to support the content naming. DONA is one of the ICN architectures which uses flat naming schemes to address the content. This flat-based naming scheme is self-certifying. In this naming scheme, responsibility for authenticity and availability is separated because the DONA name handles persistence and authenticity while name resolution handles availability. This naming format uses the structure P:L, where P denotes the public key of the producer, which is cryptographically hashed using the hash function, and L denotes the label chosen by the principal.Koponen et al. (2007)

Another architecture Of ICN is NDN and CCN; these use hierarchical humans readable names to identify content.CCN is a pull-based communication model in which the service subscriber gets the data irrespective of the location of the content provider. CCN routers use 3 data structures in order to route the data, namely Pending Interest Task, Content Store, and Forwarding Information Base(FIB).Arshad et al. (2018) PURSUIT is another architecture of ICN, which is discussed in detail below.

PURSUIT Architecture

The Publish-Subscribe Internet Technology project (PURSUIT) CORDIS (2022) is the continuation of the Publish-Subscribe Internet Routing Paradigm (PSIRP) project.PSIRP (2022) The both of PSIRP and PURSUIT are funded by the EU Framework 7 Program. PURSUIT project is a clean-slate ICN architecture that supports a complete publish/-subscribe stack. The PURSUIT architecture consists of three components, namely rendezvous, Topology Management (TM), and Forwarding Node (FN), which work together to form name-based networking architecture.

PURSUIT uses self-certifying flat naming schemes that consist of a pair of multiple Scope IDs (SIDs) and a single Rendezvous ID(RID). Each group is assigned a SID, and each content object has its own RID. PURSUIT decouples the name resolution and routing, as these functionalities are performed by separate modules. Additionally, PSIRP employs forwarding IDs employed by the forwarding fabric to move data once contact has been made at a rendezvous point. The forwarding IDs (Bloom filters in LIPSIN Jokela

et al. (2009)) are temporary and define a path from the publisher to the subscriber.

Figure. 2.1 Ahlgren et al. (2012) Shows the steps by step process of publishing data and subscribing for data.

Step 1: The publication belongs to a particular named scope.

Step 2: Subscriber subscribes to the name data object.

Step 3: Rendezvous system matches the publications and subscriptions.

Step 4: The subscription request consists of the scope identifier (SI) and the rendezvous identifier (RI) that identify the content. These identifiers are used in matching procedure results in forwarding identifiers.

Step 5-7: Data packet has been forwarding data. The forwarding identifier consists of a Bloom filter these routers use for selecting the interfaces to which to deliver content. Because of this, routers do not need to keep track of the forwarding state.

In PURSUIT, clients can unsubscribe, switch networks, and resubscribe again. Content provider mobility is more complex as it involves updating the routing state in the rendezvous nodes.

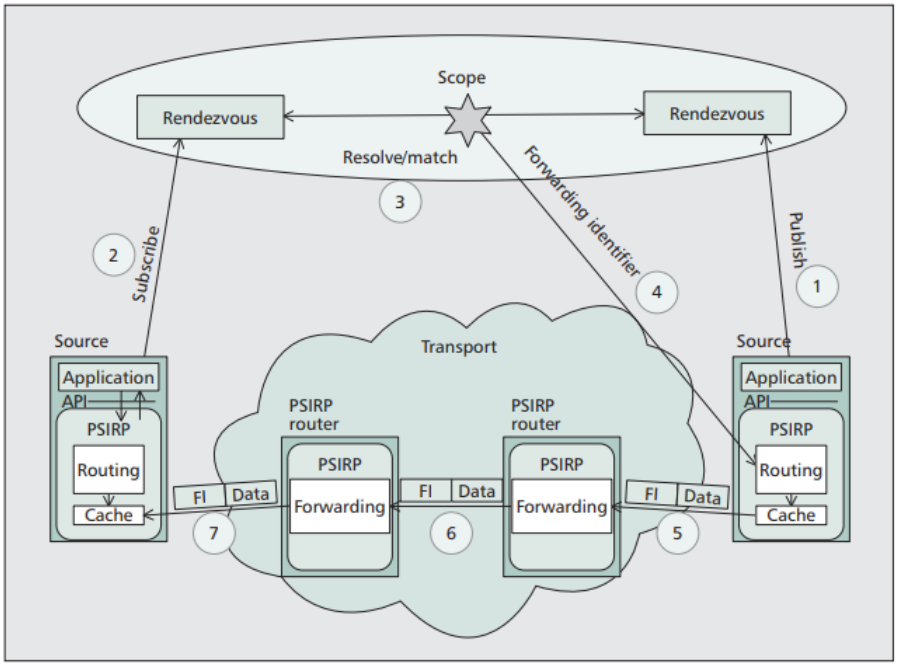

Figure 2.1: PURSUIT architecture overview

2.1.2 Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight messaging protocol introduced by IBM in 1999. This messaging protocol is designed for resource-constrained and limited devices. Gupta and Prabha.

(2021) MQTT uses the publisher-subscriber model for message communication that uses Transmission Control Protocol /Internet Protocol (TCP/IP) protocol.

This design uses Publisher-subscriber architecture, using which it can publish the data to the MQTT broker and subscribe for data from the MQTT broker. The MQTT broker architecture consists of three important components, as shown in Figure. 2.2.

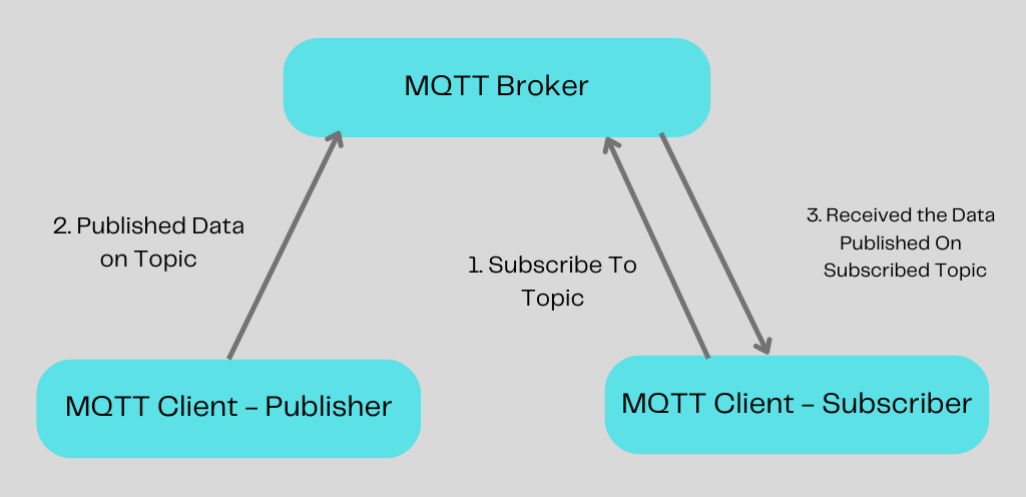

Figure 2.2: MQTT broker architecture

MQTT Client: MQTT client is either publisher or subscriber. In the case of the publisher, the MQTT client will publish the data to the MQTT broker using the topic name. Otherwise, the MQTT client will subscribe to the topic as per interest, and in the future, it can receive data that got published on the subscribed topic. In general, the publisher is the sensor that publishes data such as temperature, humidity, heart beats rate, or current location, or it could be the actuators sending their current status. Whereas, subscribers could be dashboard to monitor data or take further action taken on actuators based on conditions met on received sensors data.

MQTT Broker: MQTT Broker is considered the heart of the Publisher-subscribe protocol. It is responsible for receiving all messages published, filtering the messages, determining who is subscribed to each topic, sending messages to the subscriber based on subscription, and managing sessions of MQTT clients.

Topic: Topic refers to a UTF-8 string. Topics have been used by the MQTT brokers to filter messages for each MQTT client connected to it. As in Publisher-subscribe architecture it uses topic-based naming. It is important to agree upon the structure of the topic name used by the publisher and subscriber before implementation.

MQTT protocol is a push-based protocol. It helps to create service models such as Action-based control commands and Event-Driven Service Models.

Action-Based Control Command Service Model

This model is possible to implement with MQTT. MQTT client can send an action to be performed on the actuator with the topic naming. For example, In the case of a Smart building automation system, the MQTT client wants to turn on the light of the hallway for floor 2 in the building Red Mills. Then MQTT client will send a command as "On" to the respective topic to represent the same location. In this case, the MQTT client sends a command to the MQTT broker, which in turn sends the command to the subscriber to that topic. That, in turn, performs the action provided on the actuator present.

Event-Driven Service Model

This model is useful when data needs to be published in case of a particular event. For example, Smart Building automation system, the lights of the hallway needs to get turned on whenever a moment in hallway passage is detected. So that whenever the motion sensor detects any movement will send a notification to the MQTT broker. Then after receiving a notification on that specified topic, it will send an action command to turn on the light actuator.

MQTT Packet Format

The MQTT protocol exchanges a series of control packets in a defined way to work. This Section describes the format of these packets. An MQTT control consists of three parts, namely a Fixed header, variable header, and payload.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
Byte2 ...	Remaining Length							

Table 2.1: MQTT control packet : Fixed Header format

The fixed header is present in all types of MQTT control packets. This packet contains the MQTT control packet type; this is a 4-bit unsigned value. A few of them are listed below in Table.2.2

Name	Value	Direction of flow	Description
CONNECT	1	Client to Server	Connection request
CONNACK	2	Server to Client	Connect acknowledgment
PUBLISH	3	Client to Server or Server to Client	Publish message
PUBACK	4	Client to Server or Server to Client	Publish acknowledgment (QoS 1)
SUBSCRIBE	8	Client to Server	Subscribe request
SUBACK	9	Server to Client	Subscribe acknowledgment

Table 2.2: MQTT control packet type value and description

The remaining bits[3-0] of Byte 1 in the fixed header contain the flags that are specific to each MQTT Control Packet type. The Remaining length is a variable byte integer that represents the number of bytes remaining within the current control packet that includes the data in the variable header and the payload. Whereas variable header and payload are part of the MQTT control packet and present in some of the types of MQTT control packets. The two important MQTT control packets, PUBLISH and SUBSCRIBE, are discussed in detail below.

MQTT Control Packet PUBLISH

A PUBLISH packet is sent from a Client to a Server or from a Server to a Client to transport an Application Message. Table. 2.3 shows the fixed header packet format for PUBLISH MQTT control packet.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type (3)				DUP flag	QoS level		RETAIN
	0	0	1	1	X	X	X	X
Byte2 ...	Remaining Length							

Table 2.3: MQTT PUBLISH control packet : Fixed header packet format

Flags used in MQTT PUBLISH control packet fixed header, PUBLISH variable header and PUBLISH payload description are given below.

- **DUP:** This flag represents whether the packet that has been transmitted is a duplicate or not. If set to 0 means its first occurrence of sending a packet. Otherwise, it is a re-delivery of an earlier attempt to send the packet.
- **QoS:** This field is Quality of Service that provides the level of assurance for the delivery of an application message. This could be 0, 1 or 2.

- **RETAIN:** If the RETAIN flag is set to 1 in a PUBLISH packet sent by a Client to a Server, the Server MUST replace any existing retained message for this topic and store the Application Message.
- **PUBLISH variable header:** This is consist of topic name, packet identifier, and properties.
- **PUBLISH payload:** The payload contains the application message that is being published. The content and format of the application are specific.

MQTT Control Packet SUBSCRIBE

The SUBSCRIBE packet is sent from the client to the Server to create one or more subscriptions. Each Subscription registers a Client's interest in Topics. The Server sends PUBLISH packets to the client to forward Application Messages that were published to Topics that match these subscriptions. Table. 2.4 shows the fixed header packet format for SUBSCRIBE MQTT control packet.

Bit	7	6	5	4	3	2	1	0
Byte 1	MQTT Control Packet type (8)				Reserved			
	1	0	0	0	0	0	1	0
Byte2 ...	Remaining Length							

Table 2.4: MQTT SUBSCRIBE control packet: Fixed header packet format

- **Remaining length field:** contains the length of the variable header plus the length of the payload, encoded as a Variable Byte Integer.
- **SUBSCRIBE Variable header:** The Variable Header of the SUBSCRIBE packet contains the following fields in the order: Packet Identifier and Properties.
- **SUBSCRIBE Payload:** The Payload of a SUBSCRIBE packet contains a list of Topic Filters indicating the Topics to which the client wants to subscribe. The Topic Filters MUST be a UTF-8 encoded string.

MQTT supports the three types of Quality of Service(QOS) levels.

QoS 0: This level is referred to as at most once or fire and forget. This level provides a very simple and unacknowledged service. Publisher publishes data to MQTT

broker using PUBLISH packet. Further MQTT broker sends the data packet to the subscriber subscribed to that topic on which data has been published. In this case, none of the MQTT clients receives any acknowledgment.

