MQTT2EdgePeer: a Robust and Scalable Peer-to-Peer Edge Communication Infrastructure for Topic-Based Publish/Subscribe

by

Saeed RAHMANI

THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT OF A MASTER'S DEGREE
WITH THESIS
M.A.Sc.

MONTREAL, JULY 31, 2023

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Julien Gascon-Samson, Thesis supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Marcos Dias De Assuncao, Chair, Board of Examiners
Department of Software Engineering and IT, École de technologie supérieure

Ms. Camille Coti, Member of the Jury
Department of Software Engineering and IT, École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON JULY 28, 2023

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I would like to express my heartfelt appreciation to my esteemed supervisor for his invaluable support, unwavering kindness, and exceptional guidance throughout the course of this research. Without his expert supervision and unwavering encouragement, the completion of this study would have been an insurmountable task. I am sincerely grateful for the privilege of being under his supervision during this research endeavor.

Furthermore, I would like to extend my profound gratitude to my beloved family, namely my mother Tahereh, my father Abbas, and my sister Nazila, for their unwavering support throughout my personal and academic journey, particularly during this significant research undertaking. Their unwavering support has contributed significantly to my achievement.

Finally, I would like to convey my gratitude to the members of the jury for their diligent review of my work and their valuable feedback. I am truly appreciative of their expertise and the time they dedicated to evaluating my study.

Thank you.

MQTT2EdgePeer : une communication Edge Peer-to-Peer robuste et évolutive Infrastructure pour la publication/abonnement thématique

Saeed RAHMANI

RÉSUMÉ

Ces dernières années, il y a eu une croissance sans précédent du nombre d'appareils Internet des objets (IoT), ce qui a entraîné une adoption généralisée d'appareils interconnectés dans divers secteurs tels que les domaines résidentiel, médical, des transports, agricole et industriel. Cette montée en flèche des dispositifs IoT a créé à la fois des opportunités et des défis, nécessitant une attention particulière aux aspects de communication dans la conception des protocoles et des systèmes IoT. Un facteur critique à prendre en compte dans la conception de la communication IoT est la nécessité d'une utilisation efficace de la bande passante. Les appareils IoT ont généralement des ressources de bande passante limitées, qui doivent être gérées avec soin pour s'adapter au volume croissant de données générées et transmises par ces appareils. Une autre considération cruciale dans la conception de la communication IoT est l'importance d'une faible latence. De nombreuses applications IoT nécessitent un échange de données en temps réel ou quasi réel pour une prise de décision et une réactivité rapides. Par conséquent, minimiser la latence, ou le délai de transmission des données, devient essentiel pour répondre aux exigences strictes des applications IoT sensibles à la latence. Dans cette étude, nous introduisons MQTT2EdgePeer, un nouveau système de publication/abonnement basé sur un sujet construit sur un réseau superposé peer-to-peer structuré pour une diffusion efficace des messages. La diffusion des messages est facilitée par des approches basées sur des coordonnateurs à saut unique et à sauts multiples. MQTT2EdgePeer s'intègre de manière transparente au protocole MQTT standard, permettant une connexion sans effort pour toutes les applications IoT. L'implémentation et le déploiement de MQTT2EdgePeer à la périphérie sont présentés, accompagnés d'évaluations expérimentales qui incluent une analyse comparative avec une approche traditionnelle d'arbre à racine unique. Les résultats obtenus valident l'efficacité de MQTT2EdgePeer sous divers aspects, notamment la répartition de la charge, la latence, l'utilisation de la bande passante, l'évolutivité et la tolérance aux pannes.

Mots-clés: Internet des objets, Publier/S'abonner, Pub/Sub thématique, MQTT, Peer-to-Peer, Edge Computing

MQTT2EdgePeer: a Robust and Scalable Peer-to-Peer Edge Communication Infrastructure for Topic-Based Publish/Subscribe

Saeed RAHMANI

ABSTRACT

In recent years, there has been an unprecedented growth in the number of Internet of Things (IoT) devices, resulting in a widespread adoption of interconnected devices across various sectors such as residential, medical, transportation, agriculture, and industrial domains. This surge in IoT devices has brought forth both opportunities and challenges, necessitating careful consideration of communication aspects in the design of IoT protocols and systems. One critical factor to consider in the design of IoT communication is the need for efficient bandwidth utilization. IoT devices typically have limited bandwidth resources, which must be carefully managed to accommodate the increasing volume of data generated and transmitted by these devices. Another crucial consideration in IoT communication design is the importance of low latency. Many IoT applications require real-time or near real-time data exchange for timely decision-making and responsiveness. Therefore, minimizing latency, or the time delay in data transmission, becomes essential to meet the stringent requirements of latency-sensitive IoT applications. In this study, we introduce MQTT2EdgePeer, a novel topic-based publish/subscribe system constructed upon a structured peer-to-peer overlay network for efficient dissemination of messages. The dissemination of messages is facilitated through both single-hop and multi-hop coordinator-based approaches. MQTT2EdgePeer seamlessly integrates with the standard MQTT protocol, allowing effortless connection for any IoT applications. The implementation and deployment of MQTT2EdgePeer at the edge are presented, accompanied by experimental evaluations that include a comparative analysis with a traditional single rooted tree approach. The results obtained validate the efficacy of MQTT2EdgePeer in various aspects, including load distribution, latency, bandwidth usage, scalability, and fault-tolerance.

Keywords: Internet of Things, Publish/Subscribe, Topic-based Pub/Sub, MQTT, Peer-to-Peer, Edge Computing

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ALGORITHMS

LIST OF ABBREVIATIONS

IoT	Internet of Things
Pub/Sub	Publish/Subscribe
MoM	Message-Oriented Middleware
P2P	Peer-to-Peer
EC	Elastic Compute
GCE	Google Compute Engine
ALG	Application-Layer-Gateway
QoS	Quality of Service
SA	Sensor/Actuator
GW	Gateway
WSN	Wireless Sensor Network
AMQP	Advance Message Queuing Protocol
DSRT	Distributed Single Root Tree
DHT	Distributed Hash Table
MoG	Multiplayer Online Game
TCO	Topic-Connected Overlay
GCD	Gapless Casual Delivery
GFD	Gapless FIFO Delivery
FIFO	First-In Fist-Out

ILDM	Interworking Layer of Distributed MQTT
RQ	Research Question
API	Application Programming Interface
CW	Clockwise
CCW	Counterclockwise
JDK	Java Development Kit

INTRODUCTION

The Internet of Things (IoT) is a paradigm that refers to a ubiquitous network of heterogeneous devices that are interconnected via the Internet. The recent advancements in IoT technologies, including hardware, software, and networking, have facilitated its rapid proliferation, with projections of up to 75 billion interconnected devices by 2025 (Alam, 2018; Jamali, Bahrami, Heidari, Allahverdizadeh & Norouzi, 2020). The integration of IoT technology has found widespread adoption across various sectors, such as smart homes, healthcare, transportation, agriculture, and industrial applications, leading to a paradigm shift in people's daily lives. By enabling efficient, productive, and safe functioning of processes and systems, IoT has gained traction in the scientific community. A smart city represents a prime example of IoT application in a large-scale environment, where all objects, such as vehicles, traffic lights, surveillance cameras, and other IoT devices, cooperate and exchange data to make informed decisions. In such a setting, vehicles exchange data, such as location, speed, traffic intensity, weather, and road conditions, with nearby devices to optimize their speed and select the most efficient route to their destination. Traffic lights are scheduled based on data received from vehicles and surveillance cameras to mitigate traffic congestion. Overall, the deployment of smart city IoT systems results in increased efficiency and productivity, reduced fuel consumption and commuting time, and enhanced safety by minimizing the risk of vehicle collisions.

Internet of Things (IoT) systems are designed to handle vast amounts of data generated by IoT objects, while the traditional approach of transmitting all the data to the cloud is unreasonable due to the limited network bandwidth capacity (Ai, Peng & Zhang, 2018). Recently, there has been a trend in IoT systems to shift the communication and computation environment from the cloud to the edge, which has emerged as a promising approach (Satyanarayanan, 2017). Edge computing provides a decentralized distributed computing environment near edge/IoT devices, which can perform with lower latency and lower costs compared to the centralized model. IoT devices, with their affordable cost and sufficient computational power, can run applications and even

operating systems (Taivalsaari & Mikkonen, 2017). Although edge computing cannot match the computation and storage capacity of cloud servers, it offers advantages such as short-distance communication and faster response time, which are highly desirable for many IoT systems.

Developing IoT systems is extremely challenging due to the heterogeneity and diversity of IoT devices and networks (Razzaque, Milojevic-Jevric, Palade & Clarke, 2015). The most ubiquitous approach for constructing communication infrastructure in IoT systems is based on the publish/subscribe (pub/sub) paradigm (Gargees & Scott, 2018; Redondi, Arcia-Moret & Manzoni, 2019; Lv, Wang, Zhu, Deng & Gu, 2019; Diro *et al.*, 2020; Hasenburg & Bermbach, 2020). The pub/sub model provides a highly decoupled communication system that can efficiently manage the massive flow of events among IoT entities, as it introduces a message-oriented middleware (MoM) for IoT device communication. In contrast to the request-reply messaging model, the pub/sub paradigm maintains a decoupling between the message producer (publisher) and consumer (subscriber), with the third component (broker) mediating messages between publishers and subscribers.

The implementation of pub/sub for IoT typically involves deploying a server, such as an edge-based or cloud-based broker, to relay messages between IoT devices (i.e., publishers and subscribers). While the pub/sub MoM efficiently decouples communications, a single broker design can limit the system in terms of resource provisioning (i.e., computation, communication, and storage), scalability, and fault-tolerance. Increasing the number of brokers can improve the system limitations compared to the single broker design. However, it still faces challenges such as message dissemination, message routing, load balancing, scalability, and resiliency. Therefore, our research focuses on building a distributed, robust, and scalable pub/sub system that leverages the capabilities of brokers' networks to address the aforementioned challenges.

0.1 Contribution

The thesis introduces MQTT2EdgePeer, a robust and scalable peer-to-peer (P2P) edge communication infrastructure for topic-based pub/sub. The system utilizes novel one-hop and multi-hop message dissemination approaches to efficiently distribute load among peers, thereby reducing latency and bandwidth usage for the entire system. Additionally, it remains available by providing a fault-tolerance mechanism and easily scales up over the unmanaged P2P overlay network. Finally, the system integrates MQTT, the standard IoT communication protocol, to make it accessible to any IoT system with an MQTT client.

0.2 Thesis organization

The thesis is organized with the following structure: Chapter 1 introduces the research by highlighting the main themes explored and conducting a comprehensive review of the existing literature. In Chapter 2, the architecture of the proposed system is discussed in detail, providing a thorough explanation of the problem statement and the objectives defined. Moving forward, Chapter 3 provides a detailed account of the software tools and libraries employed during the development of the proposed methodology. Chapter 4 focuses on the evaluation of the proposed approach within a practical Internet of Things (IoT) environment.

CHAPTER 1

BACKGROUND AND LITERATURE REVIEW

The chapter provides an introduction to key concepts that will be covered in the rest of the thesis. These include the IoT general concepts (Section 1.1), pub/sub middleware (Section 1.2), and overview of pub/sub systems (Section 1.3). It also explains the connections between the IoT and these systems, and gives a summary of existing research in the area.

1.1 IoT general concepts

In this section, we provide the definition of computing models, communication models, protocols, resource management, middleware, and how to evaluate middleware using benchmarks.

1.1.1 Cloud computing

Cloud computing has become a vital aspect of modern-day computing, and its significance has been increasing exponentially. The combination of hardware and software resources at data centers and their delivery to end-users as services over the Internet has made it possible to offer various services that were previously impossible or too expensive.

The three main categories of cloud services, SaaS, PaaS, and IaaS, provide users with different types of cloud services, each with unique characteristics. SaaS, which delivers software directly to end-users, has been adopted by many businesses and individuals. Examples of popular SaaS offerings include Google's Gmail and Microsoft's Outlook. PaaS, on the other hand, provides developers with a development platform that enables the creation of cloud-based applications. PaaS offerings such as AWS Lambda and Azure Functions have gained popularity as they enable developers to create, deploy, and run applications without worrying about underlying infrastructure.

IaaS provides virtual resources such as virtual machines, virtual storage, and virtual networks as a cloud service, allowing businesses to access computing resources on-demand without the need

to invest in physical hardware. Examples of IaaS services include Microsoft's Azure Virtual Machines, Amazon's Elastic Compute Cloud (EC2), and Google Compute Engine (GCE).

The flexibility and scalability of cloud computing services make them an attractive option for businesses looking to reduce their infrastructure costs, improve their efficiency, and enhance their agility. Moreover, cloud computing has made it possible to offer advanced services such as artificial intelligence, big data analytics, and the Internet of Things (IoT), which were previously out of reach for most businesses due to their high costs and technical complexities.

1.1.2 Edge computing

Edge computing, a relatively new paradigm, involves storing and processing data at the network edge (Yu *et al.*, 2018). The concept behind edge computing is to leverage the proximity of end-users to reduce network traffic, latency, and bandwidth usage.

The proliferation of IoT devices has resulted in an exponential increase in data generation at the edge, creating the need for an efficient and scalable way of processing and managing data. Edge computing offers an effective solution to this challenge by providing a distributed computing infrastructure that moves processing closer to where the data is generated.

One of the key advantages of edge computing is the ability to offload traffic from the IoT-to-cloud and cloud-to-IoT, resulting in reduced network congestion and improved performance. Additionally, resource provisioning at the edge, including computation and storage, ensures that data is processed and analyzed at the source, reducing latency and response times.

1.1.3 Fog computing

Fog computing is a paradigm that involves the horizontal-level union distribution of resources, such as storage and computation services, between cloud and objects at any point on the network (Ai *et al.*, 2018). Fog computing differs from edge computing in several ways.

Firstly, fog computing enables resource coordination and distribution throughout networks and among devices at the edge. In contrast, servers and application placement are integral parts of the edge architecture.

Secondly, fog computing involves collaboration between the cloud and edge, whereas the separation from the cloud is a characteristic of edge computing.

Fog computing provides several benefits compared to traditional cloud computing. By moving computation and data storage closer to the edge, fog computing offers improved response times, reduced latency, and network congestion. Additionally, fog computing can handle the increasing volume of data generated by IoT devices, allowing for more effective and efficient processing and analysis.

1.1.3.1 Cloudlet

Cloudlet refers to a smaller and more limited version of common cloud servers located at the network edge that is accessible by nearby IoT devices (Satyanarayanan, Bahl, Caceres & Davies, 2009). It provides the ability to host latency-sensitive and computation-intensive applications, such as wearable cognitive aid systems, at the edge (Ha *et al.*, 2014; Satyanarayanan *et al.*, 2015a).

Furthermore, computation at the edge through cloudlet reduces traffic to/from the cloud and increases the resilience of IoT systems in case of cloud unavailability or failure (Satyanarayanan *et al.*, 2013, 2015b).

Cloudlet provides several advantages, such as reducing network congestion, providing lower latency, and enhancing data privacy and security. Additionally, it enables new IoT applications that require high computation power and low latency.

1.1.4 Communication models

There are three main communication models for building IoT devices. The first model is device-to-device (machine-to-machine) communication, which involves low packet data transmission through a direct connection between devices. However, it suffers from the limitation of different protocols of various device manufacturers. An example of this model is shown in Figure 1.1 (A), which illustrates the direct connection between a light bulb and a light switch from different manufacturers through a wireless network.

To address the limitation of the device-to-device model, cloud services are used to connect devices to cloud servers. Figure 1.1 (B) shows the device-to-cloud model, where cloud application services enable the connection among devices with different communication protocols (i.e., HTTP, CoAP, DTLS, TLS, TCP, UDP). However, the performance of this model is highly dependent on network features such as bandwidth and distance. To address this issue, a proxy/middleware is deployed to refine and control the flow of data between the devices and the cloud server.

This leads to the third model, which is the device-to-gateway or device-to-application-layer gateway (ALG). The device-to-gateway model enables partial computations at the application layer, along with network flexibility and security improvements. Figure 1.1 (C) illustrates the local gateway control data communication between front-end sensors/devices and far-end cloud services, at a near-end region.

1.1.5 Communication Protocols

In this subsection, we will provide a brief explanation of several IoT protocols, detailing their features and outlining their benefits.

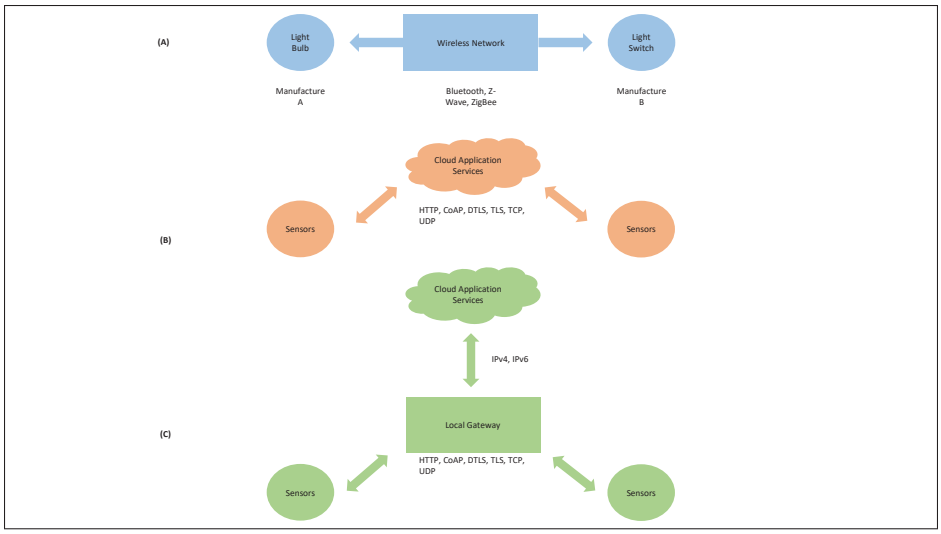


Figure 1.1 An example of different communication models for IoT systems.
Taken from (Yu *et al.*, 2018)

1.1.5.1 MQTT

MQTT is a protocol for publish/subscribe communication based on topics (MQTT, 2023). It provides a communication infrastructure that is fully decoupled, making it well-suited for IoT systems and edge computing. The protocol employs hierarchical subscriptions, using human-readable strings as topics. MQTT allows clients to initiate connections with brokers, which monitor connections for link/device failure. MQTT has different commands to control packets, such as connect, subscribe, publish, unsubscribe, disconnect, etc.