QoS 1: This level is referred to at least once. This level is acknowledgment service. This QoS level uses a PUBLISH/PUBACK packet sequence between the publisher and its broker. Also, between broker and subscribers. This acknowledgment packet provides confirmation that content has been received. It also uses a retry mechanism in which the retry mechanism will send the content again if an acknowledgment did not receive in a certain time. This will cause a duplicate copy of the data packets received. Figure. 2.3 Kurdi and Thayananthan (2022) show the MQTT control packet transmitted between publisher, subscriber and MQTT broker.

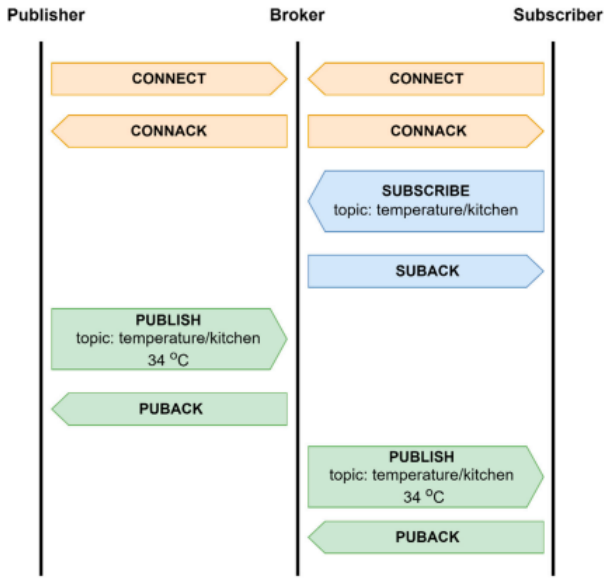

Figure 2.3: MQTT control packet transmission between MQTT client and broker

QoS 2 : This level is referred to exactly once. This level is assured service. This QoS level uses messages PUBLISH/PUBREC and PUBREL/PUBCOMP. This level ensures that the message will be delivered exactly once, and no duplicate copy of data will be delivered.

There is various protocol that exists as competing protocol to MQTT, such as Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), Simple/Streaming Text Oriented Messaging Protocol (STOMP), and Simple Media Control Protocol (SMCP).MQTT protocol has benefits and drawbacks as compared to these protocols. Those are listed below.

Benefits of MQTT

1. MQTT is a lightweight protocol that provides fast and efficient message delivery.
2. MQTT provides message security using Transport Layer Security/Secure Socket Layer (TLS/SSL) encryption. In MQTT, standard port 1883 is used for non-encrypted communication and 8883 for encrypted communication using TLS/SSL.
3. MQTT protocol provides message reliability because it uses TCP/IP protocol for communication. Unlike UDP, TCP ensures that data is not damaged, lost, or delivered out of order to a receiving process.
4. MQTT consumes less power for communication, so it is a good option for connecting low-power resource devices.

Drawbacks of MQTT

1. MQTT protocol is comparatively slow to COAP as it uses content-oriented TCP/IP protocol.
2. MQTT broker has limited storage.
3. MQTT broker is the central point of communication for publishers and subscribers. Because of the failure of the MQTT broker, subscribers and publishers will not be able to communicate. That's a reason why MQTT broker is the single point of failure.

2.1.3 Naming Conventions

In the case of ICN architecture, everything resolves around naming the content. To name the content, different kinds of schemes are supported. Each naming scheme has its own ability to scale, security and aggregate. These naming schemes can be categorized into Hierarchical, Flat, Attribute- value-based, and Hybrid.

Hierarchical Naming Convention

Hierarchical naming schemes use structural format, and they have human-readable names. Ghodsi et al. (2011) These names consist of several components to detect the components and services. Its structure is similar to a Uniform resource locator(URL). Component and subcomponent in the names are separated by each other using slash (/). This naming scheme is adopted by ICN architecture like CCN and NDN.Arshad et al. (2018)

Flat-Based Naming Convention

Flat naming schemes have self-certifying names, which are acquired by applying the hash function to the content or its subcomponent or its attribute. These names are not human-friendly.Tanenbaum and Steen (2006) Flat names are not scalable, and it requires another service to translate self-certifying names and human-readable names.Ghodsi et al. (2011) Flat naming schemes adopted by DONA, MobilityFirst, and NetInf.Arshad et al. (2018). This naming scheme assures location independence, application independence, and global uniqueness. It slow downs the aggregation mechanism , which increases the size of the routing table and entries.

Attribute-Based Naming Convention

Attribute-value naming scheme has a list of attributes. Each attribute is composed of a type, name, and set of values. These values could be generated by date or time, version, location, and many more. This naming allows different kinds of information to reside within the names. The search process is easy in this naming scheme because it allows search using known keywords for content, but there does not uniquely identify the content.

Hybrid Naming Convention

Hybrid naming schemes combine two or three naming schemes mentioned above. The main idea behind this is to combine the advantages of each naming scheme to achieve more scalability, efficiency, and security. Each naming scheme have its own strength and weaknesses, where hierarchical naming schemes supports name aggregation, flat naming schemes have the advantage of having fixed length names, and attribute value-based naming scheme provides ease in the search process. A hybrid naming scheme incorporates more than one of these to provide the best feature.

These naming schemes discussed above, their examples, benefits and drawbacks are tabularized in Table. 2.5

2.2 Related Work

This Section provides a detailed description of the previous work done in terms of using hierarchical and hybrid naming conventions in different ICN architectures.

Hierarchical Naming Schemes

As proposed in Shang et al. (2014), hierarchical naming schemes have been designed for secured building management systems for the UCLA campus. This naming scheme

Naming Type	Example	Benefits	Drawbacks
Hierarchical Naming	https://www.oasis-open.org/committees/download.php	1. Human readable names 2. Names designed using a structural format 3. Supports Name aggregation 4. Enhanced network scalability	1. Requires Standardization 2. Variable length of component name; sometimes names may be too lengthy
Flat Naming	aui://ls.edu.dn/sha256(title)	1. Fixed length of names. 2. Names are easy to generate	1. No structure and semantic to generate name 2. Does not support name aggregation 3. Difficult to generate name for dynamic content
Attribute-value based naming	Name <str>:'Title' Document Name <Str>:'Document1' Publication Year <int>: 2019	1. Easy search process 2. Support to transmit additional content information	1. Unique Naming is difficult 2. Different contents attributed by same name

Table 2.5: Naming schemes benefits and drawbacks

is used in Named Data Networking (NDN) architecture. The sample of this designed naming scheme is
"/ndn/ucla.edu/bms/building/melnitz/studio/1/data/ panel/J/voltage /<timestamp>"
This format uses room prefix, building name, and then on a more granular level, the location. It has a separate naming tree to get the user key for encryption of data and also access privilege list available for the user. This provided naming scheme separates out the data namespace and user namespace.

As proposed in Amadeo et al. (2015) paper, NDOMUS (Named Data netwOrking for sMart home aUtomation) is a hierarchical naming scheme for smart home automation. This naming scheme designed with structure as */homeID/task/type/subtype/location/*
In this naming format, the type component specifies the task type (sensing or action); the subtype describes which specific task to be performed, and the location identifies the physical location (sensors or actuators). The approach proposed uses the COAP protocol for transmitting messages between clients.

Piro et al. (2014) propose a hierarchical naming scheme designed for NDN-based platform for smart cities. As per discussed in the paper, it provides a secure naming approach by using the format *"/domain/main-service/sub-categories-of-services/type-of-question/additional-info"* The domain, main-service, and sub-categories-of-services are similar to other approaches to specify the service using a hierarchy of naming. Additionally, the type-of-question component provides information about the type of user demand. The value for this field could be discovered, reserve, query, and communicate, which represents that the user wants to discover a node offering a given service, reserve a service, interrogate a remote node, and communicate something to another device, respectively. This model discussed uses a service discovery and content delivery approach. The drawback of this approach is that it employs long names which would cause extra overhead for searching or lookup processing.

Mochida et al. (2017) proposed a naming scheme for weather monitoring in smart cities. It uses a machine learning approach to generate naming from passed queries. This is implemented with CCN. The sample of this naming scheme is *"dinfo/Japan/Tokyo-to/Shinjuku-ku/kabukicho/1Chrome-4-1/net2/35.6938401/139.7035494/42933"* It specifies the type of message followed by geographical name using country, city, and local street name, later followed by latitude, longitude, and timestamp. These names generated are long names, which will be difficult to adopt in any IoT environment also, will further slow down the searching process and aggregation process.

Hybrid Naming Schemes

As proposed in Arshad et al. (2018), A hybrid naming scheme is designed, which

consists of hierarchical and flat-based naming schemes for a smart campus system. These naming schemes are used in Content-Centric Network (CCN) architecture. The format of the designed naming scheme is *"DomainName/Location/Task:DeviceName|Data"* The data and device names are hashed using the FNV-1a hash function. This design provides benefits such as high aggregation, fixed length of naming, scalable naming, and security provided because of hashing.

Bouk et al. (2017) proposed a hybrid naming scheme consisting of hierarchical and attribute value-based naming that has been designed for managing traffic in the smart city. The format of the naming scheme is *"SC://service-id/spatio_and_or_temporal_scope .../attribute:value/"* This approach provides security by using asymmetric data encryption. These names are used to know the conditions of traffic and information about a passenger.

Arshad et al. (2017) proposed the NDN-HNS the hybrid naming approach for IoT smart campuses. This name consists of four parts. First, the root prefix indicates the type of IoT application, such as smart homes, smart cities, and smart transport. Secondly, the hierarchical component represents the campus information such as the campus name, campus sub-name, campus location, and many more. Thirdly the attribute-value component contains an attribute and value for the type of task and sub-task, such as task type, Task-sub type. Lastly, the flat-based component provides the secured and signed name using the SHA256 hash function. This naming scheme is a combination of three naming schemes; names formed using these schemes are very long. This approach is not much scalable. Also, it is challenging to achieve a fast lookup.

As proposed in Nour et al. (2020), the Multilayer Multicomponent Hierarchical Attribute-value (M2HAV) naming for wireless devices. The designed name format is
"root prefix: Task type: Service type: Network Function: Topological Location"
Each component, apart from the root prefix, supports the setting values to the set of properties using different attributes. It provides the benefit of short lengths with the help of location encoding using a prefix-based labelling scheme and variable-length encoding scheme.

As proposed in Rehman et al. (2019) Name-Integrated Query (NINQ) framework provides a hybrid naming scheme implemented in NDN for a smart building. This naming consists of three components, Hierarchical component, hashing-based flat component, and query component. The sample format of this naming is *"Unique location identifier/sublocation/floor|flat component|Query : query command"* The example of query could be WHERE_DATA_GTE_25, and the example of query command could be SET_DEVICE_ON.

The naming scheme proposed in Javaheri et al. (2020) for PURSUIT architecture.

This naming scheme is designed for the use case of the smart city. This design addresses two domains traffic on the street and services present in that room of the building. It uses a hybrid naming approach that consists of three components, namely attributes value, hierarchical and flat component. The sample naming format of this naming is
For traffic status *"#traffic | district / street / time: (traffic status)"*
For services *"#services | building / floor / room id: (device name)"*

All papers mentioned above, except Javaheri et al. (2020), proposed and implemented various naming schemes, but they either used CCN or NDN architecture. Also, as the naming scheme proposed in Javaheri et al. (2020), implementation of this naming scheme is not possible with MQTT as MQTT topics are not allowed to have '#' in that. As researched, no research has been conducted to evaluate the performance of naming schemes based on the underlying nodes or brokers' network topology.

2.3 Summary

This Section provides the summary of all related work referred to. Referred paper, used naming schemes, and remarks about that approach have been tabularized in Table. 2.6

Related Paper	**Objective**	**Hierarchical naming**	**Flat based naming**	**Attribute-value Based naming**	**ICN architecture**	**Remarks**
Shang et al. (2014)	ICN-based naming for Securing building management systems	Yes	No	No	NDN	Additional security provided with help of user identification also encrypted data transmitted.
Amadeo et al. (2015)	NDOMUS (Named Data netwOrking for sMart home aUtomation)	Yes	No	No	NDN	Names are long in length but very specific. This approach has used COAP protocol for transmitting messages.

Piro et al. (2014)	IoT based naming for smart cities	Yes	No	No	NDN	The hierarchical naming component it uses a type-of-question component that supports discovery of a node given a provided service, reserve to reserve a service. It uses extra long name, which lead to overhead in searching process.
Mochida et al. (2017)	IoT naming scheme for smart cities for weather monitoring	Yes	No	No	CCN	It uses a machine learning approach to generate names, generated names are long in length.
Arshad et al. (2018)	IoT based naming scheme for smart campus system	Yes	Yes	No	CCN	It provides secure names hased using FNV-1a hash function
Bouk et al. (2017)	IoT based naming scheme for managing traffic in smart city	Yes	No	Yes	NDN	These names are easy to search also, data encrypted using Asymmetric Key Encryption.

Arshad et al. (2017)	IoT based naming schemes for smart campus	Yes	Yes	Yes	NDN	It uses SHA256 hash function for hashing flat component. Names generated are very long and not scalable.
Nour et al. (2020)	Multilayer Multicomponent Hierarchical Attribute-value (M2HAV) naming for wireless devices	Yes	No	Yes	CCN	It provides benefit of short names by location encoding, but additional efforts required for encoding and decoding those short names
Rehman et al. (2019)	IoT based naming for Name-Integrated Query (NINQ)	Yes	Yes	No	NDN	Names generated are too long making it difficult for searching and lookup process.
Javaheri et al. (2020)	IoT based naming for smart city	Yes	Yes	Yes	PURSUIT	It uses all three naming schemes to design names.

Table 2.6: Summary of related work

Chapter 3

Problem Statement

In today's world, every device or thing needs to be smart and accessible from anywhere, anytime. For example, in a daily scenario, a person wants to turn on the heater in the bedroom of the apartment from the office, So that before arriving in the room, it is warm already. This is possible because of the Internet of Things(IoT). It is also referred to as ubiquitous computing, which allows devices to be connected to the Internet in turn that devices will be accessed from anywhere.

3.1 Problem Description

IoT consists of various sensors, actuators, and dashboards. These sensors send the data such as room temperature or the status of the light. They could be referred to as publishers as they publish the data for further processing. The dashboard could be referred to as subscribes, which subscribes to this data to process it further and take some action. The actuators are on which the action is performed, such as turning the light on and setting the temperature for AC. These publishers and subscribers use Publish-Subscribe Internet Routing Paradigm (PSIRP) to communicate with each other. PSIRP is one of the architectures of the Information-Centric Network (ICN). MQTT protocol uses the same paradigm as a core to transport messages from one client to another.

While routing, the data in these network names plays a crucial role, as data or content is being mapped with names. MQTT uses topic-based naming, which means that the topic names are mapped to content or data that gets published. This led to the study of various naming conventions that could be adopted in the MQTT protocol for distributed brokers' IoT environment. It involves addressing those naming conventions' advantages and disadvantages. Along with that, to simulate and evaluate the real-time distributed environment by considering different topologies such as hierarchical or cluster topology.

Chapter 4

Design

This chapter provides complete details about the use-case scenario considered while addressing the research question specified in the problem statement. Also provides various designs to address every aspect of the problem statement.

4.1 Overview

As the main goal of this research is to study different naming conventions in a distributed brokers environment for IoT devices and with various topologies, one real-time use case taken into consideration is the smart building automation system. The smart building automation system is a system that provides a dashboard to a building manager. This dashboard provides a manager top-level view of all the sensors data present in each house in the building. In this building, every house is equipped with smart actuators such as a light - to turn it on or off remotely, an air conditioner - to turn AC on or off remotely or set the temperature remotely, and sensors such as a humidity sensor - to check humidity levels in a particular room, and temperature sensor - to check the temperature of a particular room. These sensors and actuators can be present in different rooms such as the kitchen, hall, and bedroom.

These sensors and actuators will provide real-time data, which could be accessed by the building manager from the dashboard given. This design uses Publisher-subscribe architecture using that it publishes the data to the MQTT broker.

4.2 Design of Naming Conventions

Everything in ICN revolves around naming. Naming plays an important role in Publisher-subscribe architecture. Publisher and subscriber need to agree upon the structure of

the naming convention before actual implementation to make it convenient. Because subscriber needs knowledge about the topics on which data will be published in the future so that subscriber can request subscriptions accordingly. Two naming conventions were designed as a part of this research and have been discussed in detail below.

4.2.1 Hierarchical Naming Convention

As the name implies, the hierarchical naming convention follows the structural format. The naming indicates the hierarchy going from the top level to the bottom in that each level has been separated from another by a slash(/). For instance, the hierarchical naming structure for the topic will be :

/domainName/location/task/sensorName

In the case of a smart building automation system, the building name is referred to as a domain level; after that, it uses a floor number. Then goes one level deeper to identify apartments using their number. Accordingly, adds the name of the room to which it has been referred. After this task has been used, the task could be either sensing or action. Sensing refers to sensing data and publishing it. Whereas action refers to the action that needs to be taken. It takes an extra value that should be used while performing an action further. Such as turning on the AC or turning it off. Sample topic for smart building automation system could be :

/redMills/floor2/apt23/bedroom/sensing/temperature

As it is seen, the above topic has been used to sense the temperature of apartment 23, the bedroom which is located on the second floor of the red mills building. For instance, to turn on the light in the hall of same apartment 23 in the red mills building following topic will be used:

/redMills/floor2/apt23/hall/action/light

And data will be: *On* In this naming convention, the assumption is made that network is secured to transfer any sensitive data or information transmitted within the network that is not sensitive enough. That is why during publication, data is sent as plain text with the topic name.

4.2.2 Hybrid Naming Convention

The hybrid naming convention approach designed as a part of this research combines hierarchical and flat-based naming conventions. The sample hybrid naming structure for the topic will be :

/domainName/location/task/sensorName:sensorID

This naming convention is similar to the hierarchical naming convention, but it has the additional element, which is sensorID to uniquely identify the sensor present in the specified location in the topic.

In the case of a smart building automation system sample topic could be:

/redMills/floor2/apt23/bedroom/sensing/light:AL2K78

Above given topic, it is used to sense whether light with sensor ID AL2K78 present in the bedroom of apartment 23 located on the second floor of building red mills is turned on or off.

This sensorID needs to be unique globally. That will be helpful if there are two or more sensors present in the room to uniquely identify them with the help of ID and not create any kind of confusion. For instance, if there are two lights present in the hall, it is possible to turn on or turn off the specific light from two lights with the help of specifying the appropriate sensor ID.

Additionally, in this naming convention, the assumption has been made that the network is not secured. Also, it could be that data is much sensitive to send in such an environment. For example, this can be true if sensors are sending data related to someone's health, such as blood pressure, oxygen levels, and many more. If this sensitive data gets leaked, it will be riskier, and it could disclose important information related to a person. By considering those scenarios, in the case of hybrid naming convention, data has been encrypted before sending it to the broker.

Two ways of data encryption are possible in MQTT, namely End-to-End (E2E) encryption and client-to-broker encryption. In the case of End-to-End (E2E), encryption it is considered that both publisher and subscriber clients are present in an unsecured environment. Whereas in the case of client-to-broker encryption, it is considered that the publisher client is in an unsecured environment, but subscribers are well known and trusted. In this design, End-to-End encryption is used, considering there is no trusted environment.

As shown in Figure. 4.1 Data will be encrypted before publishing to the MQTT broker. MQTT broker forwards the encrypted data around the network and sends it to the subscriber as per subscription requests made by subscribers. It is the responsibility of the subscriber to decrypt the data to use it further.

There are two data encryption methods, namely Asymmetric Key Encryption and Symmetric Key Encryption. This use-case discussed above involves IoT devices. These devices are lightweight devices with very less computing power, so symmetric key encryption has been the best option for them. As that will consume less power. In Symmetric

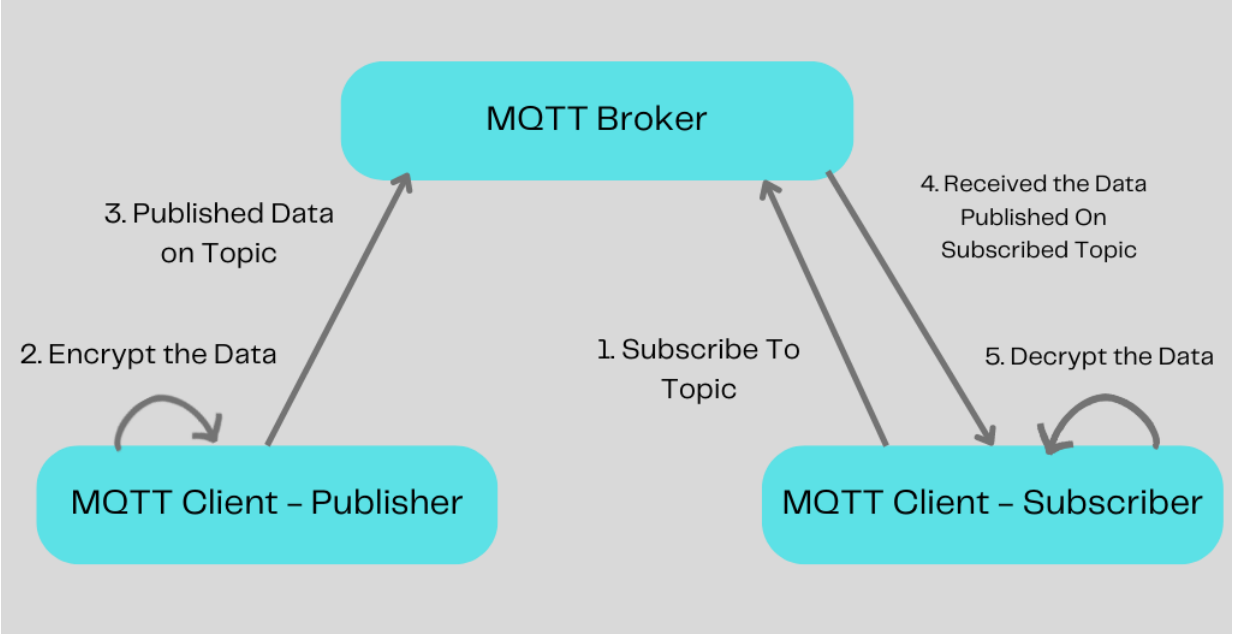

Figure 4.1: Encrypted data communication between MQTT client and broker

Key Encryption single key is used to encrypt and decrypt the data. It is the best practice to have a separate key for each topic present or created newly.

4.3 Design of Topology

This research not only focuses on different naming conventions but also looks for the evaluation of these naming conventions in different network topologies as specified in the problem statement under the Section. 3 To check, the same two types of network topology are considered hierarchical or tree topology and cluster topology for the distributed MQTT broker.

4.3.1 Scenario 1 : Hierarchical or Tree Topology With Static Sensors

In this topology, MQTT brokers are arranged in tree form. Where all leaf brokers send data to their parent broker. And data subscription is done to the MQTT broker present at the top level of the tree.

As shown in Figure. 4.2 the scenario was designed for the smart building automation system. In which the building manager is provided with a dashboard to manage all the resources. In this building, every house is equipped with smart actuators such as a light sensor - to turn it on or off remotely, an air conditioner sensor - to turn AC on or off

remotely, and sensors such as a humidity sensor - to check humidity levels in a particular room, and temperature sensor - to check the temperature in a particular room. These sensors and actuators can be present in different rooms such as the kitchen, hall, and bedroom.

As the name suggests, this topology provides a broker at each level to create a hierarchy structure. MQTT broker present at each level is responsible for MQTT client's connections at that level and levels below that. As shown in Figure. 4.2 It is a tree topology with depth 2. MQTT broker is present on each floor of the building, and one MQTT broker is present on top of all brokers present on the floor.

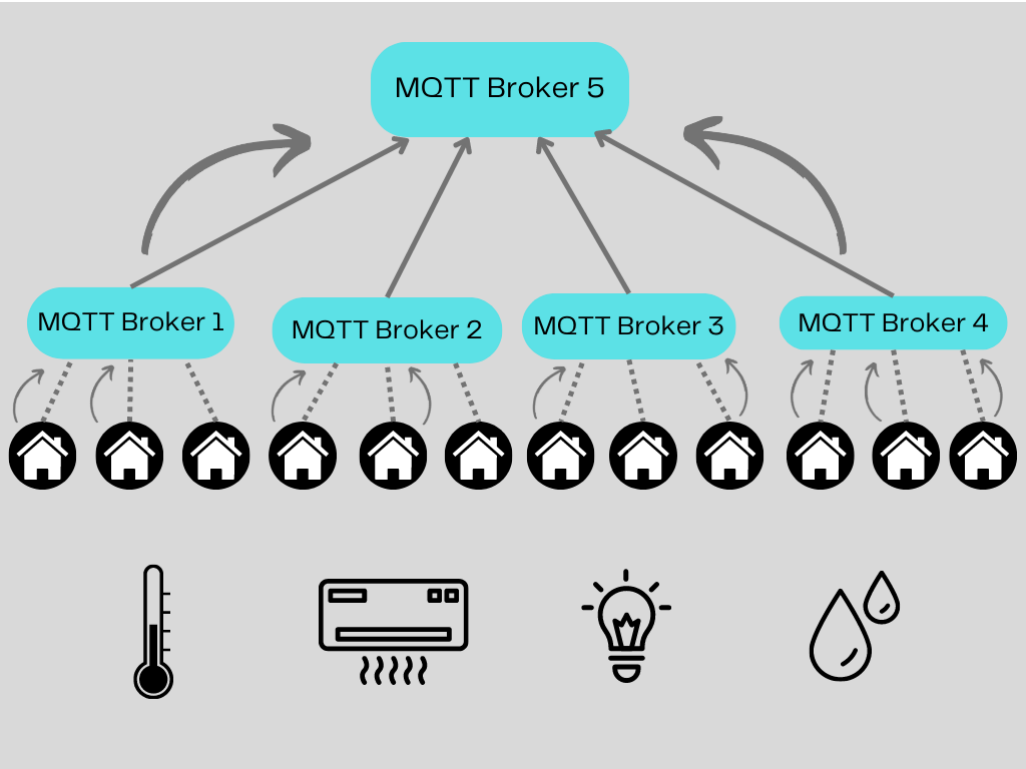

Figure 4.2: Hierarchical topology for Smart building automation system

Apartment on a specific floor will publish their data to the MQTT broker present on that floor with a topic name with a hierarchical or hybrid naming convention:

/apt23/bedroom/sensing/temperature

Whereas these MQTT brokers further send the data to the top-level brokers by prefixing topic depending upon the MQTT broker on which floor it is located:

/redMills/floor2/

The Building manager needs to subscribe to the topic :

/redMills/floor2/apt23/bedroom/sensing/temperature

In order to access the data.