The protocol also provides three levels of Quality of Service (QoS). Level-0 has no message delivery guarantee, while level-1 provides a message delivery guarantee with the possibility of message redundancy. Level-2 provides strict message delivery guarantees and ensures that messages are received only once.

Some other MQTT features include persistent sessions and queuing messages to avoid information loss due to client-broker connection interruption, retained messages to provide an immediate update for new subscribers, last will testament messages to notify clients about unexpected

connection loss of another client, wild cards enable flexible and efficient subscription and filtering of MQTT messages, and the keep-alive function to ensure client-broker connectivity.

1.1.5.2 MQTT-S

MQTT-S is a protocol based on the standard MQTT pub/sub protocol, specifically designed for low-cost and low-power sensor/actuator (SA) nodes over bandwidth-constrained networks such as ZigBee and TinyOS (Hunkeler, Truong & Stanford-Clark, 2008). All functionalities of MQTT are included in MQTT-S, except QoS, as the MQTT-S gateway (GW) cannot delay receiving acknowledgement from the client API. To optimize performance for devices with limited resources such as processing, storage, and battery, MQTT-S performs processing at the gateway/broker side. MQTT-S also adopts an approach (i.e., splitting) to reduce message size, as wireless sensor networks (WSN), particularly ZigBee, have a packet-size constraint of 60 bytes. Connect messages are split into three short messages, with clients registering the topic and receiving the topic ID from the GW. As each client has a mapping table in the GW, it only sends the topic ID with a two-byte length instead of sending the entire topic.

1.1.5.3 AMQP

The Advanced Message Queuing Protocol (AMQP) was introduced by (Vinoski, 2006) as a standard protocol for asynchronous messaging. AMQP is a binary protocol that includes both network and semantic protocols that specify the message and its implementation. Its binary nature allows for the transfer of more data in a single packet, making it suitable for messaging systems. The protocol employs the chain of responsibility pattern, allowing processors to modify or route messages. Moreover, brokers can make routing decisions, providing a distributed and scalable messaging infrastructure.

AMQP, composed of messaging clients, brokers, exchanges, queues, bindings, routing keys, and channels, provides various guaranteed messaging modes, including at-most-once, at-least-once, and exactly-once, ensuring message delivery and reliability. Additionally, AMQP supports

multiple application-defined routing topologies, which offer flexibility while requiring some application setup. This sophisticated system provides a scalable and adaptable messaging infrastructure that can accommodate a diverse range of communication scenarios.

1.1.6 Middleware

IoT middleware solutions, such as pub/sub, facilitate the development of large-scale IoT applications by managing the data flow among devices, services, and applications (Eugster, Felber, Guerraoui & Kermarrec, 2003). Implementing middleware is a crucial process that directly affects the system's performance. A well-designed middleware leads to efficient and fast deployment, while a poorly implemented or configured one can degrade the system's performance.

A message-oriented middleware (MOM) based on the pub/sub paradigm is commonly used to build the communication infrastructure for IoT systems, typically deployed over the cloud. However, in many scenarios, an edge-based deployment (edge computing) of the MOM is better than a cloud deployment (Garcia Lopez *et al.*, 2015; Shi & Dustdar, 2016). MQTT, as the standard communication protocol for IoT systems, poses some challenges when deployed at the edge, such as client-broker mobility, topology changes (churn), and scattered resources, which add extra complexity to the systems (Satyanarayanan, 2017).

1.1.7 Resource Management

Resource management is a critical aspect of edge computing that primarily deals with the resource constraints of IoT devices such as CPU, memory, storage, battery, and bandwidth. Its key components include task scheduling, load balancing, resource allocation, and quality of service (QoS). The main objective of resource management is to minimize resource consumption, latency, and service costs while enhancing the scalability, reliability, and performance of IoT systems. Distributed storage systems like Amazon Dynamo (DeCandia *et al.*, 2007) represent a notable example of resource management that stores vast amounts of IoT system information.

Dynamo is a fully decentralized distributed key-value data store that emphasizes high availability. It employs a combination of techniques such as replication and partitioning to address issues related to failure and gossiping to detect failures. Another example of resource management is presented by Khare *et al.* (2018), who introduced a QoS approach to enhance the scalability of the communication infrastructure of IoT systems using a topic-based pub/sub middleware. Their approach provides a balance between data propagation and computation load by considering QoS for each topic.

1.1.8 Benchmarks

There are many middleware solutions available for deployment into IoT systems. Benchmarking approaches are used to determine which middleware performs correctly in which conditions and also to reduce the cost of evaluation by making it reusable for testing other middleware solutions. A necessary step in building a generic benchmark is to integrate it invisibly into IoT middleware without considering its internal implementation. Zilhão, Morla & Aguiar (2018) developed a scalable, generic benchmark architecture for IoT middlewares (i.e., publish/subscribe). They compared two pub/sub middlewares: OM2M (Alaya, Banouar, Monteil, Chassot & Drira, 2014; OM2M, 2023) and Fiware (FIWARE, 2023), employing both qualitative metrics (e.g., query possibility, supported communication model) and quantitative metrics (e.g., publish time) for measurement. The qualitative evaluation focuses on characteristics and functionalities, while the quantitative evaluation focuses on performance factors (Pereira, Cardoso, Aguiar & Morla, 2018). Moreover, the selected metrics are independent of the protocols used (e.g., MQTT, HTTP, etc.).

1.2 Publish/Subscribe middleware

This section provides an introduction to pub/sub middleware and presents an architecture that is adopted by various middleware. Additionally, it describes the different layers that constitute the architecture.

1.2.1 Introduction

The publish/subscribe communication pattern, widely used in various domains, particularly in IoT (i.e., pub/sub middleware), operates based on a message-oriented approach. In the system, two primary roles exist: the publisher and the subscriber. Each subscriber informs the system of its interest in receiving specific messages that can be published by any publisher. The interaction between the publisher and subscriber is mediated by the pub/sub system, which matches each message with its interested subscriber(s) and then delivers the message to its subscribers via a notification. The pub/sub system decouples communication in three dimensions (Eugster *et al.*, 2003). The first dimension is space decoupling, as publishers and subscribers are unaware of each other (anonymity feature). Second, they are time decoupled, as there is no need to connect to the system simultaneously. Third, they send/receive messages asynchronously while performing concurrent activities, providing synchronization decoupling. Decoupling and anonymity features make the pub/sub middleware suitable for mass dissemination of information in large-scale IoT systems.

1.2.2 Architecture

The pub/sub middleware can be divided into three functional layers: overlay network layer, routing layer, and matching layer. The functionality of the middleware is determined by the implementation of these layers. For example, the scalability of the pub/sub system depends on how the routing layer utilizes the overlay network layer to propagate messages (Baldoni & Virgillito, 2005). Figure 1.2 illustrates the layered organization of the pub/sub middleware. The application is implemented at the top of the pub/sub middleware, and the details of the middleware layers are described in subsections (1.2.3, 1.2.4, and 1.2.5). At the bottom, the transport layer facilitates data transmission among the system components by employing arbitrary network protocols such as TCP and UDP.

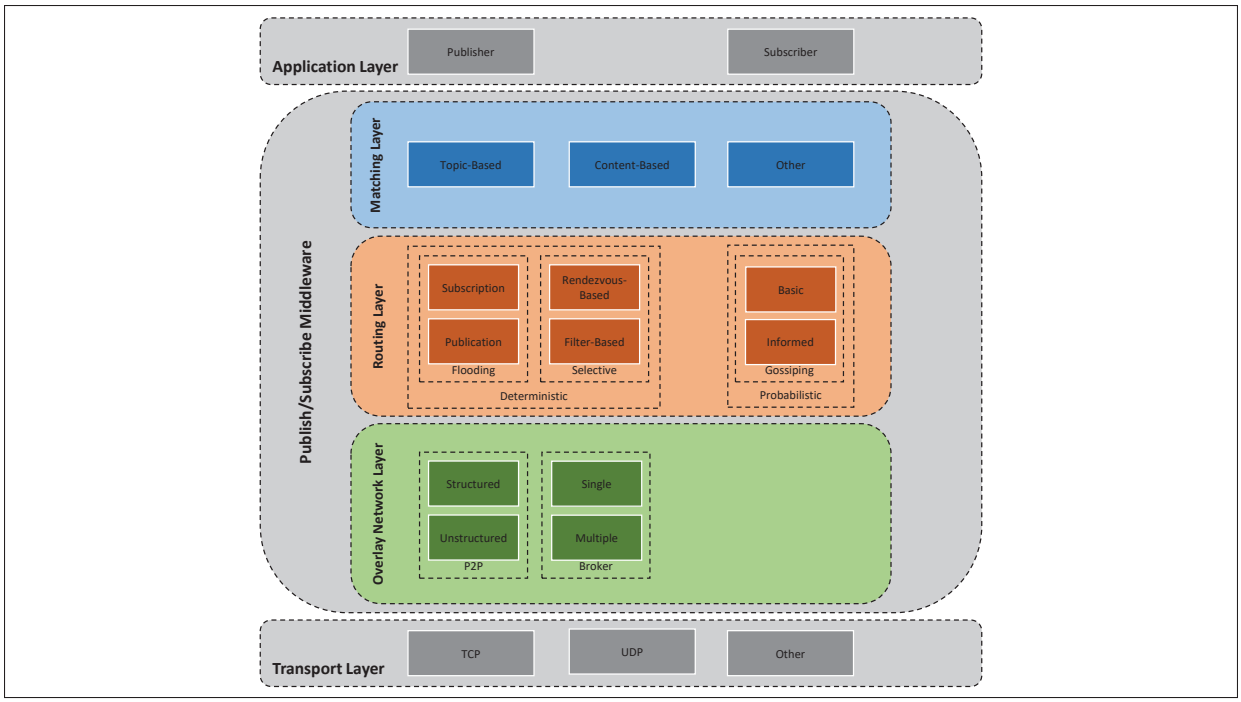


Figure 1.2 Layered organization of the Pub/Sub middleware.
Taken from Baldoni & Virgillito (2005); Esposito *et al.* (2013)

1.2.3 Matching layer (subscription model/language)

The matching layer, situated at the top of the pub/sub system, matches publications with interested subscribers based on the subscription model/language. There are various approaches to designing the subscription model, and the challenge is to set a trade-off between the expressiveness and complexity of the model.

1.2.3.1 Topic-based

Topic-based pub/sub is a communication pattern where publishers publish messages by associating them with specific topics, and subscribers receive them by subscribing to relevant topics. This creates a channel for transmitting messages between the publisher of a topic and all its subscribers (Baldoni & Virgillito, 2005). Despite its simplicity, topic-based pub/sub suffers from low expressiveness of the subscription language. To address this issue, some approaches employ

a hierarchy mechanism or wildcard instead of a flat subscription (Oki, Pfluegl, Siegel & Skeen, 1993; CORBA & Specification, 1999; Baehni, Eugster & Guerraoui, 2004).

Figure 1.3 demonstrates an example of topic-based pub/sub, where publishers on the left side publish their messages with topics a, b, and c, and the pub/sub system matches messages with interested subscribers on the right side based on the look-up table.

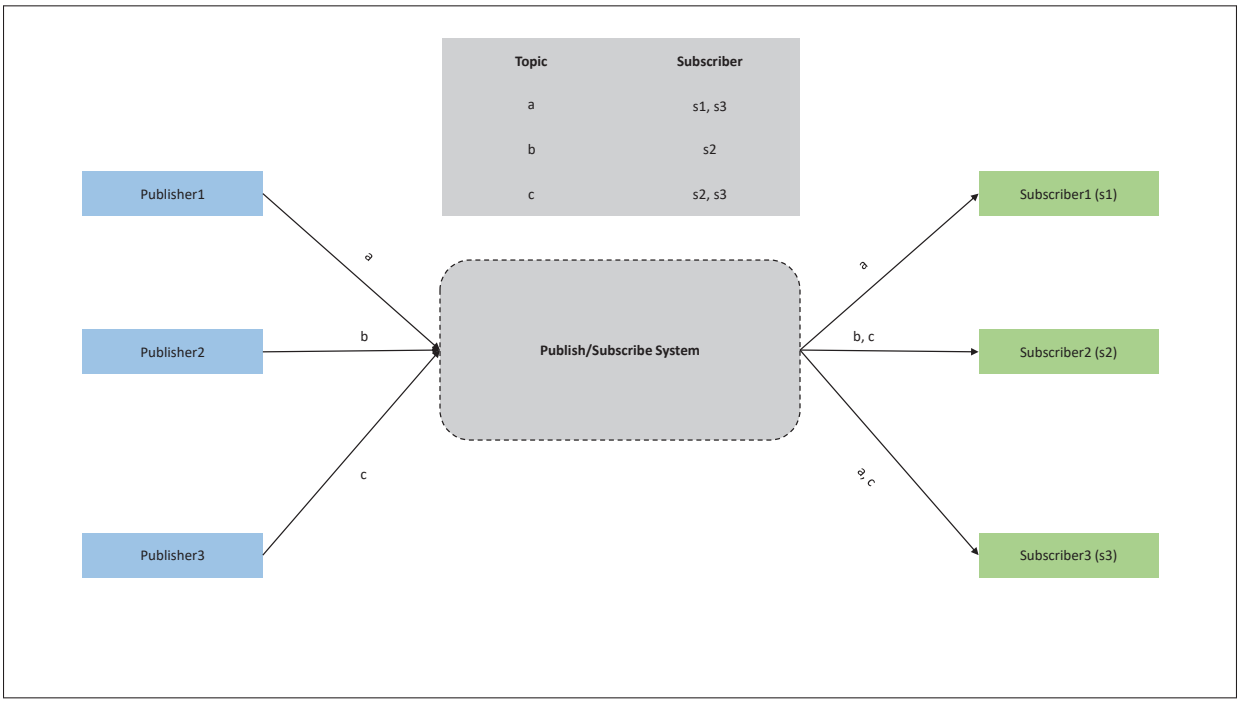


Figure 1.3 Topic-based pub/sub example

1.2.3.2 Content-based

Content-based pub/sub systems (Segall, Arnold, Boot, Henderson & Phelps, 2000) operate by subscribing to messages based on their content, applying specific constraints for the matching process. These systems prioritize the design of sophisticated subscription languages that offer enhanced flexibility and expressiveness. However, this comes at the expense of increased complexity in matching and additional runtime overhead (Carzaniga, Rosenblum & Wolf, 2001; Mühl, 2002; Pietzuch & Bacon, 2002; Aekaterinidis & Triantafillou, 2006; Li, Muthusamy & Jacobsen, 2008). An example of a content-based pub/sub system is a stock trade application, where publish-

ers disseminate stock information with various attributes such as company name, price, and date. Subscribers can receive only the desired stock information by subscribing to specific attribute constraints, such as `company name="Apple", price<100, and date=2021-08-30`.

1.2.3.3 Other subscription models

Apart from topic-based and content-based subscription models, there are other less significant models in the realm of pub/sub systems. An example of such a model is the attribute-based model, which is a subset of content-based pub/sub. Unlike content-based matching, which can involve the entire message content, the attribute-based model matches only attributes, as demonstrated by (Li, Ye, Kim, Chen & Lei, 2011). Another model is the type-based model, which restricts the type within the pub/sub system instead of the application side, as proposed by (Eugster, Guerraoui & Sventek, 2000; Eugster, Guerraoui & Damm, 2001). Some approaches also use XML documents to enhance the flexibility of the content-based model, while they often entail intensive computations, as found in studies by (Chand & Felber, 2004, 2005).

1.2.4 Routing layer

The routing layer is crucial in the pub/sub system as it is responsible for identifying subscribers and delivering messages to them. Achieving scalability is a major challenge in message routing, and it requires a trade-off between performance and the number of brokers. This trade-off should consider the overheads of the message (such as the number of hops), memory (such as the amount of stored information), and limitations of the subscription language (such as the type of constraints) (Baldoni & Virgillito, 2005). Routing algorithms can be categorized as deterministic or probabilistic depending on whether they need to maintain routing structures when subscription changes occur. Deterministic routing is ideal for networks with frequent topology changes or node churn; however, it is only suitable for small-size networks with limited dynamicity. In contrast, random routing is the primary strategy for handling a high rate of dynamicity in large networks, and it is the focus of probabilistic approaches such as gossip protocols.

1.2.4.1 Deterministic routing

Subscription flooding is a deterministic routing algorithm that disseminates publications from publishers to all nodes, which results in minimum memory overhead. This algorithm is a good option when there are mass subscriptions to most publications due to its less memory overhead. However, its drawback is the growth of message overhead with increasing scalability, and it is not suitable for a high rate of subscription changes (Segall *et al.*, 2000; Carzaniga *et al.*, 2001; Mühl, 2002). Selective routing is a better strategy to address the message overhead issue, where a subset of subscribers is stored in each node. Filter-based routing is a subset of selective routing that includes only nodes with a path to subscribers. Rendezvous-based routing is another type of selective routing that maintains forwarder nodes, called rendezvous nodes, responsible for matching. Rendezvous approaches (Castro, Druschel, Kermarrec & Rowstron, 2002; Li & Gao, 2011; Zhao, Kim & Venkatasubramanian, 2013) leverage the benefits of managed subscriptions instead of balanced subscriptions propagation (Baldoni & Virgillito, 2005). They build a distributed single root tree (DSRT) for each topic to distribute correspondent messages from the root, i.e., rendezvous point, to subscribers along tree paths. They also can have a clustering space where there would be a rendezvous point at each cluster. Although DSRT approaches (Zhuang, Zhao, Joseph, Katz & Kubiatowicz, 2001; Castro *et al.*, 2002; Li & Gao, 2011) can handle a large number of subscribers, they can have root contention issues in case of a high rate of publications.

1.2.4.2 Probabilistic routing

The complexity of probabilistic (random) routing is comparatively lower than deterministic routing. Gossiping (epidemic) protocols are widely used in pub/sub systems with probabilistic routing (Eugster & Guerraoui, 2002; Costa, Migliavacca, Picco & Cugola, 2003; Baehni *et al.*, 2004; Costa & Picco, 2005). In gossip-based protocols, each node exchanges information randomly with other nodes in each round, without storing or tracking subscription changes and node churns. They benefit from perfectly random message distribution, which is suitable for systems with mobility and numerous nodes. Redundancy and message overhead are the

two primary pitfalls of random routing. To mitigate these issues, some approaches filter each gossip by sending it only to subscribers, called informed-gossiping (Eugster & Guerraoui, 2002; Baldoni & Virgillito, 2005).