As you can notice, this scenario involves static sensors means these sensors will not be moving in the network and will be situated in a permanent position.

4.3.2 Scenario 2 : Cluster Topology With Static Sensors

For cluster topology, All MQTT brokers are presented in a ring where each broker communicates with their neighboring broker. It is different from hierarchical topology as while publishing data topic name did not split up like hierarchical topology.

As shown in Figure. 4.3 A cluster of five MQTT brokers for a smart building automation system is provided. Unlike hierarchical topology, these brokers are not at a different level, but all MQTT brokers are at the same level. In this case, the apartment can send data to any of the MQTT Brokers present in the cluster, and subscribers can get the published data by subscribing to any of the MQTT Brokers in the cluster.

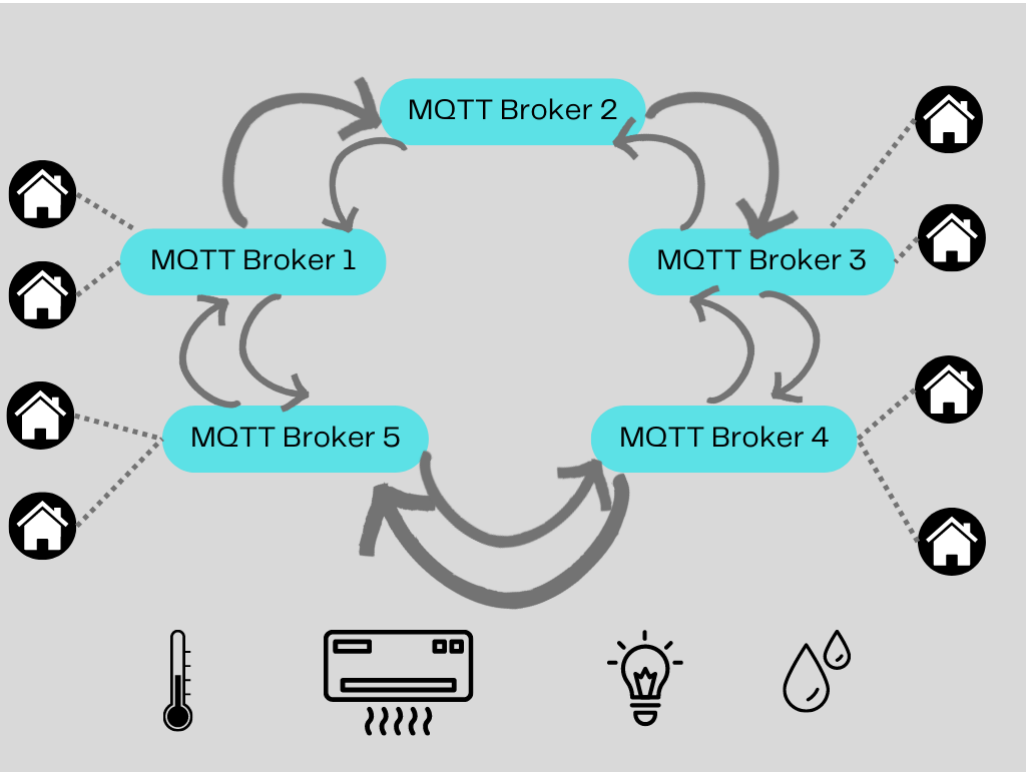

Figure 4.3: Cluster topology for Smart building automation system

4.3.3 Scenario 3 : Cluster Topology With Mobile Sensors

IoT network consists of sensors. These sensors could be static or mobile. Both of the scenarios designed and discussed above use static sensors. This use case is designed specifically for mobile sensors. In this use case, sensors give the status of the number of passengers presents currently in the buses moving around the city. Here the dashboard provides to city manager with the number of passengers and bus numbers and the current location of the moving bus. To make it easier for the city managers to utilize resources efficiently by routing more buses on busy routes. As shown in Figure. 4.4 Five MQTT brokers are connected to each other through clusters. The bus roaming around the city has the sensors, and it sends a timely update about the count to the MQTT Broker. In this case, as the bus is moving so, it will need to publish its data to the MQTT broker, which is close to it locationally, rather than sending the data to the fixed MQTT broker. This involves the challenge of searching for a broker located closely to the bus. As this is a cluster topology, the city manager can register their interest to any MQTT broker to receive an update about the data he/she is interested in.

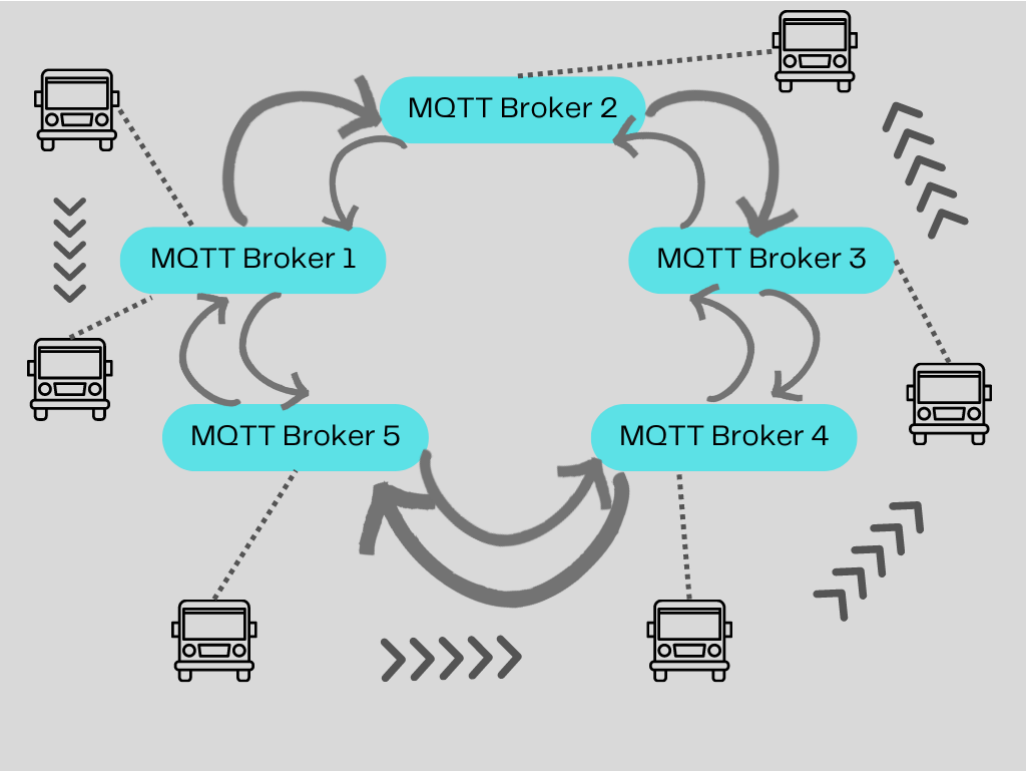

Figure 4.4: Cluster topology for Smart traffic management system

Chapter 5

Implementation

This Section provides implementation tools used to implement the design discussed in the Section. 4.3. It also gives a brief about the issues faced during the implementation. It includes the technical design of the actual implementation and their screenshots.

5.1 Challenges Faced During Implementation

As discussed in the Section. 3.1, to implement the designs described in Section. 4.3 distributed MQTT brokers' environment needed to be implemented. To conduct the various experiments using an actual environment would have cost the most in terms of money as well as resources. That is the reason why the network simulating tools option took into consideration. There are various network simulation tools available. Each simulation tool offers different features and has different capabilities and advantages. Choosing a single simulator from many of these was a challenge. A selected tool is needed to be capable of testing and experiencing the scenario specified in the design Section. Few of the network simulator tools have been studied and tried to run a sample network simulation on them to check whether they satisfy the requirements or not. The list of those network simulator tools and the reason why those were not chosen is given below:

1. Cooja Simulator :
   Cooja simulator Cooja (2022) is based on Contiki OS.OS (2022) This simulator provides a graphical user interface through which it is very easy to create mote of different types. It provides support for adding nodes for sensors like light. This simulator also supports the mobility of nodes at a basic level. But, it does not provide support for the MQTT protocol. Instead it supported the MQTT SN protocol.Ignacio (2022) MQTT SN is different from MQTT protocol at various levels.Stanford-Clark and Truong (2013) It uses the UDP protocol for data transmission. It uses topic id

over the topic names. As this research focused on topic names used in the MQTT protocol, this simulator is skipped.

2. Graphical Network Simulator-3 (GNS3) :
   GNS3 is open source and free network simulator software. It is a network simulator used to emulate, test, configure and troubleshoot virtual and real networks. This provides support to emulate not only the Cisco networking devices but also Cisco virtual switches, Cisco ASAs, Brocade vRouters, and Docker instances. This simulator is commonly used for wired devices and is not well suited for the emulation of wireless sensor networks. Therefore this simulator is skipped for this implementation.

3. Objective Modular Network Testbed in C++ (Omnet++) :
   It is a network simulator that is an extensible, component-based simulation library and framework written in C++. This simulator provides the support for emulation of the wireless sensor networks. But Omnet++ simulator does not provide native support for MQTT. Therefore this simulator also has been skipped.

4. Optimized Network Engineering Tools (Opnet) :
   This simulator is used to simulate any kind of network and measure the performance of that network. It provides support for the emulation of a wireless sensor network, as provided in the documentation. Opnet simulator has only a commercial version. There is no free and open source version available. Therefore, could not able to test and verify the support for the MQTT protocol. That's the reason why this simulator has also been skipped.

After noticing the limitation and shortcomings of the network simulation tools listed above, to implement the distributed MQTT brokers, environment docker containers are chosen. Network simulator such as Omnet++ and cooja does support the mobility of sensors, whereas it is not possible to have mobile sensors in the docker container environment. That's the reason why scenario 4.3.3 implementation is not possible to do with the docker container. So, the first two scenarios with hierarchical and hybrid naming conventions have both been implemented with hierarchical and cluster network topology.

5.2 Tools Used

This Section provides details about which tools are used in order to implement the scenarios designed and their purpose of usage.

5.2.1 Docker

As discussed in the Section. 5.1 implementation challenges are faced. There was a need to create a distributed MQTT broker environment that could be emulated and tested for the research of different naming conventions. After skipping all of the network simulators discussed. Docker has been selected to implement the containers of MQTT brokers.

Docker provides an implementation more similar to the real-world environment. Docker supports containerization because docker containers can be created from docker images presented or pulled from websites. These containers run on the docker engine. These containers provide some advantages as listed below :

1. **Lightweight** These containers share the OS of the kernel and are not required to set up an OS for each application separately. Because of this, it has been highly efficient. Docker containers are process-oriented and do not require a hardware hypervisor like a Virtual machine (VM). Also, it requires very few resources as compared to VM.

2. **Very fast** VM takes minutes to start or reboot. Whereas the docker container takes a few seconds to load and restart. Also, it is very easy to create, deploy, or modify the services on docker.

3. **Secure** Docker provides a safer environment for the application running and keep them isolated from one another as per the industry standards.

In this implementation, these docker containers are used to create multiple instances of the MQTT broker, which further interact with each other based on the topology implementation.

5.2.2 Docker Swarm

A docker swarm is a group of either physical or virtual machines that run the docker application. But these applications are configured in a way to join together in a cluster. Docker swarm is a container orchestration tool that allows the user to manage multiple containers deployed on a single virtual host. A docker swarm contains three types of items, namely nodes, services or tasks, and load balancers. Nodes are individual Docker engine instances that control your cluster and manage the containers that run your services and tasks. Load balancing is also part of Docker Swarm clusters to route requests across nodes.Powell (2021)

Docker swarm has two types of nodes, namely manager node and worker node. The primary function of the manager node is to assign the task to the worker nodes in the

swarm. In a docker swarm, multiple worker nodes can be created, whereas a maximum of seven manager nodes can be created. Each worker node receives the task to be executed by the manager node. Also, by default manager node is a worker node. Apart from managing tasks wherever resources are available, it also performs the task. For the implementation of a cluster of MQTT brokers, a docker swarm is used.

Docker swarm leverage a few benefits as listed below

1. **Cluster management integrated with docker-engine** There is no need for any additional orchestration software to create a docker swarm; it is easily created with Docker Engine Command Line Interface (CLI).