1.2.5 Overlay network layer

Distributed pub/sub systems at the application layer level are characterized by the virtual organization of nodes, known as the overlay. The overlay network layer is responsible for the organization of nodes in a specific configuration to disseminate publications from publishers to interested subscribers. The physical-level formation of the nodes may vary, as the application level governs the formation of the pub/sub system. The organization can operate in a peer-to-peer (P2P) or federated manner, with or without centralized management, respectively, as documented in prior literature (Castro *et al.*, 2002; Tam, Azimi & Jacobsen, 2004; Muthusamy & Jacobsen, 2005; Baldoni, Beraldi, Quema, Querzoni & Tucci-Piergiovanni, 2007; Muthusamy & Jacobsen, 2013). The overlay network has a direct influence on the pub/sub system's performance, notably its latency, scalability, and routing, by regulating the node degrees, number of hops, and path diameters, as evidenced in existing research (Chen & Tock, 2015; Chen, Jacobsen & Vitenberg, 2016; Chen, Tock & Girdzijauskas, 2018). The overlay network is classified into four categories: structured P2P overlay, unstructured P2P overlay, hybrid overlay (structured and unstructured combination), and broker overlay.

1.2.5.1 Structured P2P overlay

A structured P2P overlay network is responsible for assigning a unique key to each node from a virtual key space, providing a guaranteed path between each pair of nodes, facilitating node discovery, and delivering a self-organized overlay network. Among pub/sub approaches, the development of topic-based pub/sub systems based on P2P structured overlay with a distributed hash table (DHT) is widely adopted, as reported in previous studies (Zhuang *et al.*, 2001; Castro *et al.*, 2002; Li & Gao, 2011). DHT partitions topics uniformly among peers, where the peer closest to each topic name hash becomes the RP or root of its spanning tree. The size and

latency of the tree are influenced by tree members, including interested/non-interested peers. Additionally, there exists a direct relationship between the popularity of a topic and its root load, which can result in a hotspot for message dissemination. Two examples of real-world applications that typically experience a high rate of publications for popular topics (hot-topic) are Twitter (Sanlı & Lambiotte, 2015) and multiplayer online games (MOG) (Arantes, Potop-Butucaru, Sens & Valero, 2010; Gascon-Samson, Kienzle & Kemme, 2015b).

1.2.5.2 Unstructured P2P overlay

The P2P unstructured overlay is a routing structure that employs probabilistic routing (i.e., gossiping) and provides an unmanaged topology. The topic-connected overlay (TCO), a widely used method for constructing unstructured P2P pub/sub systems, maintains connections between nodes with the same topic in sub-overlays, as evidenced in prior research (Chockler, Melamed, Tock & Vitenberg, 2007). TCO effectively prevents publish messages from being forwarded to non-interested nodes and is sometimes referred to as a relay-free overlay due to the absence of intermediate relay nodes. TCO facilitates efficient routing and is a suitable choice for sensitive data dissemination among a group of trusted users. However, TCO's reliability is a significant concern, as the topic connectivity can be disrupted by a single node failure, as previously discussed (Chockler *et al.*, 2007; Onus & Richa, 2011, 2016).

1.2.5.3 Hybrid overlay

The hybrid overlay is a combination of structured and unstructured overlays. Setty, Steen, Vitenberg & Voulgaris (2012) propose a decentralized pub/sub system that integrates structured and unstructured overlays using filter-based and gossip-based routing. The authors highlight that the proposed architecture is highly robust with respect to scalability, efficiency, and fault tolerance.

1.2.5.4 Broker overlay

Pub/sub systems require a network of servers and an overlay construction at the application layer in order to scale up to the size of the internet for running distributed applications. In the context of pub/sub systems, a broker is a term used to refer to a server. A single broker provides a client-server pub/sub system, while an overlay construction with a set of nodes and a managed topology (such as flat or hierarchical) enables the creation of a distributed pub/sub system with a broker overlay.

Figure 1.4 illustrates how publishers (represented by blue circles) and subscribers (represented by green circles) are connected to local brokers (represented by gray squares), as well as the interaction among brokers (represented by white squares) within the broker overlay of a pub/sub system.

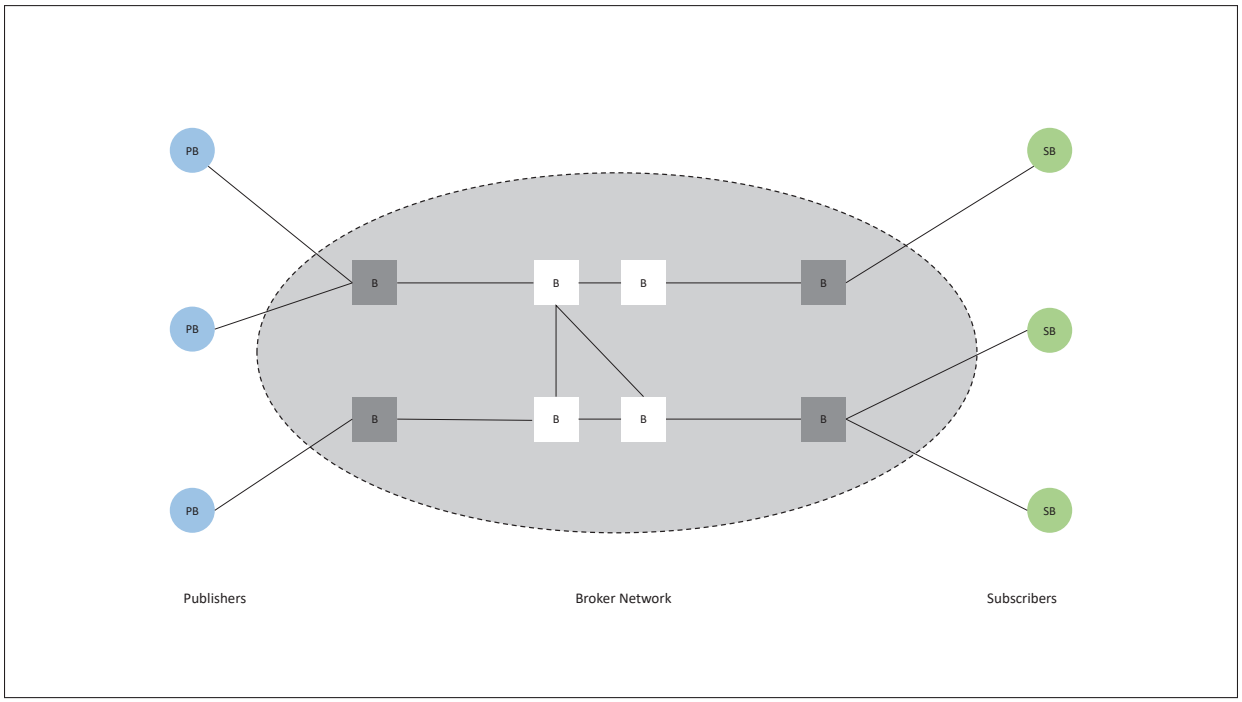


Figure 1.4 Broker overlay of a pub/sub system

1.3 Overview of pub/sub systems

In this section, we review some state-of-the-art topic-based publish/subscribe (pub/sub) systems and classify them based on their characteristics into generic, cloud-based, edge-based, and MQTT pub/sub systems. Furthermore, we present a comparative analysis of these pub/sub systems through figures 1.6, 1.7, 1.8, 1.9, and 1.10, which demonstrate how modifying the design of each system can make it suitable for different scenarios.

1.3.1 Generic pub/sub systems

This subsection provides an overview of pub/sub systems with a particular emphasis on the following key areas: reliability, load balancing, exhaust-data handling, TCO, and peer-to-peer.

1.3.1.1 Pub/Sub proactive and reactive reliabilities

The reliability of distributed pub/sub systems can be maintained proactively or reactively in the event of a failure (Chen, Vitenberg & Jacobsen, 2021). Proactive maintenance is exemplified by topic-connected overlay systems with a k factor (kTCO), where k distinct paths exist between each pair of nodes in the kTCO. On the other hand, reactive maintenance involves making constant system adjustments to preserve reliability. However, the cost of such adjustments after each churn event (i.e., when nodes join or leave frequently) is significant. In the case of a TCO pub/sub system, the system may become inaccessible during reconstruction. Nonetheless, TCO with a k factor as small as two can keep the system functional during maintenance in the event of less than k node failures. Chen *et al.* (2021) examine the trade-off between low node degrees and maintaining acceptable runtime for kTCO.

1.3.1.2 Pub/Sub GCD and GFD reliability variables

In distributed pub/sub systems, there exists a time gap between the moment a new subscriber joins the system and when it receives its first message from the corresponding publisher(s). This occurs because the system does not deliver publish messages to a newly joined subscriber until

the subscribe message is propagated to all relevant publishers. The length of the subscription gap (delay) is determined by the functions and algorithms of each pub/sub system. To reduce this delay, Pedrosa & Rodrigues (2021) categorize the gap into two reliability variables: gapless FIFO delivery (GFD) and gapless casual delivery (GCD). GFD only considers the order of the messages, while GCD prevents anomalies resulting from publishers' interactions and provides stronger reliability. Several systems offer GFD (Bhola, Strom, Bagchi, Zhao & Auerbach, 2002; Zhao, Sturman & Bhola, 2004b) and GCD (Prakash, Raynal & Singhal, 1996; Cugola, Di Nitto & Fuggetta, 2001; Pereira, Lobato, Teixeira & Pimentel, 2008; Nakayama, Duolikun, Aikebaiery, Enokidoz & Takizaw, 2014; de Araujo, Arantes, Duarte Jr, Rodrigues & Sens, 2019) reliabilities. Pedrosa & Rodrigues (2021) explore the relationship between new and previous subscriptions and propose an algorithm (LoCAPS) based on their previous work (Santos & Rodrigues, 2019) that supports GCD reliability variable.

1.3.1.3 Pub/Sub load balancing

The popularity of topics in pub/sub systems can vary greatly, with some experiencing high subscription rates and becoming hot topics, while others have relatively few subscribers. This skewed distribution of subscriptions in DHT pub/sub systems can cause load imbalances for brokers, which can in turn affect system performance, including throughput and latency. This is particularly critical for real-world pub/sub applications, such as traffic monitoring and stock market systems (Demers, Gehrke, Hong, Riedewald & White, 2006; Panagiotou *et al.*, 2016; Zacheilas *et al.*, 2017). Load distribution has been shown to be an NP-hard problem Garey & Johnson (2002); Dedousis, Zacheilas & Kalogeraki (2018). (Kreps, Narkhede, Rao *et al.*, 2011) propose a greedy algorithm for load distribution in Apache Kafka pub/sub brokers. The algorithm works by migrating loads from the most overloaded brokers to the least overloaded ones.

1.3.1.4 Pub/Sub exhaust-data handling

The low/no value data density, also known as exhaust data, is a data type that includes a significant portion of IoT system data types (Manyika *et al.*, 2011). Pub/Sub systems over structured overlays do not perform efficiently for low/no value data scenarios. The problem is that they have no mechanism to stop or control subscription forwarding and tree constructions when there are no or limited subscribers (Banno *et al.*, 2015). In contrast to structured overlay networks (DHTs) (Rowstron & Druschel, 2001; Ratnasamy, Francis, Handley, Karp & Shenker, 2001; Zhao *et al.*, 2004a), unstructured overlay networks (relay-free) (Chockler *et al.*, 2007; Setty *et al.*, 2012) provide the ability to control the subscription forwarding paths as subgraphs connect publishers and subscribers. (Banno *et al.*, 2015) take advantage of the relay-free overlay with their Skip Graph approach for no/low subscriber detection and system adjustment.

1.3.1.5 Pub/Sub TCO

Minimizing the degree of the nodes is one of the main objectives when designing robust TCO overlay networks. A lower node degree enables a reduction in maintaining outgoing links costs, resource consumption, message queues, routing tables, overlay diameter, and message latency (Chockler *et al.*, 2007). Chockler *et al.* (2007) prove the NP-hardness of building TCO with the minimum average node degree problem and propose a greedy algorithm for it. Designing a mechanism to track changes is another issue for TCOs to ensure reliability and scalability while preventing costly overlay reconstruction. Chen *et al.* (2016) introduce a divide-and-conquer algorithm to construct a TCO overlay. It employs a bulk lightweight partitioning approach to partition the network, builds sub-TCOs locally, and finally merges them into a single global TCO. Their experiments show a reduction in TCO construction time from scratch with a negligible node degree increase.

1.3.1.6 Pub/Sub peer-to-peer

A peer-to-peer substrate serves as the underlying framework for object location and routing in pub/sub systems. Castro *et al.* (2002) introduced Scribe, an expansive and entirely decentralized event notification system constructed upon the foundation of Pastry (Rowstron & Druschel, 2001), a generic P2P routing system that operates on a self-organizing overlay network comprised of interconnected nodes. Scribe encompasses the vital functionalities of topic and subscription maintenance, as well as the facilitation of efficient multicast tree construction. (Setty *et al.*, 2012) presented a hybridized strategy that combines deterministic propagation across sustained rings with probabilistic dissemination via a restricted count of random shortcuts. This distinctive approach effectively addresses the constraints associated with scalability and fault-tolerance inherent in peer-to-peer (P2P) methodologies. Furthermore, within the system, a predetermined maximum dissemination fan-out is established for each topic, typically set at 2.

In contrast to (Castro *et al.*, 2002; Setty *et al.*, 2012), our approach implements coordinator-based message dissemination instead of constructing spanning trees. Additionally, we introduce shadow properties to enhance fault-tolerance, enable customization of fan-out ranges, and ensure compatibility with the widely recognized MQTT protocol. This design is particularly well-suited for edge deployment, allowing for seamless integration and efficient utilization of edge computing resources.

1.3.2 Cloud-based pub/sub systems

In this subsection, we offer an overview of pub/sub systems designed for cloud deployment.

1.3.2.1 Scalable and elastic cloud pub/sub

The growth of smart IoT systems is increasing the demand for elastic and scalable pub/sub cloud services. Elasticity is a concern for clients to match their demands with the pub/sub cloud service resource provisioning, system scalability, and costs. Li *et al.* (2011) design a pub/sub

cloud service that places brokers over a one-hop overlay, leverages subscription skewness by assigning them to multiple brokers, and offers scalability and fault-tolerance.

1.3.2.2 Load balancing in cloud pub/sub

Consistent hashing approaches (Aurenhammer, 1991) guarantee the uniform distribution of topics over pub/sub brokers. However, the approach is not efficient in some scenarios, specifically when brokers experience different workloads. Gascon-Samson, Garcia, Kemme & Kienzle (2015a) propose a pub/sub middleware with a novel load balancer to address the limitation of the consistent hashing approach and improve scalability and elasticity for cloud-based systems.

1.3.3 Edge-based pub/sub systems

In this subsection, we present an overview of pub/sub systems with a focus on edge deployment.

1.3.3.1 Topic partitioning in edge pub/sub

Native proximity among edge nodes and clients provides lower latency for deploying pub/sub systems compared to a cloud-based environment (Gupta, Landle & Ramachandran, 2021). Consistent hashing is a prevalent approach for distributing load among pub/sub brokers, producing a consistent hash of each topic that guarantees even load distribution. However, it does not consider the client-broker vicinity and client mobility, which are key parameters for reducing latency in edge systems. Moreover, constrained resources at the edge and the possibility of hot spots with consistent hashing approaches have led to the development of load-aware topic partitioning systems (Khare *et al.*, 2018). To address these limitations, Gupta *et al.* (2021) employ a technique (Dabek, Cox, Kaashoek & Morris, 2004; Ledlie, Gardner & Seltzer, 2007) to estimate the load between each pair in the system and maintain it under a predefined threshold.

1.3.3.2 Fog computing in edge pub/sub

Broker-based pub/sub systems can take advantage of fog (cloud-edge) computing (Bermbach *et al.*, 2017) for scalability and immediacy at the same time, by deploying system components close to clients at the edge and inside or close to the cloud. There are two ways for message dissemination in a pub/sub system with fog deployment. The first strategy provides lower latency, where brokers at the edge are interconnected, and clients send/get messages to/from edge brokers, while the second strategy enables higher scalability by forwarding extra data from edge-to-cloud or cloud-to-edge. Hasenburg, Stanek, Tschorsch & Bermbach (2020) propose a fog-based pub/sub system that controls the trade-off between latency and extra data forwarding. They divide brokers into broadcast groups that flood messages among each other, while the cloud mediates messages between groups as the relay.

1.3.3.3 Edge pub/sub and QoS

Pub/sub systems at the edge leverage client vicinity to provide lower latency as well as lower bandwidth usage by pre-processing the data (i.e., data aggregation, data anonymity, data refinement) before forwarding it to the cloud or replying to clients. Despite the lower latency of pub/sub brokers at the edge, it does not guarantee the quality of service (QoS). Khare *et al.* (2018) introduced a prediction model to determine the number of required brokers and the placement of topics on them to keep the latency for subscribers below a specific QoS value.

1.3.3.4 Tail latency in edge pub/sub

When comparing communication models for IoT pub/sub systems, device-to-device communication of resource-constrained IoT devices does not guarantee the minimum latency compared to other communication models (i.e., device-to-edge/gw, device-to-cloud). Some experiments for distributed large-scale IoT systems also show that device-to-edge with a 1-hop broker can bring lower latency than device-to-device communication (Abdelwahab & Hamdaoui, 2016). Although a one-hop relay transfers computation to the edge/cloudlet side instead of

devices, it may experience long-tail end-to-end latency due to computation and communication (Abdelwahab & Hamdaoui, 2016). It is possible to reduce tail end-to-end latency with a migration strategy and statistical measurements when a relay dispatches data to long-range devices. (Abdelwahab & Hamdaoui, 2016) employed a distributed flocking algorithm to enable autonomous migration of broker clones among edge sites based on the latency and data statistics of each host.

1.3.4 MQTT pub/sub systems

This subsection presents an overview of pub/sub systems that rely on the MQTT protocol.

1.3.4.1 Auto discovery for MQTT brokers

The MQTT protocol is a widely used option for IoT system implementation, providing a pub/sub paradigm with a traditional client-server model, where publishers/subscribers connect to a broker forming a star-shaped topology. To expand the broker network, some MQTT implementations offer bridging mechanisms that connect MQTT brokers. However, a larger network may increase the complexity of static bridge configuration setup and create looping issues, which are challenging to debug in large networks. Manual configuration also limits the system in terms of fault tolerance and scalability. To address these issues, Longo et al. (2020) have introduced a Spanning Tree Protocol-based algorithm, called MQTT-ST, which allows brokers to be automatically organized in the network without creating loops and with robustness in case of failure. Furthermore, Stagliano et al. (2021) have utilized MQTT-ST as the overlay network of D-MQTT, a distributed implementation of MQTT, replacing the static bridge configuration with an automatic broker discovery technique that involves sending discovery messages periodically from each broker to others, and providing a routing algorithm to route publish messages to only interested brokers. In contrast to the aforementioned studies, our system employs a structured overlay network, which provides auto-discovery, self-organization, scalability, and fault-tolerance.

1.3.4.2 MQTT in industry

Distributed IoT systems implementation and deployment specifically in the industry domain come with challenges such as communication, maintenance, scalability, and cybersecurity due to their complexity and heterogeneous nature (Sisinni, Saifullah, Han, Jennehag & Gidlund, 2018). Amoretti, Pecori, Protskaya, Veltri & Zanichelli (2020) develop a bridging mechanism and introduce new authentication and authorization schemes to improve scalability and cybersecurity for MQTT.

1.3.4.3 Cooperation of MQTT brokers

The term "edge-heavy data" refers to a set of characteristics in IoT systems at the edge, including data production at the edge, data localization, and real-time data exchange (Okanohara, Hido, Kubota, Unno & Maruyama, 2013). Figure 1.5 presents two sample architectures for edge-heavy data distribution, where edge-based MQTT brokers offer lower latency and better throughput than cloud-based brokers. However, heterogeneous MQTT brokers' cooperation remains a challenge, which can be addressed by the Interworking Layer of Distributed MQTT brokers (ILDM) (Banno, Sun, Fujita, Takeuchi & Shudo, 2017). ILDM allows different brokers to collaborate and offers API sets for the development of cooperation algorithms. Although some MQTT implementations have introduced bridging and clustering mechanisms (Light, 2017a; HiveMQ, 2023), it is not yet a part of the MQTT protocol (MQTT, 2023).

1.3.4.4 MQTT client-broker connections

In the distributed implementation of the MQTT protocol, clients need prior knowledge to connect to the proper broker that hosts their interested topic. Hmissi & Ouni (2022) provides an approach to transparently connect subscribers to the responsible broker at joining time. It checks the broker's address and status which hosts the client's interested topics in specific intervals.

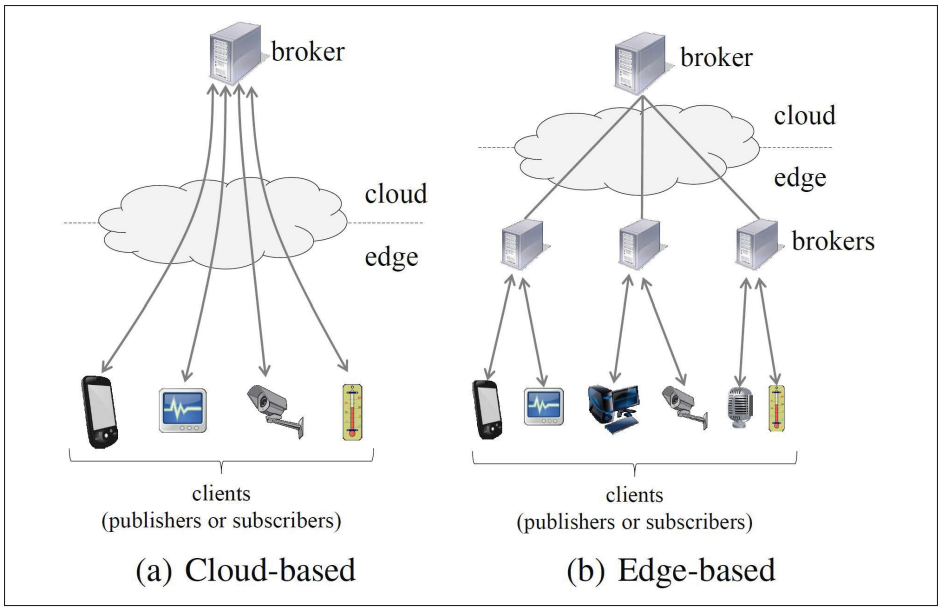


Figure 1.5 Samples data distribution architectures for edge-heavy data
Taken from Banno *et al.* (2017)

1.3.4.5 Scalability and latency of MQTT brokers

Banno & Shudo (2020) perform a low-scale test over MQTT brokers to demonstrate the relation between the number of MQTT brokers, throughput, and latency. Although the test is limited, it emphasizes that there is a trade-off between system scalability and response time. To respect the trade-off, they present a dynamic overlay that can form one-hop or multiple-hop connected subgraphs over each topic by considering the topic load.

Paper/Publish-Subscribe system	Matching			Routing				Overlay			
	Topic-based	Content-Based	Others (i.e., attribute-based, type-based)	Filter-based	Rendezvous-based	Gossiping	Flooding	Structured	Unstructured	Hybrid	Broker
Dynatops (Zhao, Kim et Venkatasubramanian, 2013)	✓				✓			✓			
FogMQ (Abdelwahab et Hamdaoui, 2016)	✓					✓					✓
Poldercast (Setty et al., 2012)	✓			✓		✓				✓	
EMMA (Rausch, Nastic et Dustdar, 2018)	✓						✓				✓
(Chen, Jacobsen et Vitenberg, 2016)	✓					✓			✓		
Dynamoth(Gascon-Samson et al., 2015)	✓						✓				✓
VCube-PS (de Araujo et al., 2019)	✓				✓			✓			
ePulsar (Gupta, Landle et Ramachandran, 2018)	✓			✓							✓
(Banno et al., 2015)	✓			✓					✓		
(Dedousis, Zachelias et Kalogeraki, 2018)	✓			✓							✓
MQTT-ST (Longo et al., 2020)	✓						✓				✓
D-MQTT (Stagliano, Longo et Redondi, 2021)	✓			✓			✓				✓
TD-MQTT (Hmissi et Ouni, 202)	✓			✓							✓
(Banno et Shudo, 2020)	✓			✓				✓			
(Chen, Vitenberg et Jacobsen, 2021)	✓					✓			✓		
(Pedrosa et Rodrigues, 2021)	✓			✓							✓
(Hasenburg et al., 2020)	✓						✓				✓
(Amoretti et al., 2021)	✓						✓				✓
(Khare et al., 2018)	✓						✓				✓
(Banno et al., 2017)	✓						✓				✓
Scribe (Castro et al., 2002)	✓				✓			✓			
Ours	✓			✓	✓			✓			

Figure 1.6 List one of the pub/sub systems

Paper/Publish-Subscribe system	Organization		Partitioner				Implementation	
	P2P (unmanaged)	Federated (managed)	DHT	Configuration (i.e., bridge, cluster, graph)	VCube	TCO	Real-word testbed/VMs	Simulation
Dynatops (Zhao, Kim et Venkatasubramanian, 2013)	✓		✓					✓
FogMQ (Abdelwahab et Hamdaoui, 2016)	✓			✓			✓	
Poldercast (Setty et al., 2012)	✓					✓		✓
EMMA (Rausch, Nastic et Dustdar, 2018)		✓		✓			✓	
(Chen, Jacobsen et Vitenberg, 2016)	✓					✓		✓
Dynamoth(Gascon-Samson et al., 2015)		✓	✓	✓			✓	
VCube-PS (de Araujo et al., 2019)	✓				✓			✓
ePulsar (Gupta, Landle et Ramachandran, 2018)		✓		✓			✓	
(Banno et al., 2015)		✓				✓		✓
(Dedousis, Zacheilas et Kalogeraki, 2018)		✓		✓			✓	
MQTT-ST (Longo et al., 2020)		✓		✓				✓
D-MQTT (Staglianò, Longo et Redondi, 2021)		✓		✓			✓	
TD-MQTT (Hmissi et Ouni, 2022)		✓		✓				✓
(Banno et Shudo, 2020)		✓		✓				✓
(Chen, Vitenberg et Jacobsen, 2021)	✓					✓		✓
(Pedrosa et Rodrigues, 2021)		✓		✓				✓
(Hasenburg et al., 2020)		✓		✓				✓
(Amoretti et al., 2021)		✓		✓			✓	
(Khare et al., 2018)		✓		✓			✓	
(Banno et al., 2017)		✓		✓			✓	
Scribe (Castro et al., 2002)	✓		✓	✓				
Ours	✓		✓	✓			✓	

Figure 1.7 List two of the pub/sub systems

Paper/Publish-Subscribe system	Proposed feature								
	Cost-driven reconfiguration plan	Hybrid routing	The skewness of data distribution	Casual delivery order	Shadow properties	Custom fan-out	Coordinator-based	Self-organized
Dynatops (Zhao, Kim et Venkatasubramanian, 2013)	✓							
FogMQ (Abdelwahab et Hamdaoui, 2016)								✓
Poldercast (Setty et al., 2012)		✓						✓
EMMA (Rausch, Nastic et Dustdar, 2018)								
(Chen, Jacobsen et Vitenberg, 2016)								✓
Dynamoth(Gascon-Samson et al., 2015)							✓	
VCube-PS (de Araujo et al., 2019)				✓				✓
ePulsar (Gupta, Landle et Ramachandran, 2018)								
(Banno et al., 2015)								
(Dedousis, Zachelias et Kalogeraki, 2018)			✓					
MQTT-ST (Longo et al., 2020)								
D-MQTT (Stagliano, Longo et Redondi, 2021)								✓
TD-MQTT (Hmissi et Ouni, 2022)								
(Banno et Shudo, 2020)							✓	
(Chen, Vitenberg et Jacobsen, 2021)								✓
(Pedrosa et Rodrigues, 2021)				✓				
(Hasenburg et al., 2020)								
(Amoretti et al., 2021)								
(Khare et al., 2018)								
(Banno et al., 2017)								
Scribe (Castro et al., 2002)								✓
Ours					✓	✓	✓	✓

Figure 1.8 List three of the pub/sub systems

Paper/Publish-Subscribe system	Proposed feature						
	MQTT compatibility	QoS optimization	Low maintenance cost	Forwarding paths adjustment	Location-based user placement	Proximity (vicinity) neighbor selection (PSN)	Hierarchical load balancer
Dynatops (Zhao, Kim et Venkatasubramanian, 2013)			✓		✓	✓	
FogMQ (Abdelwahab et Hamdaoui, 2016)							
Poldercast (Setty et al., 2012)			✓			✓	
EMMA (Rausch, Nastic et Dustdar, 2018)	✓	✓			✓		
(Chen, Jacobsen et Vitenberg, 2016)			✓				
Dynamoth(Gascon-Samson et al., 2015)							✓
VCube-PS (de Araujo et al., 2019)							
ePulsar (Gupta, Landle et Ramachandran, 2018)					✓	✓	
(Banno et al., 2015)				✓			
(Dedousis, Zacheilas et Kalogeraki, 2018)							
MQTT-ST (Longo et al., 2020)	✓						
D-MQTT (Staglianò, Longo et Redondi, 2021)	✓						
TD-MQTT (Hmissi et Ouni, 2022)	✓				✓		
(Banno et Shudo, 2020)							
(Chen, Vitenberg et Jacobsen, 2021)							
(Pedrosa et Rodrigues, 2021)							
(Hasenburg et al., 2020)	✓			✓			
(Amoretti et al., 2021)	✓						
(Khare et al., 2018)	✓	✓					
(Banno et al., 2017)	✓						
Scribe (Castro et al., 2002)							
Ours	✓						

Figure 1.9 List four of the pub/sub systems

Paper/Publish-Subscribe system	Proper scenario					
	Dynamic subscription/publication	Edge/IoT deployment	Exhaust data	Heavy publication load	Node churn
Dynatops (Zhao, Kim et Venkatasubramanian, 2013)	✓				
FogMQ (Abdelwahab et Hamdaoui, 2016)		✓			
Poldercast (Setty et al., 2012)					✓
EMMA (Rausch, Nastic et Dustdar, 2018)		✓			
(Chen, Jacobsen et Vitenberg, 2016)	✓				
Dynamoth(Gascon-Samson et al., 2015)	✓				
VCube-PS (de Araujo et al., 2019)				✓	
ePulsar (Gupta, Landle et Ramachandran, 2018)		✓			
(Banno et al., 2015)			✓		
(Dedousis, Zachelias et Kalogeraki, 2018)					
MQTT-ST (Longo et al., 2020)		✓			
D-MQTT (Stagliano, Longo et Redondi, 2021)		✓			
TD-MQTT (Hmissi et Ouni, 2022)		✓			
(Banno et Shudo, 2020)			✓		
(Chen, Vitenberg et Jacobsen, 2021)					✓
(Pedrosa et Rodrigues, 2021)	✓				
(Hasenburg et al., 2020)		✓			
(Amoretti et al., 2021)		✓			
(Khare et al., 2018)		✓			
(Banno et al., 2017)		✓			
Scribe (Castro et al., 2002)	✓				
Ours	✓	✓		✓	

Figure 1.10 List five of the pub/sub systems

CHAPTER 2

APPROACH

This chapter outlines the methodology used in our research. We present the main problem or gap in knowledge that the research is addressing (2.1), the research goals and objectives (2.2), the list of assumptions (2.3), the list of the specific questions that the research aims to answer (2.4), the key components of the system and how they are integrated (2.5), the system's overlay utilization (2.6), the system's commumications (2.7), the parameters that are used in the model (2.8), the system's protocol compatibility (2.9), the system's fault-tolerance (2.10), and the algorithms that are used in the research (2.11). Overall, the chapter provides a comprehensive overview of the research project, from the problem being addressed to the methods and algorithms used to investigate it.

2.1 Research problem

The performance of a distributed topic-based pub/sub system is directly influenced by the overlay network and routing algorithm employed. These factors impact various aspects such as scalability, robustness, fault-tolerance, and load balancing. In the previous chapter, we reviewed several approaches that utilize distinguished overlay networks and message dissemination techniques. Our focus is on the structured P2P overlay network and its application for building distributed topic-based pub/sub. A common approach for routing pub/sub messages over the structured P2P overlay network is to create a distributed single-root tree (DSRT) for each topic. DSRT routes all pub/sub messages to the root (rendezvous point/RP), and the tree consists of nodes among the path between subscribers to the root. However, message distribution from a single point has a high potential to become a bottleneck by receiving a high load for a correspondent topic. Additionally, DSRT approaches are fragile and undergo a significant tree reconfiguration (reconstruction) cost in case of node churns and dynamic subscriptions. In summary, these issues degrade and limit the functionality of DSRT approaches to run distributed pub/sub applications.

From an IoT perspective, MQTT is a dominant protocol proposed for implementing IoT system communications. It utilizes the pub/sub paradigm with the traditional client-server architecture. The MQTT broker serves as the server side of the protocol that disseminates messages among its connected MQTT clients (subscriber/publisher). Although the client-server design enables lightweight and simple message distribution, it cannot efficiently support distributed large-scale applications with a single broker. Distributed IoT applications require a network of MQTT brokers to cooperate with each other for message dissemination. Some MQTT implementations have features to connect MQTT brokers to each other. However, all settings and configurations for connecting brokers need to be managed, which can cause other issues such as message loops and difficult debugging. In other words, unmanaged decentralized mechanisms are not included in the standard MQTT protocol to construct distributed IoT applications.

2.2 Design goals and hypothesis

In this thesis, we propose MQTT2EdgePeer as a novel approach to address the limitations of existing DSRT approaches in building distributed MQTT brokers for large-scale IoT applications. MQTT2EdgePeer is designed as a scalable and robust P2P edge communication infrastructure for topic-based pub/sub. The system leverages the structured P2P overlay network to distribute messages in a coordinated manner, ensuring load balancing, fault-tolerance, and scalability.

The overlay network is utilized as a fully unmanaged and self-organizing P2P infrastructure to construct a distributed MQTT broker overlay network at the edge. MQTT2EdgePeer provides a transparent distributed MQTT broker side that any existing IoT applications with a standard MQTT client side can connect to. This enables IoT applications to leverage the benefits of the distributed MQTT broker network without the need for additional configuration or changes in the existing MQTT client side.

In this thesis, we focus on the implementation of a subset of the MQTT standard protocol, including subscribe, unsubscribe, and publish operations, on the distributed MQTT broker network built by MQTT2EdgePeer. Our experimental results demonstrate that the proposed

approach is scalable and efficient, achieving better performance than the traditional DSRT-based MQTT brokers, particularly in terms of load balancing, fault tolerance, and message delivery.

We aim to achieve the following objectives:

- Disseminate messages using two dispatching algorithms: Direct Dispatching (Section 2.11.1) and Guided Dispatching (Section 2.11.2).
- Balance topic load by distributing messages from multiple points, instead of a single rendezvous point per topic.
- Balance resource consumption among peers through Direct Dispatching and Guided Dispatching methods.
- Improve the fault-tolerance and scalability of the overlay through replication and maintenance techniques.
- Build a distributed MQTT system over the improved structured P2P overlay network.
- Utilize the overlay for MQTT broker discovery, cooperation, routing, and message distribution.

2.3 Assumptions

We consider some assumptions in this thesis and list them as follows:

- The connections between MQTT clients and MQTT2EdgePeer are stable.
- The subscriber, publisher, and shadow nodes in the overlay network are stable.
- Shadow nodes constantly maintain correct lists of subscribers and related topics as they are stable.
- Coordinator and root (rendezvous point) nodes are not stable and are susceptible to experiencing failures, such as network link outages, crashes of apps or devices, or other disruptive events. It is important to consider these failure scenarios in order to analyze the robustness and fault-tolerance of the system under study.

2.4 Research questions

In this research, we aim to address the following research questions (RQs) related to MQTT2EdgePeer, which facilitates a distributed peer-to-peer deployment of edge-located nodes:

- RQ1: Does MQTT2EdgePeer efficiently distribute heavy loads of hot topics?
- RQ2: Does MQTT2EdgePeer handle load dynamicity?
- RQ3: Does MQTT2EdgePeer support message distribution fault-tolerance?
- RQ4: Does MQTT2EdgePeer reduce the reconfiguration cost compared to the considered base DSRT approach?
- RQ5: Does MQTT2EdgePeer provide a scalable and robust P2P overlay for the MQTT protocol?

To answer these research questions, we perform experiments and compare our system with Scribe as a DSRT baseline approach.