2. **Scaling** It is very easy to add new worker nodes or delete worker nodes with the help of a command. As swarm manager automatically adapts and accommodates the docker swarm as per the state requested using the command.

3. **Highly availability** In a docker swarm, there is a need for a manager node to assign tasks to the worker node. Having multiple manager nodes present in the docker swarm makes the cluster available all the time, even if one of the manager nodes fails for some reason, as a docker swarm allows the creation of manager nodes up to seven.

4. **Load balancing** Docker swarm provides the capability to schedule the tasks to ensure the efficient usage of available resources. The swarm manager node ensures that the workload has been assigned in a way to run on the most appropriate host to manage and optimize resources efficiently.

5.2.3 Eclipse Mosquitto

Eclipse Mosquitto is an open-source message broker that supports the various version of the MQTT protocol. Mosquitto is lightweight and suitable for low-power devices. The Docker image used for this implementation is

docker pull eclipse-mosquitto [1]

The latest tag of this image has been used. This docker image of Mosquitto broker supports MQTT of version 5.0. Mosquitto broker supports the vertical scaling of MQTT brokers, that is, the chaining of the MQTT broker. By adding a connection in the configuration file, it is possible to forward all the messages received on the specified topic or all topics on one broker to another broker-specific topic.

[1] `https://hub.docker.com/_/eclipse-mosquitto`

As shown in Figure.5.1 The message has been published on broker1 on topic /temp, and the message could be retrieved by subscribing to topic /level1/temp at broker2. As the configuration for only topic /temp has been added in broker1, forward it to broker2 by appending /level1 to it. At broker 1, Only those messages received on /temp will be forwarded, and no messages received on any other topic will be forwarded like /humidity.

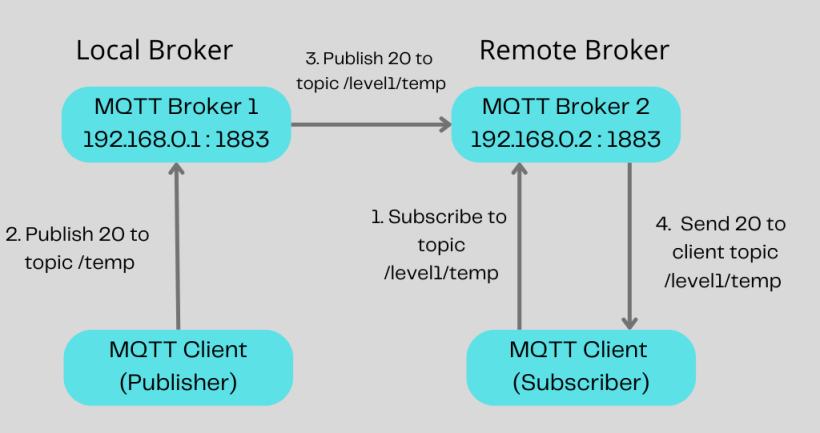

Figure 5.1: Mosquitto MQTT brokers' bridge

5.2.4 Hive MQ

Hive MQ is an MQTT broker. This allows fast, efficient, and reliable movement of data to and from connected IoT devices with the help of a client-based messaging platform. The Docker image used for this implementation is

docker pull hivemq/hivemq4:dns-latest [2]

The DNS-latest tag of this image has been used. This docker image of hive MQ supports MQTT of version 5.0. This tag supports the DNS cluster service discovery, which allows the scaling dynamically. Hive MQ instances are added or removed runtime as soon as they become available via DNS. Docker orchestration environment supports service discovery using a round-robin fashion. Hive MQ supports the horizontal scaling of the MQTT brokers. It allows the creation of the cluster of MQTT brokers with the help of the docker swarm. There is no need to add a specific configuration in a config file like a Mosquitto bridge configuration.

As shown in Figure. 5.2, The cluster of three docker nodes is created. Each node contains a swarm load balancer which allows the load balancing between the available

[2] `https://hub.docker.com/r/hivemq/hivemq4`

resources. 1883 port of each node is published, which is the default port for the MQTT protocol.

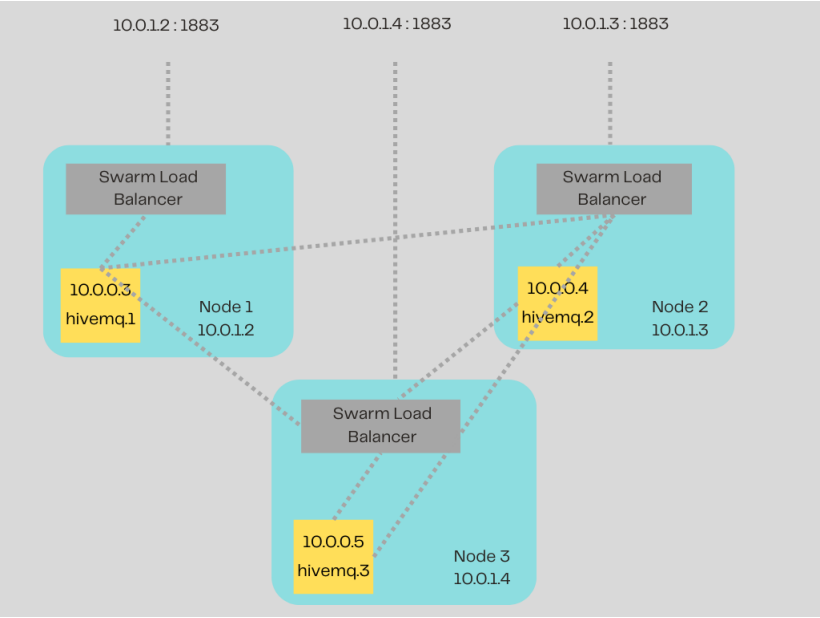

Figure 5.2: HiveMQ MQTT broker cluster configuration with docker swarm

5.2.5 Wireshark

Wireshark is a free and open-source packet sniffer and analysis tool. This tool is used to capture network traffic on various network devices connected to the local machine. It involves network traffic from Ethernet, Bluetooth, Wireless (IEEE. 802.11), and more. This tool is used for evaluation purposes of packets transmitted between MQTT client and broker and another way round.

5.3 Technical and Implementation Details

This Section gives a brief about the actual technical implementation done for addressing the designs discussed. It also includes the screenshots captured of the web pages for publisher and subscriber MQTT clients.

5.3.1 Scenario 1 : Hierarchical or Tree Topology With Static Sensors

As discussed in Section. 4.3, Figure. 5.3 shows the implementation-specific diagram for hierarchical or tree configuration of distributed MQTT brokers with static sensors. The

tree shown in the figure has two levels.Broker1, broker2, broker3, and broker4 are present at the lower level, and broker5 is at a higher level broker for all of the brokers. Each broker uses port number 1883 to listen to the published message for the MQTT protocol. For broker1, port 127.0.0.1: 9002 is used for WebSocket connection. Similarly, for broker5, port 127.0.0.1:9001 is used for WebSocket connection. WebSocket connection allows communication between the docker container and the localhost running program.

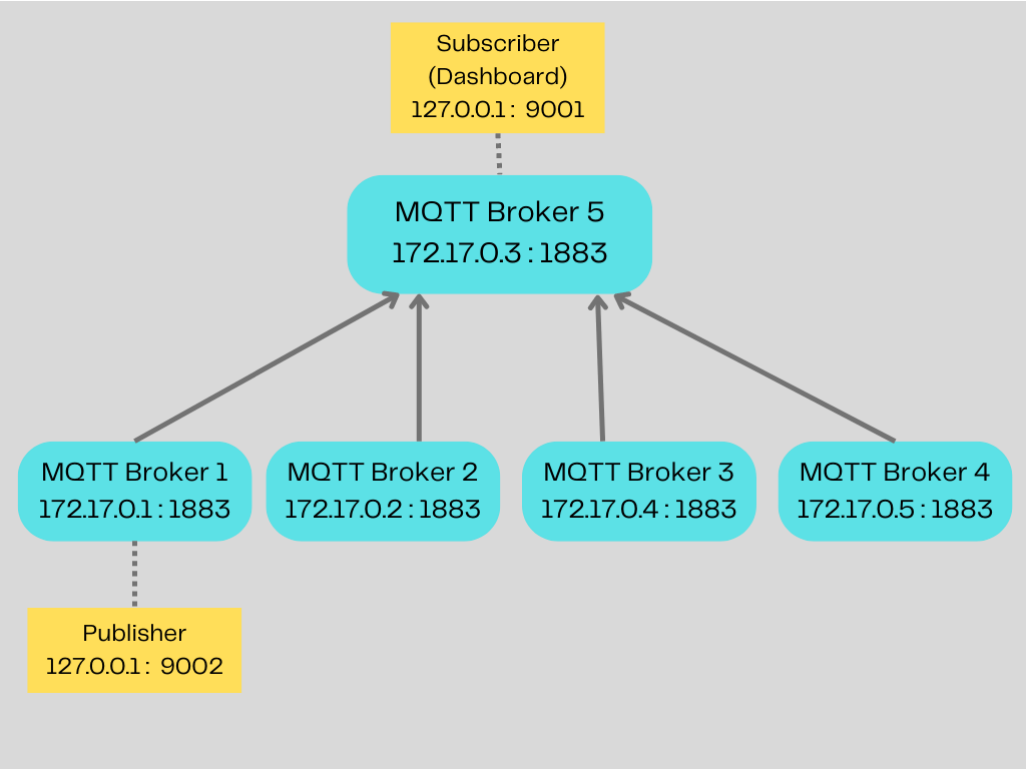

Figure 5.3: Hierarchical topology of mosquitto MQTT brokers with bridge configuration

To create a bridge from all brokers at a lower level to a higher level, configuration for the bridge needs to be added. Figure. 5.4 shows a sample of the configuration files provided for broker1.

As specified, the connection name "bridge" that uses forwarding address "172.17.0.3", which is the address of broker5 to port "1883". As mentioned in the next line, it is specified using the format

topic topic_pattern direction QOS local_prefix remote_prefix

It could be noticed from the sample configuration file topic pattern mentioned is # that is, all topics present on broker1. Direction specified as out means messages will be forwarded to the address specified of broker5 by prefixing topic "redmills/floor1/" to all topics received as its remote prefix.

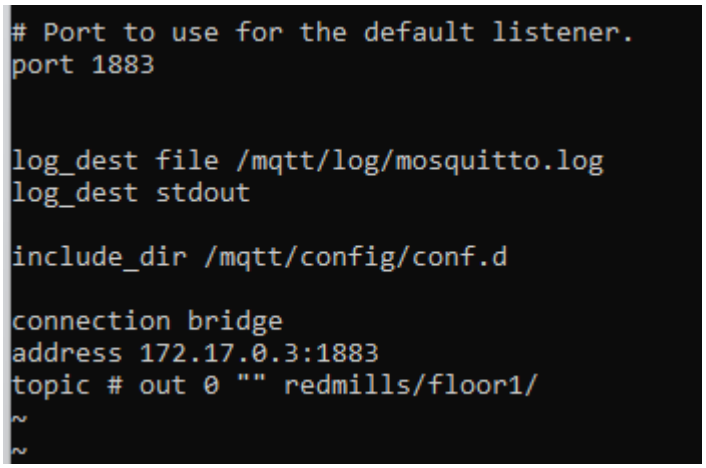

Figure 5.4: Configuration file for broker1 to create a bridge to broker5

Hierarchical Naming Convention

As shown in Figure. 5.5 shows the screenshot of the MQTT publisher page. That is written using HTML and javascript. It uses the paho-MQTT client to connect to the broker1 using WebSocket. It could be seen in the screenshot that the port is mentioned as 9002. To publish the data, 20 have been published on the topic "/apt23/bedroom/sensing/temperature".