2.5 System Architecture

In this section, we describe the system architecture of MQTT2EdgePeer. Figure 2.1 depicts the ring formation of nodes over the structured P2P overlay, the system components, and the communication types. The system supports two types of communication: inter-overlay (represented by red dashed lines) and MQTT (represented by blue dashed lines). All messages inside the structured P2P overlay network are sent through the inter-overlay message type, while all outer-overlay messages are sent by the MQTT message type. Clients, which can be one or many, are able to communicate with any of the nodes inside the overlay by using the standard MQTT client.

MQTT2EdgePeer consists of four major components: pub/sub peer, local MQTT broker, MQTT logger, and MQTT log collector. A description of each component is provided in the following subsections.

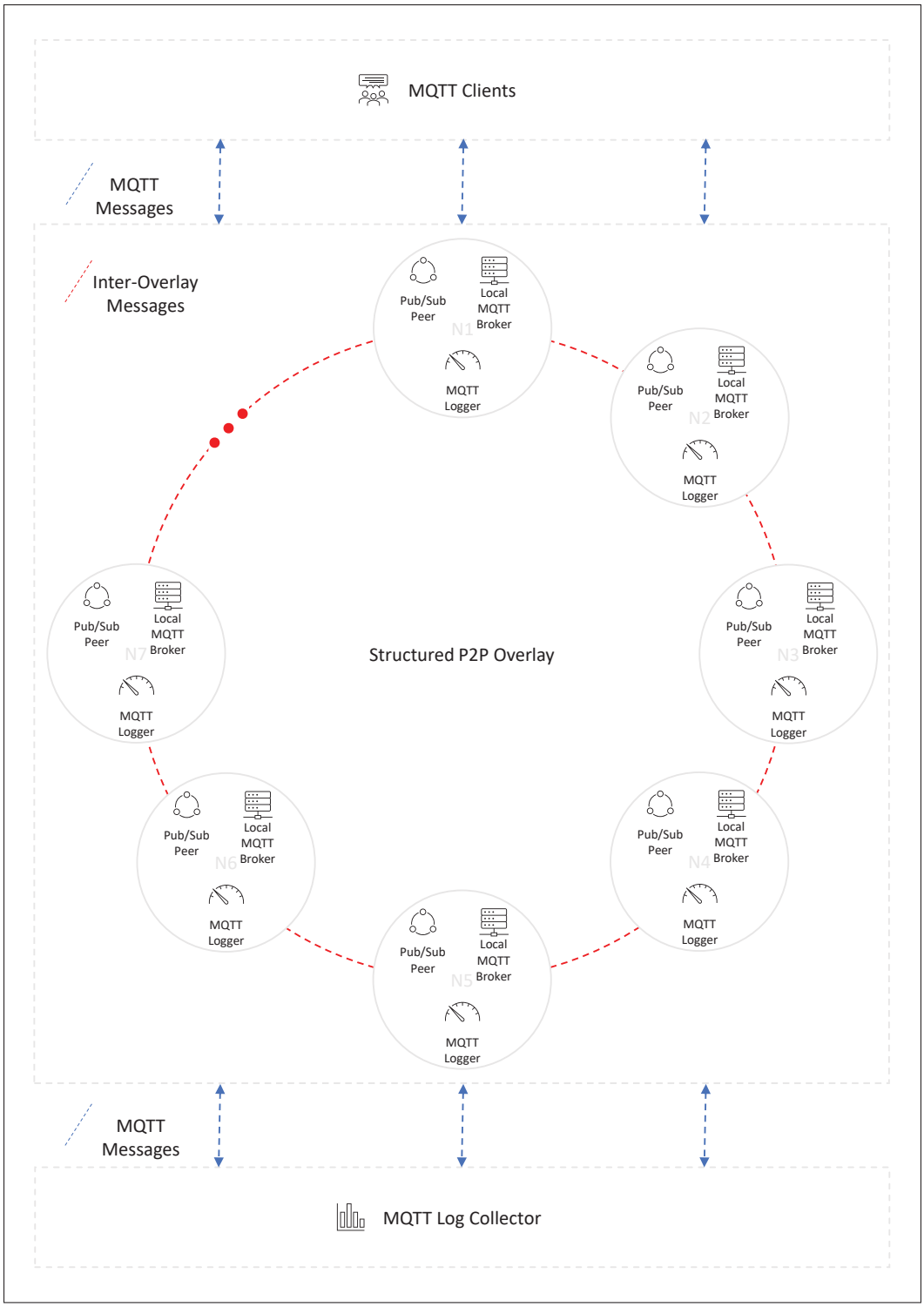


Figure 2.1 MQTT2EdgePeer system architecture

2.5.1 Pub/Sub peer

This component is the system's main component responsible for building the pub/sub peer over the structured P2P overlay. The pub/sub peer communicates with other peers in the overlay through inter-overlay messages. The pub/sub peer has four available roles: publisher, subscriber, coordinator, and shadow. Each peer is able to take the role of publisher or subscriber; however, only one node is assigned as the coordinator per topic with a specific number of shadow nodes. The coordinator maintains the list of subscribers and publishers of its correspondent topic, and the shadow is the coordinator's replica.

The coordinator handles membership changes and enables Direct Dispatching of publication messages to a topic from publishers to all correspondent subscribers. Unlike single-root-tree approaches that include both interested and non-interested nodes in the tree and dissemination paths, our coordinator-based approach ensures the exclusion of non-interested nodes. This is achieved by enabling publisher nodes to directly disseminate messages to subscriber nodes. Furthermore, all information in a coordinator is replicated on a separate node, called shadow. The shadow node assures the publishers always receive the correct list of subscribers. The replication and synchronization mechanism between coordinator and its shadows leads to improvement of the system fault-tolerance in case of failure in the coordinator node.

2.5.2 Local MQTT broker

The local MQTT broker component handles external MQTT messages between MQTT clients and each node in the overlay network. It relays incoming MQTT messages from clients to inter-overlay messages by transforming them into inter-overlay messages, and performs the same function for outgoing messages from the overlay to MQTT clients. Clients are able to connect to any node in the overlay to subscribe, unsubscribe, or publish messages for a topic t. The overlay disseminates clients' messages for topic t to relevant peers. Consequently, all local MQTT brokers in the overlay transparently cooperate with each other for routing messages.

2.5.3 MQTT Logger

This component logs relevant metrics of the node, such as IP, port, memory usage, CPU usage, network bandwidth usage, number of subscribers, number of messages, etc. It then publishes the collected logs periodically to the MQTT log collector component through MQTT messages.

2.5.4 MQTT Log Collector

This component subscribes to predefined topics on each node in the overlay. It collects logs from all nodes in the overlay and computes various performance metrics, such as the total number of messages, the total number of subscribers, and the average usage of resources. These metrics are used to analyze the performance of the overlay.

2.6 MQTT2EdgePeer structured P2P overlay network utilization

The design of MQTT2EdgePeer is based on utilizing the capabilities of Pastry structured peer-to-peer (P2P) overlay network and its associated Application Programming Interfaces (APIs) (Rowstron & Druschel, 2001). Although the system is built on top of Pastry, the overlay network can be replaced with any other middleware that supports routing and self-organizing, along with relevant APIs to access these features. Further information about the implementation and APIs can be found in the chapter 3. In this section, we outline the processes involved in node formation within the overlay network, the routing of messages, and the selection of coordinators and shadows.

2.6.1 Formation of nodes and routing messages

The structured overlay network is a fully decentralized and self-organized system that organizes nodes in a circular nodeId space, also known as the ring. The placement of nodes within the ring is determined by a unique identifier (nodeId) that is assigned to each node. These nodeIds can either be generated randomly or by hashing the node's IP address, with the assumption that they are uniformly distributed within the nodeId space.

To facilitate auto-discovery of other nodes within the network, each node keeps track of its immediate neighbors in the nodeId space. Furthermore, new nodes can join the overlay by contacting a bootstrap node. The overlay uses the nodeId space and uniform distribution of nodeIds to route messages. Each message has a destination address key that is used to identify the recipient node. The message is then routed to the node with the numerically closest nodeId to the message key within the nodeId space.

2.6.2 Coordinator and Shadow selection

MQTT2EdgePeer employs the structured overlay network's capabilities to identify the coordinator and its shadows for a specified topic. The system initially computes the hash address of the topic. Subsequently, it identifies the node with the closest numerically nodeId to the hash address as the coordinator, and the subsequent S (number of shadows) nodes are selected as the coordinator's shadows.

2.7 MQTT2EdgePeer communications

In this section, we provide an example in Figure 2.2 to clarify the communications of MQTT2EdgePeer. First, MQTT client one sends a subscription message (1) for topic *t* to node six using MQTT. Next, node six routes the subscription message (2, 3) to the candidate coordinator of topic *t* at node one and its shadow at node two using inter-overlay messages. Then, MQTT client two publishes a message (4) for topic *t* to node eight using MQTT. After that, node eight routes publish request messages (5, 6) to the coordinator and its shadow for topic *t* and gets the subscriber list from them as reply messages (7, 8). Finally, node eight routes the published message (9) to node six, and node six sends the published message (10) for topic *t* to MQTT client one.

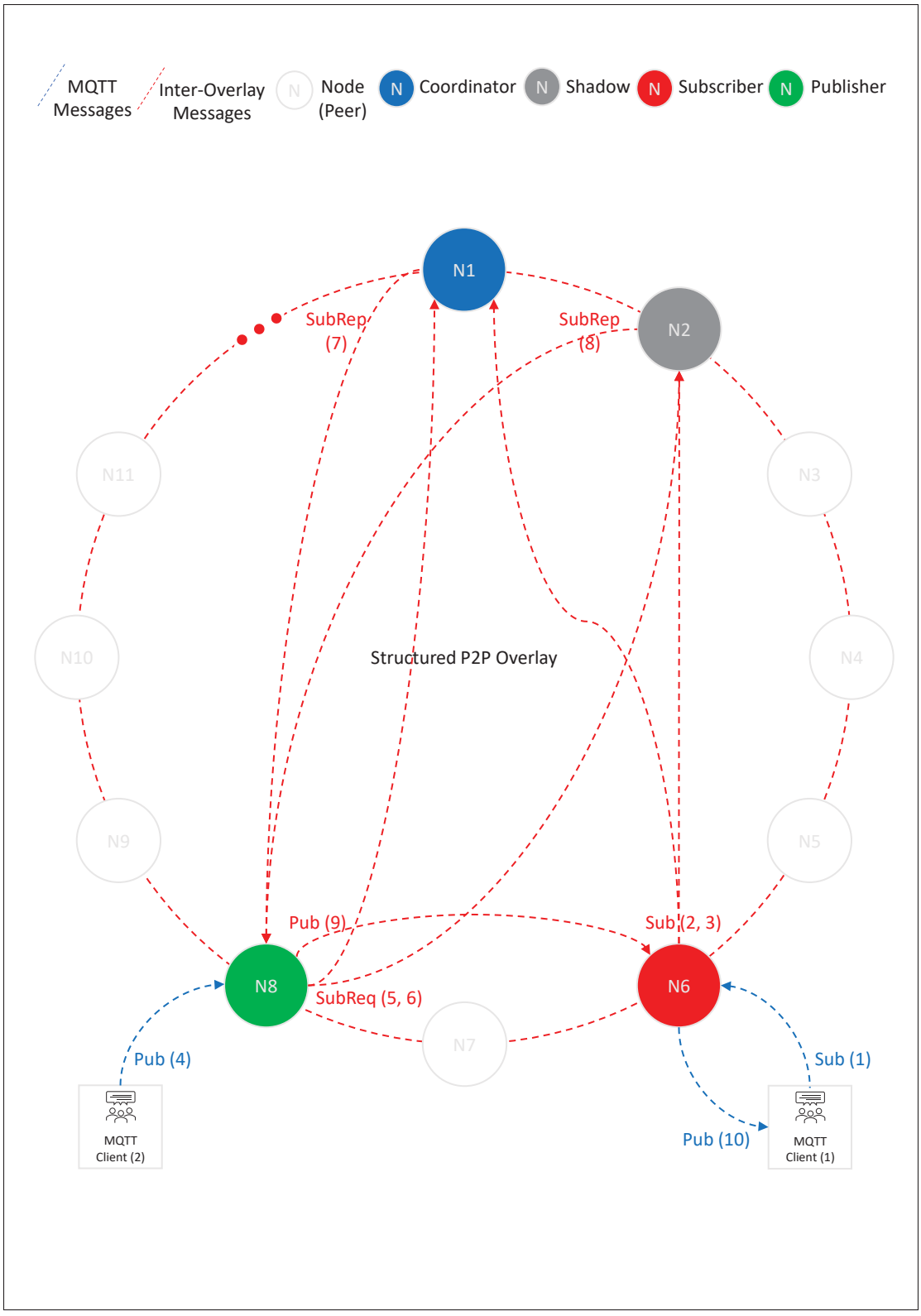


Figure 2.2 MQTT2EdgePeer communications example

2.8 Modeling and Parameters

In this section, we define the inter-overlay message types, their usage for coordinating message dissemination, and the constraints to consider when modeling the system.

2.8.1 System Coordination

Eleven types of inter-overlay messages are defined to coordinate message dissemination within the overlay network. These message types are presented in Table 2.1, all messages are routed within the overlay network through specific topics, as described below.

- **message$_{\mathbf{Sub}}$**: A subscriber sends this message to the related coordinator and shadows. The coordinator and shadows then add the subscriber information and topic to their respective lists.
- **message$_{\mathbf{Pub}}$**: A publisher sends this message to all its subscribers.
- **message$_{\mathbf{UnSub}}$**: A subscriber sends this message to the related coordinator and shadows. The coordinator and shadows then remove the subscriber information and topic from their respective lists.
- **message$_{\mathbf{SubReq}}$**: A publisher sends this message to the related coordinator and shadows. The coordinator and shadows then add the publisher information and topic to their respective lists.
- **message$_{\mathbf{SubRep}}$**: A coordinator and shadows send this message to the requesting publisher. The publisher then adds the subscriber list and topic to their respective lists.
- **message$_{\mathbf{NewSub}}$**: A coordinator sends this message to all related publishers who do not have the new subscriber in their subscriber list. Publishers then add the new subscriber to their respective lists.
- **message$_{\mathbf{NewUnSub}}$**: A coordinator sends this message to all related publishers who have the subscriber in their subscriber list. Publishers then remove the subscriber from their respective lists.
- **message$_{\mathbf{UpSub}}$**: A shadow sends this message to the related coordinator. The coordinator then updates its respective list.

- **message$_{\mathbf{ReSub}}$**: A coordinator sends this message to all its publishers. Publishers then reset their respective lists.
- **message$_{\mathbf{PubCW}}$**: A publisher/subscriber sends this message to the next subscribers in the clockwise direction.
- **message$_{\mathbf{PubCCW}}$**: A publisher/subscriber sends this message to the next subscribers in the counterclockwise direction.

Table 2.1 Inter-overlay Messages

Notation	Description
$message_{Sub}$	Inter-overlay subscribe message
$message_{Pub}$	Inter-overlay publish message
$message_{UnSub}$	Inter-overlay unsubscribe message
$message_{SubReq}$	Inter-overlay subscriber list request message
$message_{SubRep}$	Inter-overlay subscribe list reply message
$message_{NewSub}$	Inter-overlay new subscribe notice message
$message_{NewUnSub}$	Inter-overlay new unsubscribe notice message
$message_{UpSub}$	Inter-overlay updated subscribe list message
$message_{ReSub}$	Inter-overlay reset subscribe list message
$message_{PubCW}$	Inter-overlay clockwise publish message
$message_{PubCCW}$	Inter-overlay counterclockwise publish message

2.8.2 Constraints

We consider two critical constraints for our edge/IoT system: latency and bandwidth. Latency refers to the delay in transmitting data between devices, and it is a crucial factor in determining the speed and responsiveness of the system. Bandwidth, on the other hand, refers to the amount of data that can be transmitted over a network within a given time, and it is vital in ensuring the efficient operation of the system. These constraints are particularly crucial for our edge/IoT system, as it involves devices that require real-time data processing and transmission.

In order to examine the constraints of our system, we exclude any considerations regarding CPU and memory constraints due to the availability of adequate computational resources to accommodate our solution. In addition, topic-based pub/sub systems demand lower computational

and storage resources in comparison to content-based pub/sub systems, as the matching process involves a simple hash map look-up. This strategy ensures the correct operation of our system, eliminating computational bottlenecks that may impede its performance.

2.8.3 Latency Calculation

In our system, we track the latency of inter-overlay publication messages for evaluation purposes. Specifically, we attach a creation time to each message at the time of its creation, which enables us to calculate the latency when the message is received by the destination node. This approach allows us to monitor the efficiency and reliability of the message delivery process in our system.

To determine the average message latency of our system, we collect data on the latency of inter-overlay publication messages at each node. We then aggregate this data to calculate the average latency across the entire system. This calculation is based on the total number of publication messages at each node and the latency of each message.

2.8.4 Bandwidth Calculation

To accurately measure the bandwidth usage of our system, we track the incoming and outgoing traffic at each node during message dissemination for evaluation purposes. This tracking allows us to calculate the amount of data transmitted by each node, which we then aggregate to obtain an overall estimate of bandwidth usage.

2.9 MQTT2EdgePeer compatibility with standard MQTT protocol

MQTT2EdgePeer is a system that provides support for the three core message types in the MQTT standard protocol, including subscribe, unsubscribe, and publish. Unlike traditional distributed MQTT-based applications, the MQTT2EdgePeer system enables clients to connect to any nodes within the overlay network using a standard MQTT client implementation, without requiring prior knowledge of the correspondent MQTT brokers in the network.

The MQTT2EdgePeer system functions by taking control of the discovery of brokers, routing of messages between brokers, and responding to MQTT clients. Specifically, the system employs a decentralized approach in which brokers and clients interact in a peer-to-peer fashion, without the need for a centralized broker or message broker registry.

This decentralized approach enables the MQTT2EdgePeer system to operate in a more efficient and flexible manner compared to traditional MQTT-based systems, by providing clients with a greater degree of autonomy and control over their interactions with the network. Furthermore, the use of standard MQTT client implementation allows for easy integration with existing systems, thereby minimizing the need for additional development and integration efforts.

2.10 MQTT2EdgePeer coordinator failure resiliency

MQTT2EdgePeer employs shadow nodes as a mechanism for enhancing the fault tolerance of the system, in the event of coordinator node failures. A shadow node functions by maintaining a complete copy of all data from its corresponding coordinator. To synchronize shadows with the coordinator, whenever subscriber nodes or publisher nodes send membership change messages or subscriber request list messages, respectively, to the corresponding coordinator and all its shadows. The shadows subsequently send subscriber list update messages to the coordinator, which then the coordinator combines its list with the received updates. Moreover, shadows provide extra sources for publisher nodes to request and receive subscriber lists in addition to the coordinator.

In the event of coordinator failure, MQTT2EdgePeer utilizes coordinator and shadow selection technique 2.6.2. The first shadow is selected as the new coordinator, and the subsequent S nodes (where S denotes the number of shadows) following the original coordinator remain or are chosen as new shadows. The synchronization between the coordinator and its shadows, the provision of subscriber lists from multiple sources, and the aforementioned coordinator and shadows selection technique work in concert to enhance the overall fault-tolerance of MQTT2EdgePeer, particularly in instances where coordinator nodes experience failure.

2.11 Algorithms

This section provides full descriptions and examples of the proposed algorithms in this research.

2.11.1 Direct Dispatching algorithm

As shown in table 2.2 and algorithms 2.1 and 2.2, the Direct Dispatching algorithm takes as input the current node's Id, topic, publication, subscriber list expiry interval, and shadow count, and then either sends the inter-overlay publication message to its subscribers (2.11.1.1) or sends the inter-overlay subscriber list request message to the coordinator of the topic and its shadows (2.11.1.2).

2.11.1.1 Sending publication

To initialize the algorithm 2.1 parameters, the subscriber list is set to the current node's subscriber list for topic *t* (line 2). The next step is to apply a condition that checks the value of the subscriber list for topic *t* and the validity of the subscriber list expiry interval (line 3). The subscriber list expiry interval parameter ensures that the publisher always routes inter-overlay publication messages to a valid and up-to-date list of subscribers, even if there are node failures or subscription changes. If the condition is met, the publisher sends the inter-overlay publication message to every subscriber on the list (lines 4-6).

2.11.1.2 Requesting subscribers

The algorithm 2.2 presents the alternative for algorithm 2.1 in case of having invalid subscriber list. First, the hash is calculated for topic *t*, the coordinator is determined as the closest node to the hash of topic *t* as we described in the section 2.6, and the shadow list for topic *t* is initialized with a new list (lines 2-4). The algorithm then determines the shadows in a loop by selecting the next closest nodes to the hash of topic *t* after the coordinator (lines 5-7). Finally, the publisher sends inter-overlay subscriber list request messages to the coordinator of topic *t* and its shadow (lines 8-13). This allows the publisher to retrieve the list from multiple sources, which improves

the reliability of the algorithm. To enable the retry process for sending publications in algorithm 2.1, the publication is attached to the request messages. Additionally, it will be received from the coordinator and shadows in reply messages.

Table 2.2 Parameters of algorithms 2.1 and 2.2

Notation	**Description**
node	Current node Id
topic	Topic t
publication	Publication message content
$subscribers_t$	List of all subscribers to topic t
$hash_t$	Hash of topic t
$coordinator_t$	Coordinator of topic t
$shadows_t$	List of shadows of topic t
shadowsNum	Number of shadows
subExpInt	Expiration interval for subscriber list

Algorithm 2.1 Direct Dispatching (sending publication)

Input: node, topic, publication, subExpInt
Output: route $message_{Pub}$ to $subscribers_t$

```
1 Begin
2     subscribers_t ← getSubscribers(node, topic);
3     if subscribers_t > 0 and checkSubLastUpdate(node, subExpInt) then
4         for subscriber ∈ subscribers do
5             message_Pub ← generatePubMessage(node, topic, publication);
              route(message_Pub, subscriber);
6         end for
7     end if
8 End
```

2.11.1.3 Direct Dispatching example

To better understand algorithm 2.1, we provide an example in Figure 2.3. The example demonstrates the ring formation of 12 nodes in a structured P2P overlay network. For simplicity, we illustrate only the routing of inter-overlay publication messages from node 1, which acts as the publisher for topic t, to its subscribers located at nodes 3, 5, 6, 9, 11, and 12 ($Pub(1)$). In this

Algorithm 2.2 Direct Dispatching (requesting subscribers)

Input: node, topic, publication, subExpInt, shadowsNum
Output: route $message_{SubReq}$ to $coordinator_t$ and $shdows_t$

Begin
  $hash_t \leftarrow getHash(topic)$;
  $coodinator_t \leftarrow getClosestNode(hash_t, 1)$;
  $shadows_t \leftarrow []$;
  for $w \in \{1, ..., shadowsNum\}$ **do**
    $shadow_t.insert(getClosestNode(hash_t, w + 1))$;
  end for
  $message_{SubReq} \leftarrow$
   $generateSubReqMessage(node, topic, publication, coordinator_t)$;
  $route(message_{SubReq}, coordinator_t)$;
  for $shadow_t \in shadows_t$ **do**
    $message_{SubReq} \leftarrow$
     $generateSubReqMessage(node, topic, publication, shadow_t)$;
    $route(message_{SubReq}, shadow_t)$;
  end for
End

example, we assume that publisher node 1 has a valid list of subscribers and only routes publish messages. In case of a lack of a valid subscriber list, the routing of inter-overlay subscriber list request messages follows a similar pattern to the communication example shown in Figure 2.2 in steps 5-6. Furthermore, the following steps in Figure 2.2 can also similarly be applied in this example:

- Sending the MQTT subscription message in steps 1-3.
- Replying subscriber list in steps 7-8.
- Sending the MQTT publication message in steps 4 and 10.

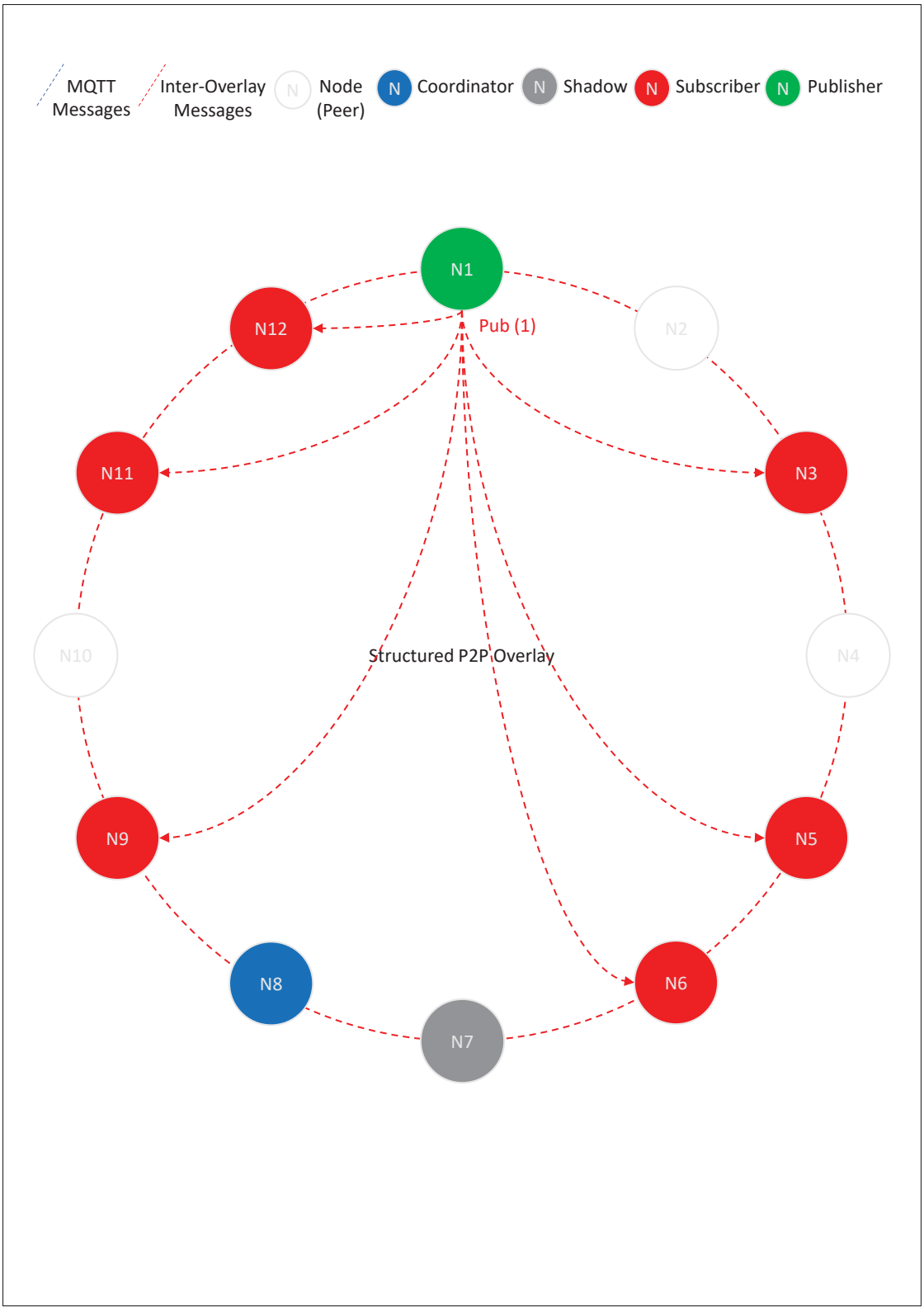


Figure 2.3 Direct Dispatching example

2.11.2 Guided Dispatching algorithm

The Guided Dispatching algorithm enhances the efficient dissemination of inter-overlay publication messages in structured P2P overlay networks. The algorithm utilizes subscriber nodes in the routing paths, both clockwise and counterclockwise, to achieve multi-hop routing. This approach effectively distributes the load among a larger number of nodes, resulting in improved load distribution. By involving subscriber nodes and striking a balance between bandwidth usage and latency, the Guided Dispatching approach can potentially reduce resource usage, such as bandwidth in publisher and subscriber nodes. This is particularly beneficial when dealing with a large number of subscribers. Therefore, Guided Dispatching significantly improves the load distribution of publish messages compared to the single-hop Direct Dispatching approach.

The Guided Dispatching algorithm is presented in distinct parts for publisher (2.11.2.1, 2.11.2.2, and 2.11.2.3) and subscriber nodes (2.11.2.4). Table 2.3 displays the relevant parameters for each part of the algorithm.

2.11.2.1 Publisher node part

Algorithm 2.3 and 2.4 outlines the steps involved in the Guided Dispatching algorithm, specifically the publisher node part. The algorithm takes node, topic, publication, subscriber list expiry interval, and shadow count as input parameters, and uses these to route the inter-overlay publication message to its subscribers in both clockwise and counterclockwise directions, or to route the inter-overlay subscriber list request message to the corresponding coordinator of the topic and its shadows.

2.11.2.2 Sending publication

The algorithm initializes the parameters by setting the subscriber list with the current node subscriber list of topic t (line 2).

Once the initialization of the Guided Dispatching algorithm is complete, the next step involves evaluating the subscriber list of topic *t* and the validity of the subscriber list expiry interval by applying a condition (line 3). Similar to Direct Dispatching algorithm, the subscriber list expiry interval parameter serves to ensure that the inter-overlay publication messages are always routed to a valid and up-to-date list of subscribers, even in the event of node failures or changes to the subscription list. This helps to maintain the reliability and effectiveness of the network communication by minimizing the chances of message delivery failure due to outdated subscriber lists or expired subscription intervals. The condition thus plays a critical role in enhancing the overall performance and resilience of the Guided Dispatching algorithm in overlay networks.

Assuming the condition evaluating the subscriber list of topic *t* and the validity of the subscriber list expiry interval is satisfied (line 3), the Guided Dispatching algorithm can proceed to disseminate multiple inter-overlay publish messages in both clockwise and counterclockwise directions. This is achieved by determining the node index of each subscriber and their relative position compared to the current node in terms of clockwise or counterclockwise (lines 4-13). The algorithm then routes inter-overlay publication messages to subscribers located in both directions as long as the number of outgoing messages does not exceed the fan-out parameter.

For the clockwise direction, the algorithm iterates over the clockwise subscriber list (line 14), then retrieves and updates the number of subscribers to whom the publication message can be sent before exceeding the fan-out value (line 15). Next, it checks whether the node has exceeded the fan-out limit. If the limit has not been exceeded, the algorithm proceeds to obtain and update the next subscriber nodes in the routing path to which the next node should send the publication message (line 16). Finally, the algorithm generates and routes the publication message (lines 17-19), or alternatively, it stops the iteration if the fan-out limit has been exceeded (line 21).

For the counterclockwise direction, the same process as the aforementioned process for the clockwise direction is applied (lines 24-33).

2.11.2.3 Requesting subscribers

Algorithm 2.4 provides an alternative condition to algorithm 2.3 when encountering an invalid subscriber list. The entire process of requesting the subscriber list (lines 2-13) is similar to algorithm 2.2, as described in the subsection 2.11.1.2.

2.11.2.4 Subscriber node part

The subscriber node part of the Guided Dispatching algorithm is presented in algorithm 2.5. This algorithm takes node, topic, publication, fan-out, subCW, and subCCW as inputs, and is responsible for routing inter-overlay publication messages to the next subscribers of topic *t* in either clockwise or counterclockwise directions.

In the clockwise direction, the algorithm performs an iteration over the clockwise subscriber list (line 2), retrieving and updating the number of subscribers to whom the publication message can be sent without exceeding the predetermined fanout value (line 3). Subsequently, it verifies whether the fanout limit has been surpassed by the node (line 4). If the limit remains within bounds, the algorithm proceeds to acquire and update the subsequent subscriber nodes along the routing path, to which the next node must transmit the publication message (line 5). Ultimately, the algorithm generates and routes the publication message (lines 6-7). Alternatively, if the fanout limit has been exceeded, the iteration ceases (line 9).

Similarly, for the counterclockwise direction, the algorithm applies the same aforementioned process employed for the clockwise direction (lines 12-21).

Table 2.3 Parameters of algorithms 2.3, 2.4, and 2.5

Notation	**Description**
node	Current node Id
topic	Topic t
publication	Publication message content
$subscribers_t$	List of all subscribers to topic t
$hash_t$	Hash of topic t
$coordinator_t$	Coordinator of topic t
$shadows_t$	List of shadows of topic t
shadowsNum	Number of shadows
subExpInt	Expiration interval for subscriber list
fan-out	Number of outgoing publication messages per node
$index_{sub}$	Index of subscriber
subCW	Map of subscribers to topic t in clockwise order with node index
subCCW	Map of subscribers to topic t in counterclockwise order with node index
sendCW	List of clockwise send-to-subscribers
sendCCW	List of counterclockwise send-to-subscribers

Algorithm 2.3 Guided Dispatching (publisher node part - sending publication)

Input: node, topic, publication, subExpInt, fanout
Output: route $message_{PubCW}$ and $message_{PubCCW}$ to $subscribers_t$

```
1  Begin
2      subscribers_t ← getSubscribers(node, topic);
3      if subscribers_t > 0 and checkSubLastUpdate(node, subExpInt) then
4          subCW, subCCW ← {};
5          sendCW, sendCCW ← [];
6          for subscriber ∈ subscribers_t do
7              index_sub ← getNodeIndex(subscriber);
8              if index_sub > 0 then
9                  subCW.insert(subscriber, index_sub);
10             else
11                 subCCW.insert(subscriber, index_sub);
12             end if
13         end for
14         for sub_CW ∈ subCW do
15             sendCW ← getUpdateSendCW(sub_CW, subCW, fanout);
16             if (checkFanoutCW(sendCW, subCCW, fanout)) then
17                 nextCW ← getUpdateNextCW(sub_CW, sendCW, fanout);
18                 message_PubCW ← generatePubCWMessage(node, topic,
                     publication, coordinator_t, nextCW);
19                 route(message_PubCW, sub_CW);
20             else
21                 Break;
22             end if
23         end for
24         for sub_CCW ∈ sendCCW do
25             sendCCW ← getUpdateSendCCW(sub_CCW, subCCW, fanout);
26             if (checkFanoutCCW(sendCCW, subCW, fanout)) then
27                 nextCCW ← getUpdateNextCCW(sub_CCW, sendCCW, fanout);
28                 message_PubCCW ← generatePubCCWMessage(node, topic,
                     publication, coordinator_t, nextCCW);
29                 route(message_PubCCW, sub_CCW);
30             else
31                 Break;
32             end if
33         end for
34     end if
35 End
```

Algorithm 2.4 Guided Dispatching (publisher node part - requesting subscribers)

Input: node, topic, publication, subExpInt, shadowsNum
Output: route $message_{SubReq}$ to $coordinator_t$ and $shdows_t$

1 **Begin**
2 $hash_t \leftarrow getHash(topic)$;
3 $coodinator_t \leftarrow getClosestNode(hash_t, 1)$;
4 $shadows_t \leftarrow []$;
5 **for** $w \in \{1, ..., shadowsNum\}$ **do**
6 $shadow_t.insert(getClosestNode(hash_t, w + 1))$;
7 **end for**
8 $message_{SubReq} \leftarrow$
$generateSubReqMessage(node, topic, publication, coordinator_t)$;
9 $route(message_{SubReq}, coordinator_t)$;
10 **for** $shadow_t \in shadows_t$ **do**
11 $message_{SubReq} \leftarrow$
$generateSubReqMessage(node, topic, publication, shadow_t)$;
12 $route(message_{SubReq}, shadow_t)$;
13 **end for**
14 **End**

Algorithm 2.5 Guided Dispatching (subscriber node part)

Input: node, topic, publication, fanout, subCW, subCCW
Output: route $message_{PubCW}$ to sub_{CW}, or route $message_{PubCCW}$ to sub_{CCW}

```
1  Begin
2      for sub_CW ∈ subCW do
3          sendCW ← getUpdateSendCW(sub_cw, subCW, fanout);
4          if (checkFanoutCW(sendCW, fanout)) then
5              nextCW ← getUpdateNextCW(sub_CW, sendCW, fanout);
6              message_PubCW ← generatePubCWMessage(node, topic,
                 publication, coordinator_t, nextCW);
7              route(message_PubCW, sub_CW);
8          else
9              Break;
10         end if
11     end for
12     for sub_CCW ∈ subCCW do
13         sendCCW ← getSendCCW(sub_CCW, subCCW, fanout);
14         if (checkFanoutCCW(sendCCW, fanout)) then
15             nextCCW ← getNextCCW(sub_CCW, sendCCW, fanout);
16             message_PubCCW ← generatePubCCWMessage(node, topic,
                 publication, coordinator_t, nextCCW);
17             route(message_PubCCW, sub_CCW);
18         else
19             Break;
20         end if
21     end for
22 End
```

2.11.2.5 Guided Dispatching example

To provide a better understanding of the Guided Dispatching algorithm, we present an example in Figure 2.4, where a structured P2P overlay network consists of 12 nodes arranged in a ring formation, and the fan-out parameter is set to 2. In the first round, node 1 as the publisher node of the algorithm routes two inter-overlay publication messages to subscribers of topic t in the clockwise direction to node 3 ($Pub_{CW}(1)$) and in the counterclockwise direction to node 2 ($Pub_{CCW}(1)$), respecting the fan-out value. In the second round, node 3, the first subscriber node in the clockwise direction, then routes the inter-overlay publication message to the next two subscriber nodes, 5 and 6 ($Pub_{CW}(2)$), in the same direction, following the fan-out value. Similarly, node 12, the first subscriber node in the counterclockwise direction, routes the inter-overlay publication message to nodes 11 and 9 ($Pub_{CCW}(2)$). The process continues until all subscribers of topic t receive the publication message.