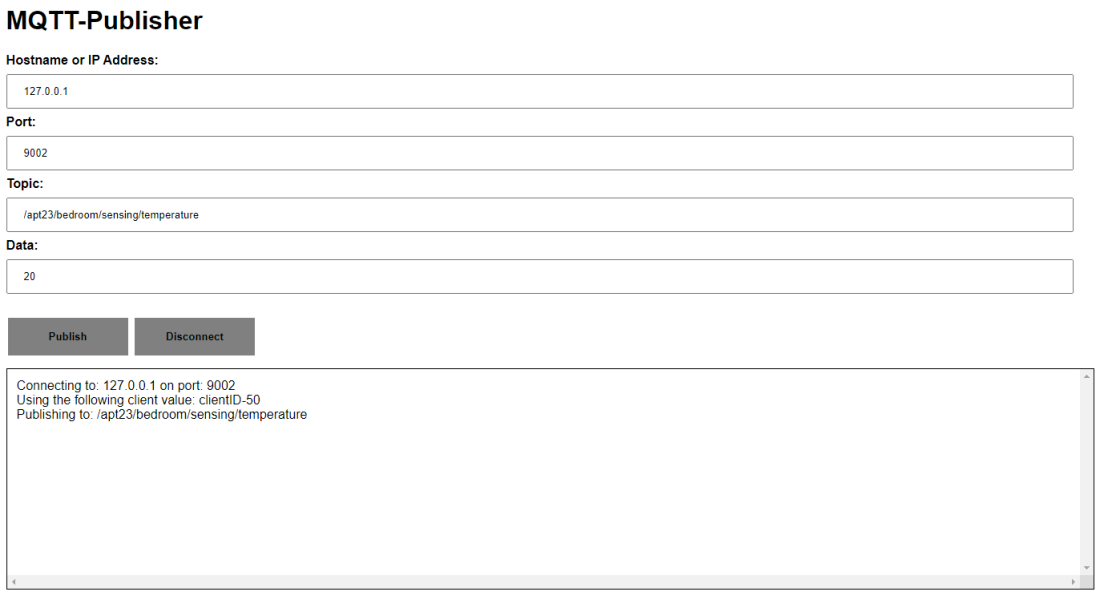

Figure 5.5: Webpage for MQTT client to publish data to broker1

As shown in Figure. 5.6 shows the screenshot of the MQTT subscriber page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker5 using WebSocket. It could be seen in the screenshot that the port is mentioned

as 9001. It subscribes to the topic '#' that is all topics at the broker5. As specified in the configuration file, "redmills/floor1" has been prefixed to the topic received. So, this data 20 will be available on the topic "redmills/floor1/apt23/bedroom/sensing/temperature", as shown in the text box.

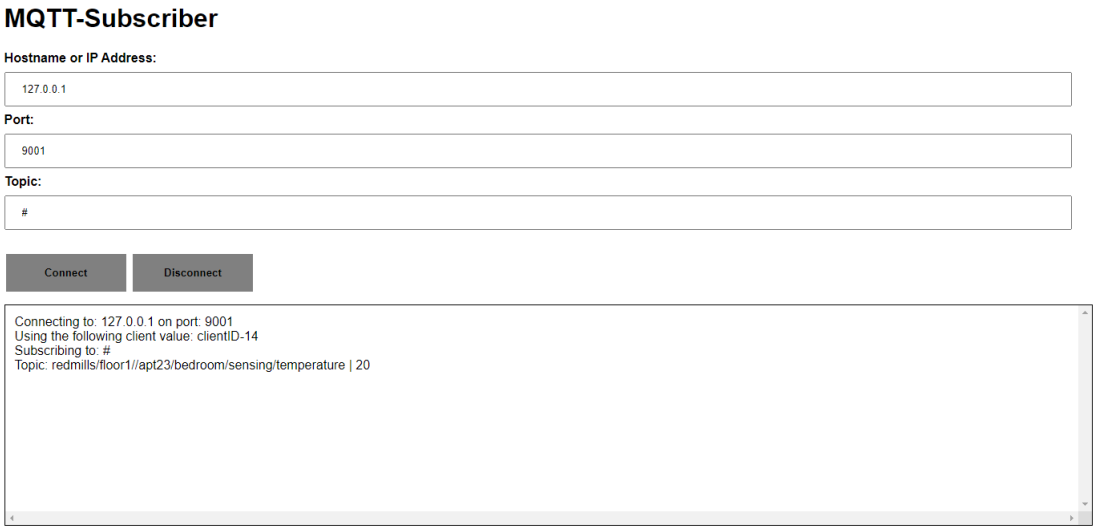

Figure 5.6: Webpage for MQTT client to subscribe to broker5

Hybrid Naming Convention

As shown in Figure. 5.7 shows the screenshot of the MQTT publisher page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker1 using WebSocket.It could be seen in the screenshot that the port is mentioned as 9002. It is mentioned in the design that data has been encrypted before publishing it to the topic. To encrypt data CryptoJS.AES.encrypt() method is used along with passkey. As provided, encrypted data has been printed in the text box. To publish the encrypted data has been published on the topic "/apt23/bedroom/sensing/temperature:AL2K36".

As shown in Figure. 5.8 shows the screenshot of the MQTT subscriber page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker5 using WebSocket. It could be seen in the screenshot that the port is mentioned as 9001. It subscribes to the topic '#' that is all topics at the broker5. As specified in the configuration file, "redmills/floor1" has been prefixed to the topic received. Broker5 receives the encrypted data, which is then decrypted using CryptoJS.AES.decrypt() method along with passkey used during encryption of data. So, this decrypted data will be available on the topic "redmills/floor1/apt23/bedroom/sensing/temperature:AL2K36" as shown in the text box.

Figure 5.7: Webpage for MQTT client to publish encrypted data to broker1

Figure 5.8: Webpage for MQTT client to subscribe to and decrypt data from broker5

5.3.2 Scenario 2 : Cluster Topology With Static Sensors

As discussed in Section. 4.3, Figure. 5.9 shows the implementation-specific diagram for cluster configuration of distributed MQTT brokers with static sensors. It contains a cluster of five nodes. These clusters were created using a docker swarm. Each node runs an instance of service hivemq that also contains a swarm load balancer. These load balancer helps to manage the available resources efficiently. Each load balancer is connected with all services available and running on various nodes. In this cluster, only

one node from the cluster has been exposed with port number 8000 for WebSocket.

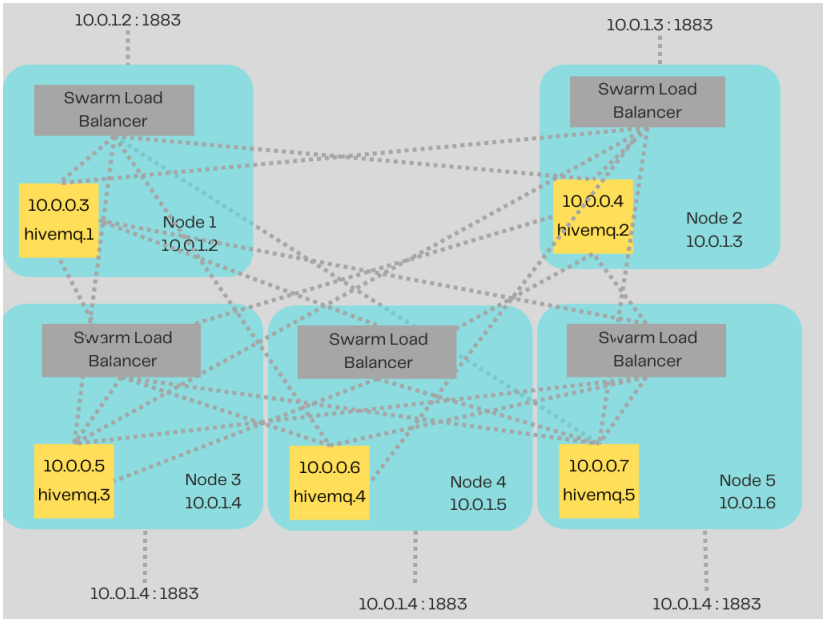

Figure 5.9: Cluster topology of HivqMQ MQTT brokers with docker swarm

Hierarchical Naming Convention

As shown in Figure. 5.10 shows the screenshot of the MQTT publisher page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker1 using WebSocket.It could be seen in the screenshot that the port is mentioned as 8000. To publish the data, 15 have been published on the topic "redmills/floor1/apt23/bedroom/sensing/temperature"

As shown in Figure. 5.11 shows the screenshot of the MQTT subscriber page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker1 using WebSocket. It could be seen in the screenshot that the port is mentioned as 8000. It subscribes to the topic '#', which is all topics present in the cluster received. So, this data 15 will be available on the topic "redmills/floor1/apt23/bedroom/sensing/temperature", as shown in the text box.

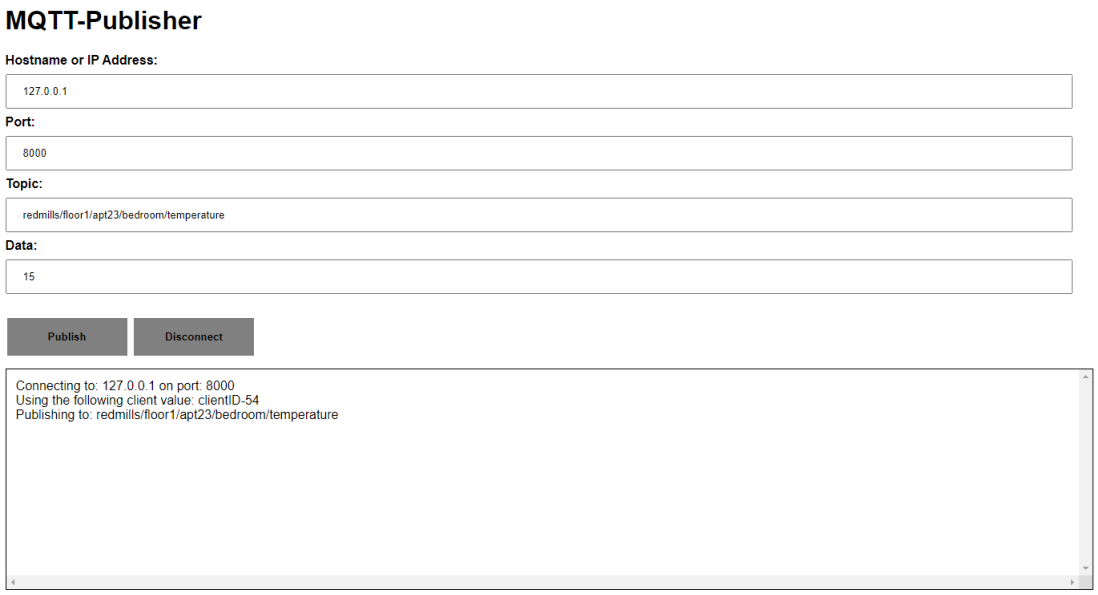

Figure 5.10: Webpage for MQTT client to publish data to broker present in cluster

Figure 5.11: Webpage for MQTT client to subscribe to broker present in the cluster

Hybrid Naming Convention

As shown in Figure. 5.12 shows the screenshot of the MQTT publisher page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker1 using WebSocket.It could be seen in the screenshot that the port is mentioned as 8000. It is mentioned in the design data has been encrypted before publishing it to the topic. To encrypt data CryptoJS.AES.encrypt() method is used along with passkey. As provided, encrypted data has been printed in the text box. To publish the encrypted data has been

published on the topic “redmills/floor1/apt23/bedroom/sensing/temperature:Al2k36”

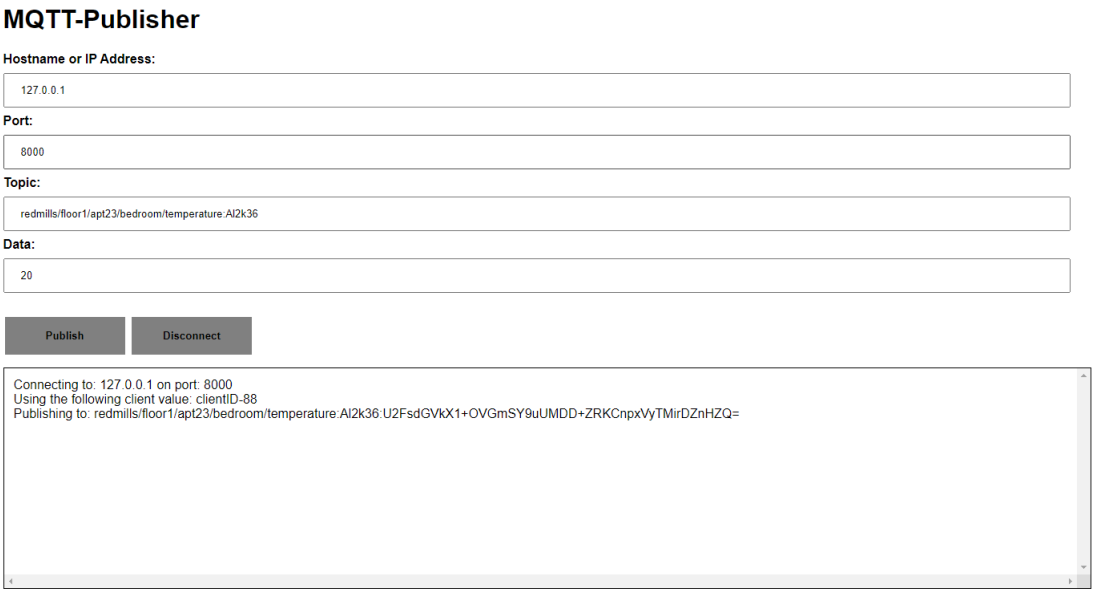

Figure 5.12: Webpage for MQTT client to publish encrypted data to broker present in the the cluster

As shown in Figure. 5.13 shows the screenshot of the MQTT subscriber page. That is written using HTML and JavaScript. It uses the paho-MQTT client to connect to the broker1 using WebSocket. It could be seen in the screenshot that the port is mentioned as 8000. It subscribes to the topic ‘#’, which is all topics present in the cluster received. Broker1 receives the encrypted data, which is then decrypted using CryptoJS.AES.decrypt() method along with passkey used during encryption of data. So, this decrypted data will be available on the topic “redmills/floor1/apt23/bedroom/sensing/temperature:Al2k36”, as shown in the text box.

Figure 5.13: Webpage for MQTT client to subscribe to and decrypt data received on broker present in the cluster

Chapter 6

Experiments and Discussion

This Section contains the experiment conducted on the network topology designed for distributed MQTT brokers' environments. Also, a detailed discussion about the results of those experiments. This discussion consists of a comparison made between the naming conventions and network topologies.

6.1 Experiments

This Section provides an overview of experiments conducted on a distributed MQTT broker network topology.

6.1.1 Experiment 1: Addition Of A New Broker To The Existing Network Topology

In this experiment, a new broker has been added to the existing network topology. As shown in Figure. 6.1 This shows hierarchical network topology; for this topology, the MQTT broker has been added at a different level to check the additional overhead required to set up the new broker. In the case of a cluster topology, a new MQTT broker has been added to a cluster as there is no hierarchy present.

6.1.2 Experiment 2: Failure Of A Broker In A Network Topology

In this experiment, one of the MQTT broker from the network has been failed manually to check the consequences on network topology. This experiment is designed to evaluate the aftereffects of the failure of the MQTT broker, whether the subscriber could get the data seemingly or not, would whether the part of the network will be disconnected or not.

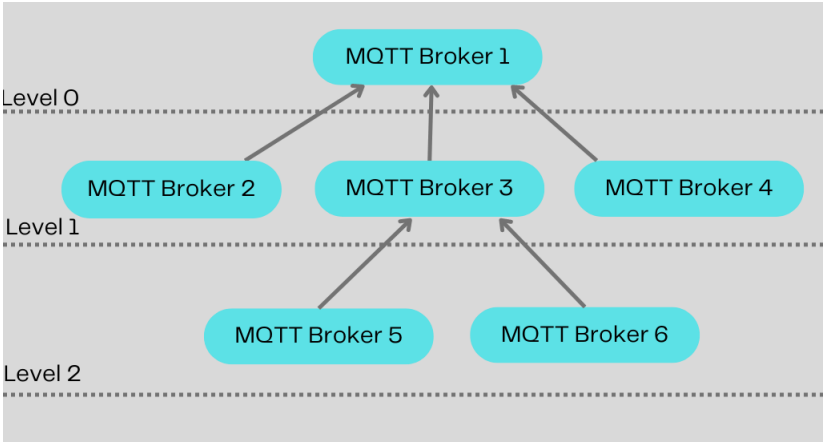

Figure 6.1: Hierarchy of MQTT brokers

In the case of hierarchical topology, brokers at different levels failed manually to observe the consequences. Whereas in the case of a cluster, as it forms a ring, only one broker from the available brokers has failed manually.

6.2 Discussion and Comparison

This Section provides the comparison made between designed naming approaches and MQTT brokers' topology. Also, the discussion on the results obtained from experiments conducted.

6.2.1 Comparison Between Naming Schemes

This Section provides a detailed comparison made between hierarchical naming schemes and designed hybrid naming schemes. This comparison is performed using a few of the comparison factors.

Aggregation Of Topic

In MQTT, topic names are used during subscription to the broker and also while publishing data to the respective broker. It is possible to filter the topic by using topic filters. This topic filter supports the aggregation of the topic. For example, a subscriber connected to an MQTT broker needs to subscribe to all topics on which data is published. Instead of subscribing to each topic separately. It will only subscribe to '#', which in turn provides a subscription for all topics available.

Two types of wildcards are used for topic filtering.

- #: Multilevel wildcard

  - The multilevel wildcard is used to match any number of levels within the topics.
  - Multilevel wildcard represents zero or more levels.
  - For example, */apt23/#* subscription will match to the topics */apt23/bedroom/sensing/humidity* , */apt23/kitchen/action/light*.

- ?: single level wildcard
  - Single level wildcard matches one and only one level.
  - For example, */apt23/?* Subscription will match to the topic */apt23/bedroom/*and not */apt23/bedroom/sensing/*

Hierarchical naming schemes allow a higher level of topic aggregation as these name schemes consist of structured string literals separated using slash (/). It supports the use of both the topic filter wildcard.

1. #: Multilevel wildcard
   For example, If available topics in hierarchical naming scheme are as follows
   redmills/floor2/apt23/bedroom/sensing/temeprature ,
   redmills/floor2/apt23/kitchen/sensing/temeprature,
   redmills/floor2/apt23/kitchen/action/ac,
   Then to get all the data present related to apartment 23 only, a subscription to topic */redmills/floor2/apt23/#* is sufficient.

2. ?: Single level wildcard
   For example, If available topics in hierarchical naming scheme are as follows
   redmills/floor2/apt23/bedroom/sensing/temeprature ,
   redmills/floor2/apt23/kitchen/sensing/temeprature,
   redmills/floor2/apt23/kitchen/action/ac,
   Then to get all the temperature-related data present for apartment 23 only, a subscription to topic */redmills/floor2/apt23/?/sensing/temperature* is sufficient.

A hybrid naming scheme implemented is a combination of hierarchical and flat-based naming schemes. This naming schemes consist of a string literal separated using slash(/) and appended by the device id with a semicolon(:) at the end. It does offer the advantage of topic aggregation as a hierarchical naming scheme. But as this naming scheme uses device id after the colon(:) which could not get covered in aggregation.

For example, If available topics in hybrid naming scheme are as follows
redmills/floor2/apt23/bedroom/sensing/temeprature:2Dbnj7 ,
redmills/floor2/apt23/kitchen/sensing/temeprature:Jul6gh,

redmills/floor2/apt23/kitchen/action/ac:Uoh765,
redmills/floor2/apt23/kitchen/sensing/light:L78hul

Then to get all the data from sensing devices present for all the rooms in apartment 23 only, a subscription to topic */redmills/floor2/apt23/?/sensing/#* is sufficient. But if further there is a need to get all the temperature-related data present for apartment 23 only, a subscription to topic */redmills/floor2/apt23/?/sensing/temperature* is not possible in this case as content after a semicolon(:) will not get filter by topic filter.

Security

In hierarchical naming schemes, it sends data as plain text by taking into consideration that data has been transmitted in a secured environment. That's the reason why if in case data packets are leaked; a third person would be able to read the data simply by looking at the leaked data packet. Whereas, in the case of hybrid naming convention, End-to-End(E2E) data encryption is used during data publication. The hybrid naming convention uses Symmetric key encryption to encrypt the data. This data encryption provides an extra layer of protection to the data. This is very helpful while transmitting sensitive data such as personal information, health data, and many more. Although this approach provides an advantage, there is an extra overhead cost required in terms of encrypting the data at the publisher end and decrypting the data at the subscriber end.

Flexibility To Add New Topic

The hierarchical naming convention offers the flexibility to add new topics without any extra overhead. Publishers can publish the data to the new topic created as per the structure given, and subscribers can subscribe to this new topic. In the case of a hybrid naming convention, It uses a unique device id in the topic. For that purpose, additional configuration is needed at the publisher side to add the device id in the configuration file so that the device id will be unique and constant for future instances.

The summary of the comparison between these naming schemes is tabularized under Table. 6.1

Naming scheme	Total number of component	Does it have a fixed number of component in naming	Secure	Aggregation	Scalable
Hierarchical naming	6	No	No	Yes	Yes
Hybrid naming	7	Yes	Yes	Less as compared to hierarchical naming	Less as compared to hierarchical naming

Table 6.1: Summary of comparison between naming schemes

6.2.2 Comparison Between Network Topologies

This Section provides a detailed discussion of the results and a description of the comparison done on the basis of the observations after conducting experiments. The following sections contain comparisons made between naming schemes as well as between network topology using the various factors.

Setup Cost To Add A New Broker

In the case of a hierarchical topology, to add a new MQTT broker as a docker container, an additional bridge configuration is required to forward the data packets in the MQTT configuration file. Hierarchical topology resembles the tree structure, As shown in Figure. 6.2, in which the addition of a broker at the top level will cause configuration file changes to all the intermediate child brokers. For example, adding a broker at level n will need updation in the configuration file of the broker at level n+1.

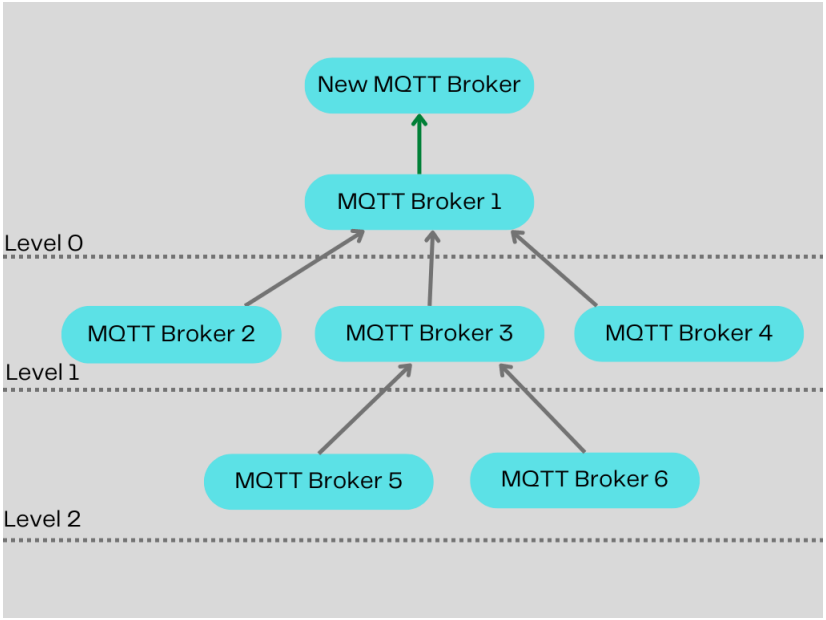

Figure 6.2: Addition of MQTT broker at hierarchy level 0

Whereas in the case of adding a broker at the bottom of the tree, there are no changes needed in an existing broker configuration files, but a new bridge configuration needs to be added for the newly added broker. This can be depicted in Figure. 6.3.

On the other hand, in the case of cluster topology, to add a new MQTT broker to the existing cluster, no additional manual configuration file changes are required. It is easier to scale up and scale down the cluster with the help of a single command.
Command to scale:

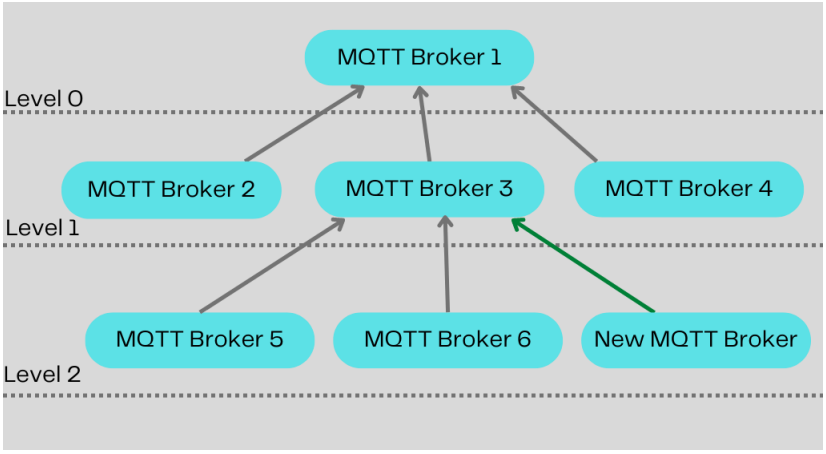

Figure 6.3: Addition of MQTT broker at hierarchy level 2

docker service scale hivemq=5 [1]

Consequences Of Failure Of Broker

In the case of a hierarchical topology as it resembles the tree structure. MQTT broker at different levels has been marked failure manually to check the consequences. As shown in Figure. 6.4. If the broker at the top level marked failed, there is no way for subscribers to get the data as that's the only broker visible to all subscribers. Subscriber will get disconnected from the broker network and will not be able to receive any data. The QoS level of the message being published does not affect the message being reached, as after restarting or recovering; the broker subscriber needs to re-establish the connection with the broker.