In this example, we assume that publisher node 1 has a valid list of subscribers and only routes publish messages. In case of a lack of a valid subscriber list, the routing of inter-overlay subscriber list request messages follows a similar pattern to the communication example shown in Figure 2.2 in steps 5-6.

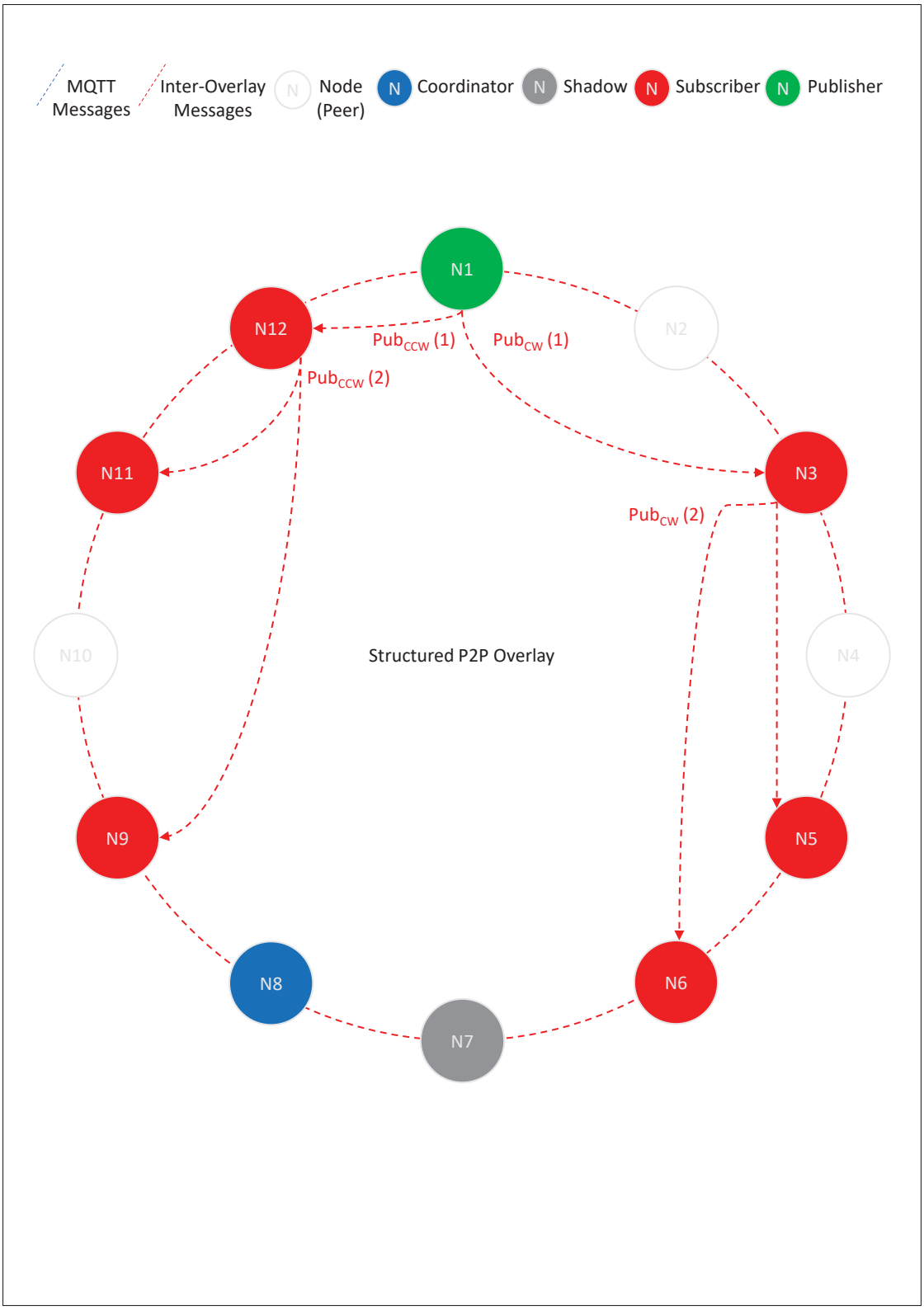


Figure 2.4 Guided Dispatching example with fan-out=2

CHAPTER 3

IMPLEMENTATION

This chapter is dedicated to the development of MQTT2EdgePeer, a system designed for efficient and reliable communication at the edge. The chapter is divided into four sections. First, the various programming languages and libraries used during the development process are discussed. Second, the development tools used to increase productivity and efficiency during system development are described. The third section outlines the steps involved in configuring, setting up, and running MQTT2EdgePeer. Finally, the chapter concludes with a discussion on the logging mechanisms implemented in the system to facilitate easy monitoring.

3.1 Programming languages and libraries

In the development of MQTT2EdgePeer, we utilized several programming languages and libraries, which are described in this section.

3.1.1 JAVA

We developed the proposed algorithms in the previous chapter and implemented most parts of the MQTT2EdgePeer system using the Java programming language. Java is widely used for object-oriented programming and benefits from cross-platform support (i.e., Windows, UNIX, Mac), automatic memory management, multi-threading, etc. There are several versions of the Java Development Kit (JDK) available for developing Java applications. In this thesis, we used OpenJDK version 1.8 for development.

3.1.2 Python

We used Python to develop the MQTT log collector component of the MQTT2EdgePeer system. We chose to use Python version 3.8 due to its fast runtime and ease of development, which facilitated the component's development.

3.1.3 Bash

Bash is a Unix shell and command language that provides a convenient interface for executing a sequence of commands. We employed Bash to automate the deployment, running, and testing the system.

3.1.4 FreePastry

FreePastry (Rice, 2009) is an open-source implementation of the Pastry (Rowstron & Druschel, 2001) structured peer-to-peer overlay network, written in the Java programming language. It provides a self-organizing and efficient routing overlay network of Pastry nodes. In the FreePastry overlay, each Pastry node has a unique identifier and receives messages that have the closest destination Id to its node Id. While we developed MQTT2EdgePeer on top of the FreePastry version 2.1 structured peer-to-peer overlay network, it can be implemented on any structured P2P overlay network that provides the same capabilities and APIs.

3.1.4.1 Primary APIs

We describe the three main APIs of the structured overlay network utilized by MQTT2EdgePeer as follows:

- **route(msg, key):** This API is utilized to route a message to the node with the closest nodeId to the given key.
- **deliver(msg, key):** This API is invoked when a message is delivered to the node with the closest nodeId to the key.
- **forward(msg, key,nextId):** This API is invoked before forwarding a message to a node with a nodeId that matches nextId.

3.1.5 Moquette

We utilized Moquette (Moquette, 2023) to develop the local MQTT broker component of the MQTT2EdgePeer system. Moquette is a lightweight, server-side implementation of the MQTT

protocol, developed using the Java programming language. We chose to use Moquette due to its lightweight nature and its implementation in Java, which made it a suitable choice for our system.

3.1.6 Aiomqtt

We utilized Aiomqtt (Aiomqtt, 2023), which is an asynchronous wrapper for the Paho Python MQTT client library (Eclipse, 2023), to establish MQTT client connections between the MQTT log collector component and the local MQTT broker component of the system.

3.1.7 eMQTT-Bench

We utilized eMQTT-Bench (eMQTT Bench, 2023), a lightweight and efficient MQTT benchmarking tool, to generate numerous MQTT clients and transmit/receive MQTT messages. This tool proved to be fast and reliable for testing MQTT systems.

3.2 Development tools

To increase productivity and efficiency during system development, we utilized various tools to set up our development environment.

3.2.1 Intellij IDEA

IntelliJ IDEA is an integrated development environment that supports several programming languages, including Java and Python. We chose to use IntelliJ IDEA because it offers automatic code completion, which accelerates the development process and helps to write clean and concise code.

3.2.2 Gradle

We utilized Gradle as our build tool to automate building the system and generating output files for deployment. Gradle is a simple and efficient build tool for Java applications, which is configured to determine project dependencies, repositories, versions, and other build-related settings.

3.3 Configuration

This section outlines the steps involved in configuring, building, and running MQTT2EdgePeer.

3.3.1 Construct a Pastry node

The provided Java code in Figure 3.1 demonstrates how to create and join a new node to a Pastry peer-to-peer structured overlay network. The code consists of several steps:

Firstly, a node identifier is generated using the IPNodeIdFactory class and the specified values for the host IP address, host binding port, and environment.

Secondly, a SocketPastryNodeFactory object is created using the previously generated NodeId-Factory, along with the specified host port value and environment value.

Thirdly, a new PastryNode object is created using the PastryNodeFactory created in the previous step.

Finally, the newly created PastryNode is booted by calling the boot() method with the bootstrap node's bootAddress parameter, which joins the node to the Pastry overlay network and enables it to communicate with other nodes in the network.

```
1  // Generate node identifier
2  NodeIdFactory ipNodeIdFactory =
3  new IPNodeIdFactory(hostAddress, bindPort, env);
4  // Construct the PastryNodeFactory
5  PastryNodeFactory factory =
6  new SocketPastryNodeFactory(ipNodeIdFactory,
7  bindPort, env);
8  // Construct a new node
9  PastryNode node = factory.newNode();
10 // Join the ring
11 node.boot(bootAddress);
```

Figure 3.1 Set up a Pasty node

3.3.2 Build the MQTT2EdgePeer on a Pastry node

In Figure 3.2, a sample Java code is provided to illustrate the construction of MQTT2EdgePeer on top of a pastry node, with the process being carried out in several distinct steps. A description of each of these steps is presented below:

At first, the broker is created by instantiating a Server object. After that, a new PubSubPeer object is created with two parameters: Pastry node and Moquette mqttBroker. The PubSubPeer class represents a publish-subscribe peer, which is a component in the MQTT2EdgePeer that enables the communication between a publisher and a subscriber. Next, the broker port is configured by creating a Properties object and setting the property to the value of the broker port. A list of InterceptHandler objects is created using Collections.singletonList(). In this case, the list only contains the peer object created earlier. Finally, the MQTT broker is started, which takes the conf and userHandlers parameters. The conf parameter is the configuration for the broker, while the userHandlers parameter is a list of InterceptHandler objects that the broker will use to intercept and handle MQTT messages.

The diagram presented in Figure 3.3 illustrates the class structure of the PubSubPeer component, as well as the methods employed for managing inter-overlay and MQTT communications.

```
// Construct a Moquette MQTT broker
Server mqttBroker = new Server();
// Construct a new publish-subscribe peer
PubSubPeer peer = new PubSubPeer(node, mqttBroker);
// Configure the broker port
Properties props = new Properties();
props.put(BrokerConstants.PORT_PROPERTY_NAME,
String.valueOf(brokerPort));
IConfig conf = new MemoryConfig(props);
// Create a list of InterceptHandler objects
final List<? extends InterceptHandler>
userHandlers = Collections.singletonList(peer);
// Start the MQTT broker
mqttBroker.startServer(conf, userHandlers);
```

Figure 3.2 Build the MQTT2EdgePeer on a Pastry node

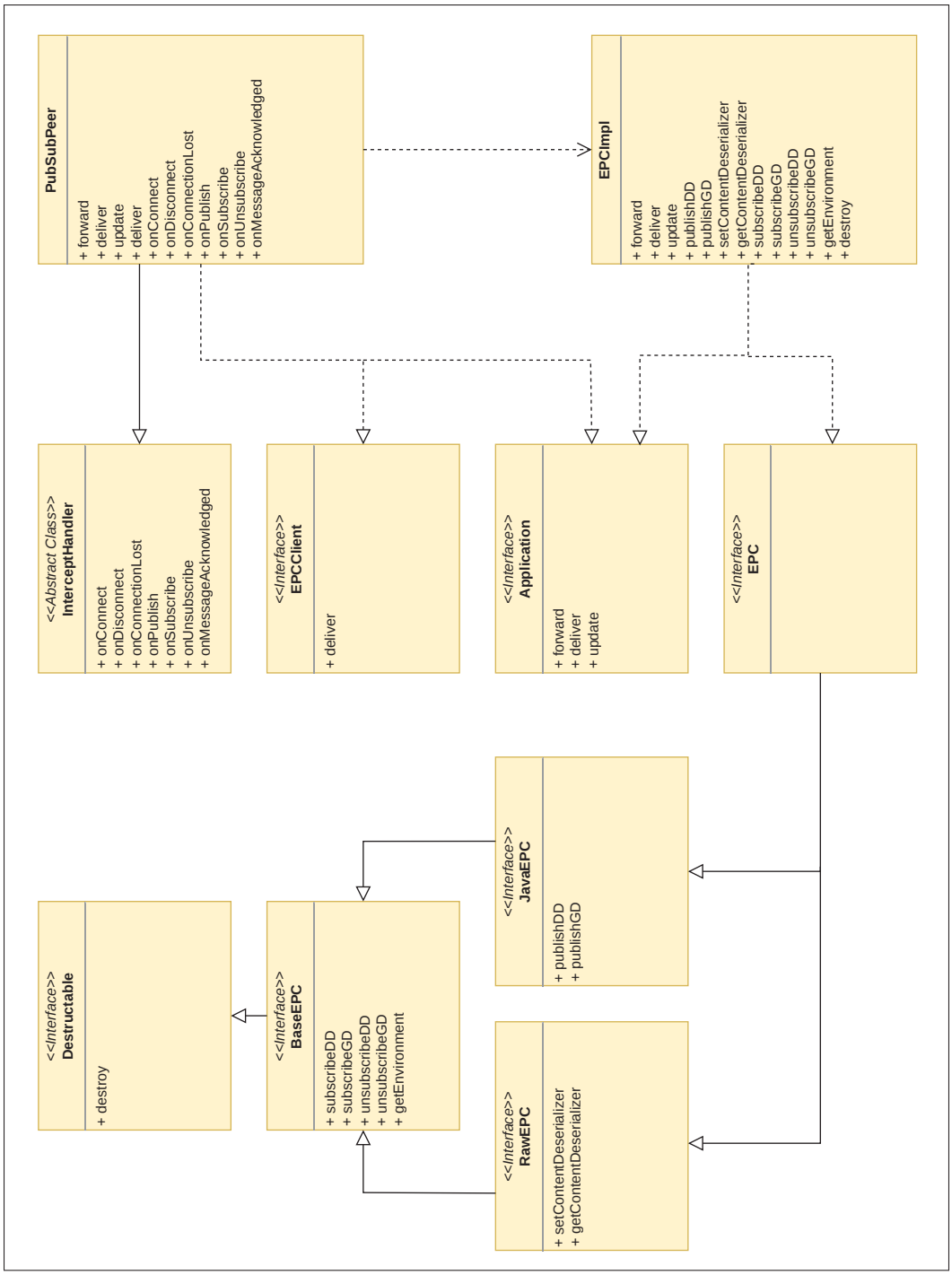


Figure 3.3 Diagram of classes

3.3.3 Run MQTT2EdgePeer

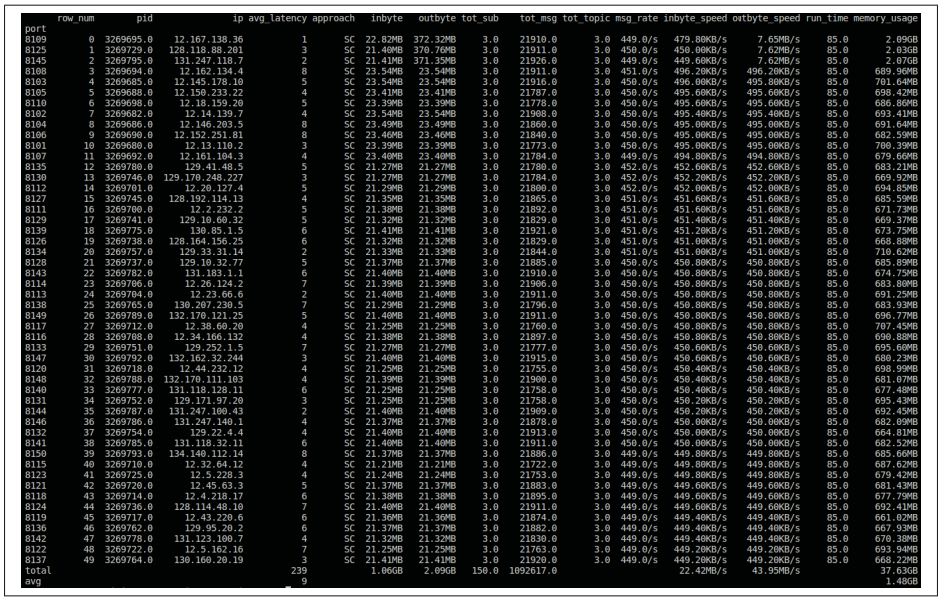
We developed several Bash scripts to simplify the deployment and execution of MQTT2EdgePeer on multiple Pastry nodes, due to the large scale of the project. The following are brief descriptions of these Bash files:

- **node-builder.sh**: The program takes several inputs, such as the number of Pastry nodes, the dispatching approach for MQTT2EdgePeer, the address of the bootstrap node, the port number of the node, and the port number of the first MQTT broker. It then starts MQTT2EdgePeer on the Pastry nodes with different process IDs on the target machine, using sequential port numbers.
- **node-builder-batch.sh**: The program takes several inputs, such as the number of Pastry nodes, the dispatching approach for MQTT2EdgePeer, the IP address of the bootstrap node, the port number of the first node, the port number of the first MQTT broker, and the number of batches. It then starts MQTT2EdgePeer on Pastry nodes with the same process ID for each batch on the target machine, using sequential port numbers. The number of nodes in each batch equals the number of nodes divided by the number of batches.
- **emqtt-sub.sh**: The program takes several inputs, such as the number of subscriber nodes, the host IP address, the client ID prefix, the port number of the first MQTT broker, the number of clients, the topic, and the number of topics. It then starts the Emqtt-bench to subscribe to each topic on each host and broker port number.
- **emqtt-pub.sh**: The program takes several inputs, such as the number of publisher nodes, the host IP address, the client ID prefix, the port number of the first MQTT broker, the number of clients, the topic, the number of topics, and the payload size. It then starts the Emqtt-bench to publish messages to each topic on each host and broker port number.