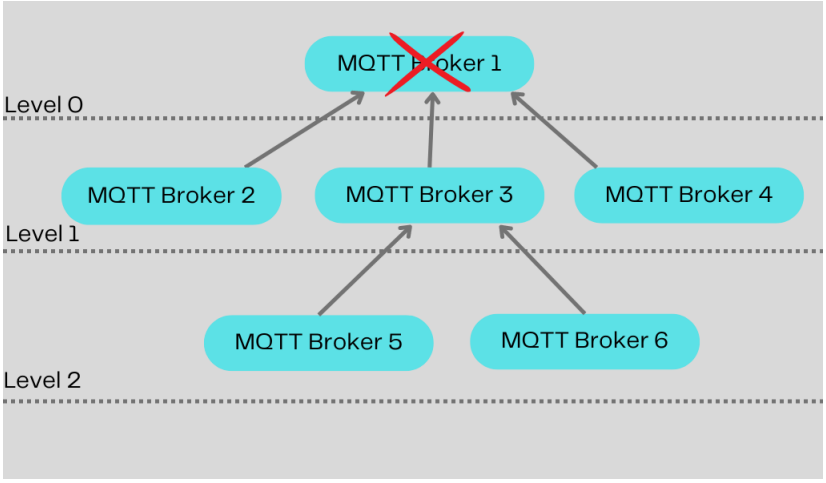

Figure 6.4: Deletion of MQTT broker at hierarchy level 0

[1] `https://www.hivemq.com/docs/hivemq/4.8/user-guide/docker.html`

In another scenario, as depicted in the Figure. 6.5, if a broker other than the top level fails, then only that part of the network will get disconnected from the top level broker. The subscriber did not get aware of this situation. The subscribe did receive the data from available publishing brokers, whereas failed brokers would continue sending the data after recovering.

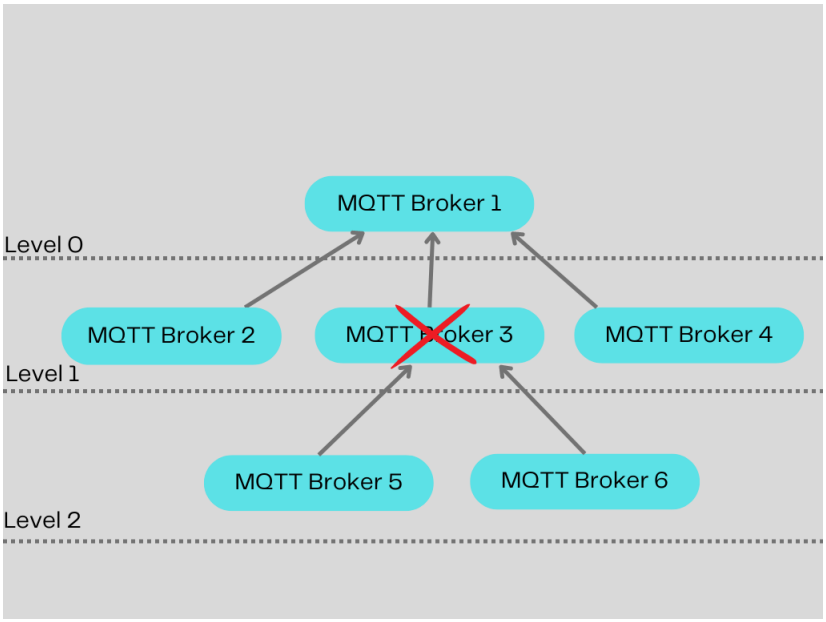

Figure 6.5: Deletion of MQTT broker at hierarchy level 1

In case of an MQTT broker failure in a cluster topology, this implementation uses a docker swarm which is a manager node that constantly checks the number of tasks running; if any of the tasks get failed, it creates a new task immediately. Also, all brokers are at the same level. That's why the failure of any broker does not break the network or disconnect any part of the network. Additionally, failure broker gets recovered automatically; no manual recovery of a broker is required.

Payload Length

The Table. 6.2 shows the payload length of all MQTT control packets captured for hierarchical and cluster network topology with both hierarchical and hybrid naming schemes. PUBLISH and SUBSCRIBE payload length varies in the case of a hierarchical naming scheme as the topic name gets prefixed while sending data from one broker to another. Whereas in the case of a cluster, payload length remains the same for PUBLISH and SUBSCRIBE control packets. Also, The payload length for hybrid naming is more than the hierarchical naming payload as in the case of hybrid naming, encrypted data has been sent than the plain text.

Network Topology	Naming Scheme	MQTT control packet	Payload Length (Byte)
Hierarchical Topology	Hierarchical naming	PUBLISH	42
		SUBSCRIBE	58
	Hybrid naming	PUBLISH	91
		SUBSCRIBE	107
Cluster Topology	Hierarchical naming	PUBLISH	49
		SUBSCRIBE	49
	Hybrid naming	PUBLISH	98
		SUBSCRIBE	98

Table 6.2: Payload length of MQTT control packets in network topology

Chapter 7

Conclusions & Future Work

This Section describes in detail the conclusion of the research being conducted. The conclusion is drawn on the basis of results observed after conducting the experiments as a part of the thesis. Also, the future scope of this thesis.

7.1 Conclusion

This work was motivated because of the increased usage of IoT devices. These devices use data-oriented naming rather than the host-centric approach. This led to having an understanding and design of the naming scheme, which provides benefits in terms of security, scalability, and flexibility.

A review of existing approaches and work leads to an understanding that most of the design and implementation provided earlier most commonly use CCN or NDN architecture. Only one work has been dedicated to PURSUIT architecture. Furthermore, as reviewed, none of the work presented before evaluated the performance of the naming scheme as per the underneath network topology used for the implementation of nodes.

As per the naming schemes designed and implemented as part of the thesis, it can be concluded that the hierarchical naming schemes approach is more scalable and aggregated than the designed hybrid naming scheme approach. On the other hand, the hybrid naming scheme approach is more secure.

In terms of the selection of naming approach, there is a need to address the overhead of security vs scalability. In terms of use cases in which more sensitive data has been involved, such as IoT system for health monitoring, it shares the sensitive data which needs to be protected, and leakage of these data would be risky. In that case, the hybrid naming scheme fits perfectly.

In terms of topology, cluster topology is a good option to select for any IoT use case.

It offers various benefits such as

- Load balancer allows the efficient utilization of resources by sharing the tasks among available nodes
- Brokers are available all the time, as the failure of any broker does not cause the part of the network to get disconnected or data unavailable to the client like hierarchical network topology.
- It is easier to manage various brokers in terms of adding the new one; no additional setup or config file changes are required. It can be scaled up or down with the help of single command.

7.2 Future Work

In this thesis, the hierarchical naming scheme and the hybrid naming scheme have been implemented with network topologies such as hierarchical topology and cluster topology. The scenarios considered in the thesis contain a network of static sensors. To extend the same research in the future, it is possible to evaluate the same naming schemes within a given network topology but with mobile sensors. There are various real-time use cases possible in which mobile sensors can be used. One of them has been explained under Section. 4.3.3.

To implement such a scenario, as mentioned in the research, there was a lack of network simulation tools to provide support for the implementation of the distributed environment using mobile sensors. So there is a need to search for tools to satisfy these requirements. Additionally, more research can be conducted to implement the new variation of hybrid naming schemes with the hierarchical and attribute-value-based naming scheme. Evaluating the performance of those naming schemes with the current implemented naming schemes as a baseline would be possible.

Bibliography

Ahlgren, B., Dannewitz, C., Imbrenda, C., Kutscher, D., and Ohlman, B. (2012). A survey of information-centric networking. *IEEE Communications Magazine*, 50(7):26–36.

Amadeo, M., Campolo, C., Iera, A., and Molinaro, A. (2015). Information centric networking in iot scenarios: The case of a smart home. In *2015 IEEE International Conference on Communications (ICC)*, pages 648–653.

Arshad, S., Azam, M. A., Ahmed, S. H., and Loo, J. (2017). Towards information-centric networking (ICN) naming for internet of things (IoT). In *Proceedings of the International Conference on Future Networks and Distributed Systems*. ACM.

Arshad, S., Shahzaad, B., Azam, M. A., Loo, J., Ahmed, S. H., and Aslam, S. (2018). Hierarchical and flat-based hybrid naming scheme in content-centric networks of things. *IEEE Internet of Things Journal*, 5(2):1070–1080.

Bouk, S. H., Ahmed, S. H., Kim, D., and Song, H. (2017). Named-data-networking-based its for smart cities. *IEEE Communications Magazine*, 55(1):105–111.

Cooja (2022). Cooja simulator. `https://anrg.usc.edu/contiki/index.php/Cooja_Simulator`. [Online; accessed 16-Aug-2022].

CORDIS (2022). Publish subscribe internet technology. `https://cordis.europa.eu/project/id/257217`. [Online; accessed 16-Aug-2022].

Din, I. U., Hassan, S., Khan, M. K., Guizani, M., Ghazali, O., and Habbal, A. (2018). Caching in information-centric networking: Strategies, challenges, and future research directions. *IEEE Communications Surveys Tutorials*, 20(2):1443–1474.

Ghodsi, A., Koponen, T., Rajahalme, J., Sarolahti, P., and Shenker, S. (2011). Naming in content-oriented architectures. In *Proceedings of the ACM SIGCOMM Workshop on Information-Centric Networking*, ICN '11, page 1–6, New York, NY, USA. Association for Computing Machinery.

Gupta, P. and Prabha., I. O. (2021). A survey of application layer protocols for internet of things. In *2021 International Conference on Communication information and Computing Technology (ICCICT)*. IEEE.

Ignacio, A. (2022). aignacio/mqtt-sn-contiki_example. `https://github.com/aignacio/mqtt-sn-contiki_example`. [Online; accessed 16-Aug-2022].

Javaheri, A., Hashemi, S. N. S., and Bohlooli, A. (2020). Hybrid naming scheme based pursuit architecture for smart city. In *2020 4th International Conference on Smart City, Internet of Things and Applications (SCIOT)*, pages 33–38.

Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. (2009). Lipsin: Line speed publish/subscribe inter-networking. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication*, SIGCOMM '09, page 195–206, New York, NY, USA. Association for Computing Machinery.

Jovanovic, B. (2022). Internet of things statistics for 2022 - taking things apart. `https://dataprot.net/statistics/iot-statistics/`. [Online; accessed 16-Aug-2022].

Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. (2007). A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192.

Kurdi, H. and Thayananthan, V. (2022). A multi-tier mqtt architecture with multiple brokers based on fog computing for securing industrial iot. *Applied Sciences*, 12(14):7173.

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H., and Zhao, W. (2017). A survey on internet of things: Architecture, enabling technologies, security and privacy, and applications. *IEEE Internet of Things Journal*, 4(5):1125–1142.

Mochida, T., Nozaki, D., Okamoto, K., Qi, X., Wen, Z., Sato, T., and Yu, K. (2017). Naming scheme using nlp machine learning method for network weather monitoring system based on icn. In *2017 20th International Symposium on Wireless Personal Multimedia Communications (WPMC)*, pages 428–434.

Nour, B., Sharif, K., Li, F., Moungla, H., and Liu, Y. (2020). A unified hybrid information-centric naming scheme for iot applications. *Comput. Commun.*, 150(C):103–114.

OS, C. (2022). Contiki os. `https://github.com/contiki-os/contiki`. [Online; accessed 16-Aug-2022].

Piro, G., Cianci, I., Grieco, L., Boggia, G., and Camarda, P. (2014). Information centric services in smart cities. *Journal of Systems and Software*, 88:169–188.

Powell, R. (2021). Docker swarm vs kubernetes: how to choose a container orchestration tool. `https://circleci.com/blog/docker-swarm-vs-kubernetes/`. [Online; accessed 16-Aug-2022].

PSIRP (2022). Publish subscribe internet routing paradigm. `http://www.psirp.org/`. [Online; accessed 16-Aug-2022].

Rehman, M. A. U., Ullah, R., and Kim, B. S. (2019). Ninq: Name-integrated query framework for named-data networking of things. *Sensors*, 19(13).

Shang, W., Ding, Q., Marianantoni, A., Burke, J., and Zhang, L. (2014). Securing building management systems using named data networking. *IEEE Network*, 28(3):50–56.

Stanford-Clark, A. and Truong, H. (2013). Mqtt for sensor networks (mqtt-sn) protocol specification version 1.2. `https://www.oasis-open.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf`. [Online; accessed 16-Aug-2022].

Tanenbaum, A. S. and Steen, M. v. (2006). *Distributed Systems: Principles and Paradigms (2nd Edition)*. Prentice-Hall, Inc., USA.

Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. V., and Polyzos, G. C. (2014). A survey of information-centric networking research. *IEEE Communications Surveys Tutorials*, 16(2):1024–1049.

Appendix

Abbreviation	Expansion
IoT	Internet of Things
ICN	Information Centric Networking
MQTT	Message Queuing Telemetry Transport
PSIRP	Publish-Subscribe Internet Routing Paradigm
TCP/IP	Transmission Control Protocol/Internet Protocol
DONA	Data-Oriented Network Architecture
NDN	Named Data Networking
CCN	Content-Centric Networking
PURSUIT	Publish-Subscribe Internet Technology
MQTT SN	Message Queuing Telemetry Transport for Sensor Network
E2E	End-to-End
UDP	User Datagram Protocol
GNS3	Graphical Network Simulator-3
Opnet	Optimized Network Engineering Tools
Omnet++	Objective Modular Network Testbed in C++ Omnet++
VM	Virtual Machine
NetInf	The Network of Information
CoAP	Constrained Application Protocol
AMQP	Advanced Message Queuing Protocol
STOMP	Simple/Streaming Text Oriented Messaging Protocol
SMCP	Simple Media Control Protocol
URL	Uniform Resource Locator