3.4 Logging

We developed the MQTT log collector using the Python programming language to gather logs from the MQTT logger component of MQTT2EdgePeer. The MQTT log collector takes inputs such as the host IP address, the port number of the first MQTT broker, the collecting time

interval, and the output file name and directory path. Then, it runs the collector by subscribing to MQTT2EdgePeer nodes through the logging topic. After that, the MQTT logger component publishes logs to the log collector at specific time intervals. Figure 3.4 displays sample results of the MQTT log collector.



port	row_num	pid	ip	avg_latency	approach	inbyte	outbyte	tot_sub	tot_msg	tot_topic	msg_rate	inbyte_speed	outbyte_speed	run_time	memory_usage
8109	0	3269695.0	12.167.138.36	1	SC	22.02MB	372.32MB	3.0	21910.0	3.0	449.0/s	479.80KB/s	7.65MB/s	85.0	2.09GB
8125	1	3269729.0	128.118.88.201	3	SC	21.40MB	370.76MB	3.0	21911.0	3.0	450.0/s	450.00KB/s	7.62MB/s	85.0	2.03GB
8145	2	3269795.0	131.247.118.7	2	SC	21.41MB	371.35MB	3.0	21926.0	3.0	449.0/s	449.60KB/s	7.62MB/s	85.0	2.07GB
8108	3	3269694.0	12.162.134.4	8	SC	23.54MB	23.54MB	3.0	21911.0	3.0	451.0/s	496.20KB/s	496.20KB/s	85.0	689.96MB
8103	4	3269685.0	12.145.178.10	5	SC	23.54MB	23.54MB	3.0	21916.0	3.0	450.0/s	496.00KB/s	495.80KB/s	85.0	701.64MB
8105	5	3269688.0	12.150.233.22	4	SC	23.41MB	23.41MB	3.0	21787.0	3.0	450.0/s	495.60KB/s	495.60KB/s	85.0	698.42MB
8110	6	3269698.0	12.18.159.20	5	SC	23.39MB	23.39MB	3.0	21778.0	3.0	450.0/s	495.60KB/s	495.60KB/s	85.0	686.86MB
8102	7	3269682.0	12.14.139.7	4	SC	23.54MB	23.54MB	3.0	21908.0	3.0	450.0/s	495.40KB/s	495.40KB/s	85.0	693.41MB
8104	8	3269686.0	12.146.203.5	8	SC	23.49MB	23.49MB	3.0	21860.0	3.0	450.0/s	495.80KB/s	495.00KB/s	85.0	691.64MB
8106	9	3269690.0	12.152.251.81	8	SC	23.46MB	23.46MB	3.0	21840.0	3.0	450.0/s	495.00KB/s	495.00KB/s	85.0	682.59MB
8101	10	3269680.0	12.13.110.2	3	SC	23.39MB	23.39MB	3.0	21773.0	3.0	450.0/s	495.00KB/s	495.00KB/s	85.0	700.39MB
8107	11	3269692.0	12.161.104.3	4	SC	23.40MB	23.40MB	3.0	21784.0	3.0	449.0/s	494.80KB/s	494.80KB/s	85.0	679.66MB
8135	12	3269780.0	129.41.48.5	5	SC	21.27MB	21.27MB	3.0	21780.0	3.0	452.0/s	452.60KB/s	452.60KB/s	85.0	683.21MB
8130	13	3269746.0	129.170.248.227	3	SC	21.27MB	21.27MB	3.0	21784.0	3.0	452.0/s	452.20KB/s	452.20KB/s	85.0	669.92MB
8112	14	3269701.0	12.20.127.4	5	SC	21.29MB	21.29MB	3.0	21800.0	3.0	452.0/s	452.00KB/s	452.00KB/s	85.0	694.85MB
8127	15	3269745.0	128.192.114.13	4	SC	21.35MB	21.35MB	3.0	21865.0	3.0	451.0/s	451.60KB/s	451.60KB/s	85.0	685.59MB
8111	16	3269700.0	12.2.232.2	5	SC	21.38MB	21.38MB	3.0	21892.0	3.0	451.0/s	451.60KB/s	451.60KB/s	85.0	671.73MB
8129	17	3269741.0	129.10.60.32	5	SC	21.32MB	21.32MB	3.0	21829.0	3.0	451.0/s	451.40KB/s	451.40KB/s	85.0	669.37MB
8139	18	3269775.0	130.85.1.5	6	SC	21.41MB	21.41MB	3.0	21921.0	3.0	451.0/s	451.20KB/s	451.20KB/s	85.0	673.75MB
8126	19	3269738.0	128.164.156.25	6	SC	21.32MB	21.32MB	3.0	21829.0	3.0	451.0/s	451.00KB/s	451.00KB/s	85.0	668.88MB
8134	20	3269757.0	129.33.31.14	2	SC	21.33MB	21.33MB	3.0	21844.0	3.0	451.0/s	451.00KB/s	451.00KB/s	85.0	710.62MB
8128	21	3269737.0	129.10.32.77	5	SC	21.37MB	21.37MB	3.0	21885.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	685.89MB
8143	22	3269782.0	131.183.1.1	6	SC	21.40MB	21.40MB	3.0	21910.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	674.75MB
8114	23	3269706.0	12.26.124.2	7	SC	21.39MB	21.39MB	3.0	21906.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	683.80MB
8113	24	3269704.0	12.23.66.6	2	SC	21.40MB	21.40MB	3.0	21911.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	691.25MB
8138	25	3269765.0	130.207.230.5	7	SC	21.29MB	21.29MB	3.0	21796.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	683.93MB
8149	26	3269789.0	132.170.121.25	5	SC	21.40MB	21.40MB	3.0	21911.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	696.77MB
8117	27	3269712.0	12.38.60.20	4	SC	21.25MB	21.25MB	3.0	21760.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	707.45MB
8116	28	3269708.0	12.34.166.132	4	SC	21.38MB	21.38MB	3.0	21897.0	3.0	450.0/s	450.80KB/s	450.80KB/s	85.0	690.88MB
8133	29	3269751.0	129.252.1.5	7	SC	21.27MB	21.27MB	3.0	21777.0	3.0	450.0/s	450.60KB/s	450.60KB/s	85.0	695.60MB
8147	30	3269792.0	132.162.32.244	3	SC	21.40MB	21.40MB	3.0	21915.0	3.0	450.0/s	450.60KB/s	450.60KB/s	85.0	680.23MB
8120	31	3269718.0	12.44.232.12	4	SC	21.25MB	21.25MB	3.0	21755.0	3.0	450.0/s	450.40KB/s	450.40KB/s	85.0	698.99MB
8148	32	3269788.0	132.170.111.103	4	SC	21.39MB	21.39MB	3.0	21900.0	3.0	450.0/s	450.40KB/s	450.40KB/s	85.0	681.07MB
8140	33	3269777.0	131.118.128.11	6	SC	21.25MB	21.25MB	3.0	21758.0	3.0	450.0/s	450.40KB/s	450.40KB/s	85.0	677.48MB
8131	34	3269752.0	129.171.97.20	3	SC	21.25MB	21.25MB	3.0	21758.0	3.0	450.0/s	450.20KB/s	450.20KB/s	85.0	695.43MB
8144	35	3269787.0	131.247.100.43	2	SC	21.40MB	21.40MB	3.0	21909.0	3.0	450.0/s	450.20KB/s	450.20KB/s	85.0	692.45MB
8146	36	3269786.0	131.247.140.1	4	SC	21.37MB	21.37MB	3.0	21878.0	3.0	450.0/s	450.00KB/s	450.00KB/s	85.0	682.09MB
8132	37	3269754.0	129.22.4.4	4	SC	21.40MB	21.40MB	3.0	21913.0	3.0	450.0/s	450.80KB/s	450.00KB/s	85.0	664.81MB
8141	38	3269785.0	131.118.32.11	6	SC	21.40MB	21.40MB	3.0	21911.0	3.0	450.0/s	450.00KB/s	450.00KB/s	85.0	682.52MB
8150	39	3269793.0	134.140.112.14	8	SC	21.37MB	21.37MB	3.0	21886.0	3.0	449.0/s	449.80KB/s	449.80KB/s	85.0	685.66MB
8115	40	3269710.0	12.32.64.12	4	SC	21.21MB	21.21MB	3.0	21722.0	3.0	449.0/s	449.80KB/s	449.80KB/s	85.0	687.62MB
8123	41	3269725.0	12.5.228.3	4	SC	21.24MB	21.24MB	3.0	21753.0	3.0	449.0/s	449.80KB/s	449.80KB/s	85.0	679.42MB
8121	42	3269720.0	12.45.63.3	5	SC	21.37MB	21.37MB	3.0	21883.0	3.0	449.0/s	449.60KB/s	449.60KB/s	85.0	681.43MB
8118	43	3269714.0	12.4.218.17	6	SC	21.38MB	21.38MB	3.0	21895.0	3.0	449.0/s	449.60KB/s	449.60KB/s	85.0	677.79MB
8124	44	3269736.0	128.114.48.10	7	SC	21.40MB	21.40MB	3.0	21911.0	3.0	449.0/s	449.60KB/s	449.60KB/s	85.0	692.41MB
8119	45	3269717.0	12.43.220.6	6	SC	21.36MB	21.36MB	3.0	21874.0	3.0	449.0/s	449.40KB/s	449.40KB/s	85.0	661.02MB
8136	46	3269767.0	129.95.20.2	6	SC	21.37MB	21.37MB	3.0	21882.0	3.0	449.0/s	449.40KB/s	449.40KB/s	85.0	667.93MB
8142	47	3269778.0	131.123.100.7	4	SC	21.32MB	21.32MB	3.0	21830.0	3.0	449.0/s	449.40KB/s	449.40KB/s	85.0	670.38MB
8122	48	3269722.0	12.5.162.16	7	SC	21.25MB	21.25MB	3.0	21763.0	3.0	449.0/s	449.20KB/s	449.20KB/s	85.0	693.94MB
8137	49	3269764.0	130.160.20.19	3	SC	21.41MB	21.41MB	3.0	21920.0	3.0	449.0/s	449.20KB/s	449.20KB/s	85.0	668.22MB
total				239		1.06GB	2.09GB	150.0	1092617.0			22.42MB/s	43.95MB/s		37.63GB
avg				9											1.48GB

Figure 3.4 Results of MQTT log collector

CHAPTER 4

EVALUATION

This chapter presents the results of experiments carried out to evaluate MQTT2EdgePeer. The experiments were conducted using a testbed consisting of hardware and software components that were configured to simulate a distributed IoT environment. The methodology involved preparing the testbed, running experiments, and analyzing the results.

4.1 Methodology

This section provides an overview of the methodology used to evaluate the performance of the system in an edge network. The methodology included the development of a testbed, which involved separating nodes, calculating bandwidth, and simulating latency. Additionally, an experimental design was created to test the system under varying conditions, and various evaluation techniques were employed to analyze the results. The following subsections provide more detail on each aspect of the methodology.

4.1.1 Testbed

To evaluate the performance and functionality of our system, we have deployed it on a real-world testbed. Our testbed comprises a high-performance server with an AMD EPYC 7401P 24-core processor with 48 threads, 256GB of RAM, and running on Ubuntu 22.04.2 LTS. The server is configured to act as a cluster for our system, with each node running on a single process.

The choice of a real-world testbed for our evaluation is motivated by the need to assess our system's performance and robustness under real-world conditions. Real-world testbeds enable us to evaluate our system in a controlled environment that closely mimics the actual deployment scenario, which is critical to identifying and addressing potential issues early on.

4.1.2 Node separation and profiling

The experimental approach was used to evaluate the performance of the MQTT2EdgePeer system. The system was deployed on a server with up to 50 nodes. To ensure efficient management of the nodes, each node was run on a separate process. The system was profiled to evaluate critical performance metrics.

4.1.2.1 Bandwidth calculation

We computed the bandwidth usage by analyzing the total incoming and outgoing data by bit per second on each process ID representing a node in our system.

4.1.2.2 Latency simulation

We simulated random latency between each pair of nodes within the ring, assuming deployment on an edge network. The simulated latency values ranged from 1 to 10 milliseconds, emulating a test-bed with numerous nodes in close proximity, such as a large-scale enterprise network within the same city. Subsequently, we calculated the average latency of all delivered messages for each process ID. In our server-based experiments, clock synchronization is implicitly guaranteed as everything was run on the server.

4.1.2.3 Message delivery calculation

We profiled all published messages that were delivered to their corresponding subscriber nodes. Next, we calculated the delivery rate and the total number of delivered messages on each process ID.

4.1.3 Baseline

We compared our proposed approaches, Direct Dispatching and Guided Dispatching, with an alternative DSRT approach to Scribe called Scribe S. Additionally, we explored different values

for the fan-out parameter in the Guided Dispatching approach, including 5, 10, 25, and ∞. The fan-out value determines the maximum number of outgoing publish messages from each publisher or subscriber node involved in the message dissemination process. The infinity symbol represents the total number of nodes in the overlay network, providing a theoretical upper limit for the fan-out parameter. It can also be concluded that Guided Dispatching with an infinite fan-out value is equivalent to Direct Dispatching.

4.1.4 Experiment design

We designed diverse experiments to examine the system's performance and address the research questions posed in chapter two. In this regard, we evaluated the system's performance by manipulating the number of nodes in the ring, the number of topics, the number of subscribers, and the number of publishers under different conditions. Furthermore, we assessed the effectiveness of the system's fault tolerance mechanism during the failure of root and coordinator nodes.

4.1.5 Experiment evaluation

We conducted an in-depth analysis of the system's performance using five distinct experiments to demonstrate its scalability, dynamicity, fault-tolerance, and robustness while integrated with the MQTT protocol.

4.1.6 Experiment setup

The configuration for all experiments consisted of up to 50 nodes, 14 publishers, 7 topics, 50 subscribers, and a 4KB publish message payload size. We set an approximate equivalency between the number of subscribers, publishers, and topics by a percentage of the total number of nodes inside the overlay as each subscriber or publisher was linked by a single distinguished MQTT client to a single node within the overlay (i.e., 14 publishers ≡ 28% of 50 nodes). Additionally, every subscriber and publisher transmitted subscription and publication messages to all topics, resulting in a total of up to 98 publishers and 350 subscribers (i.e., total publishers=14×7).

4.1.7 Data visualization

To evaluate the different approaches based on bandwidth usage and latency, we utilized box plots. A box plot displays the distribution of a dataset, including the minimum and maximum values, the median (50th percentile), and the first and third quartiles (25th and 75th percentiles). It also allows us to identify any outliers in the dataset, which are represented as individual points outside the whiskers. Additionally, we depicted the total delivered and delivery rate using bar plots, showcasing the differences between the approaches over time.

4.2 Experiments and analysis

This section presents the results of a series of experiments conducted to evaluate the performance of the MQTT2EdgePeer system. We analyze these results to gain insights into the performance of MQTT2EdgePeer and its potential as a solution for edge networks. Our analysis includes a comparison of different approaches to message dissemination, as well as an evaluation of the impact of subscriber, publisher, topic, and node count on latency and bandwidth usage. We also examine the impact of failure of root and coordinator nodes on the total delivered and delivery rate of publish messages.

To compare the results from the box plots in each experiment, we calculated the maximum values, which represent the average of the highest latency or bandwidth for the worst nodes at all levels of the experiment. Additionally, we calculated the load distribution from the box plots, which is the average standard deviation across all levels of the experiment. It is also worth noting that the similar results observed for Guided Dispatching ($F=\infty$) and Direct Dispatching can be attributed to the similarity between these two approaches, as described in subsection (4.1.3).

These findings shed light on the strengths and limitations of MQTT2EdgePeer and provide a basis for further research in this area.

4.2.1 Experiment one - Increasing the number of subscribers

In this experiment, we assessed the ability of MQTT2EdgePeer to handle changes in message volume and distribute the load as the number of subscribers increased. We dynamically increased the number of subscribers from 25% to 100% of the total node count in the overlay.

Figure 4.1 illustrates the results of experiment one regarding the average delivery latency within the overlay. The data indicate that both Direct Dispatching and Guided Dispatching with infinite fan-out (F=∞) exhibit maximum values that are comparatively lower than Scribe S, regardless of the subscriber levels. It is worth noting that the median value for (A) is 0 due to a significant number of nodes having no subscribers or not being involved in message dissemination. Additionally, an increase in subscribers, accompanied by a decrease in fan-out values for Guided Dispatching, leads to a latency increase caused by the message dissemination with more hops. Our findings confirm that Direct Dispatching outperforms Scribe S by reducing the maximum values by 14% and improving load distribution and consistency by 11% on average over the average delivery latency, for all subscriber counts.

The results of experiment one regarding the bandwidth usage of nodes within the overlay are illustrated in Figure 4.2. As shown, both Direct Dispatching and Guided Dispatching with all fan-out values exhibit lower maximum bandwidth usage values than Scribe S. As the number of subscribers increases, Scribe S experiences a significant increase in the maximum bandwidth usage due to the high message load on root nodes, which can be observed as outliers for subscribers with 50%, 75%, and 100% of the total nodes. Additionally, Guided Dispatching with lower fan-out values demonstrates lower bandwidth usage compared to Scribe S, particularly for higher subscriber counts. On average, Guided Dispatching (F=5) displays 40% lower maximum bandwidth usage values and 31% better load distribution and consistency compared to Scribe S for all subscriber levels.

To summarize, the findings demonstrate the effective management of MQTT2EdgePeer in handling dynamic increases in subscribers across all levels. Moreover, our analysis confirms that Direct Dispatching provides the best performance in terms of average delivery latency

compared to the multi-hop Scribe S and Guided Dispatching approaches. It is observed that multi-hop approaches resulted in higher latencies in this experiment. On the other hand, Guided Dispatching emerges as the optimal choice in terms of bandwidth usage. However, it is noteworthy that a trade-off exists between the two metrics in the context of Guided Dispatching approaches. Specifically, decreasing the fan-out degree results in an increase in average delivery latency while reducing average bandwidth usage. This behavior is attributable to the involvement of more hops for message dissemination, which leads to better load distribution, albeit higher latency.

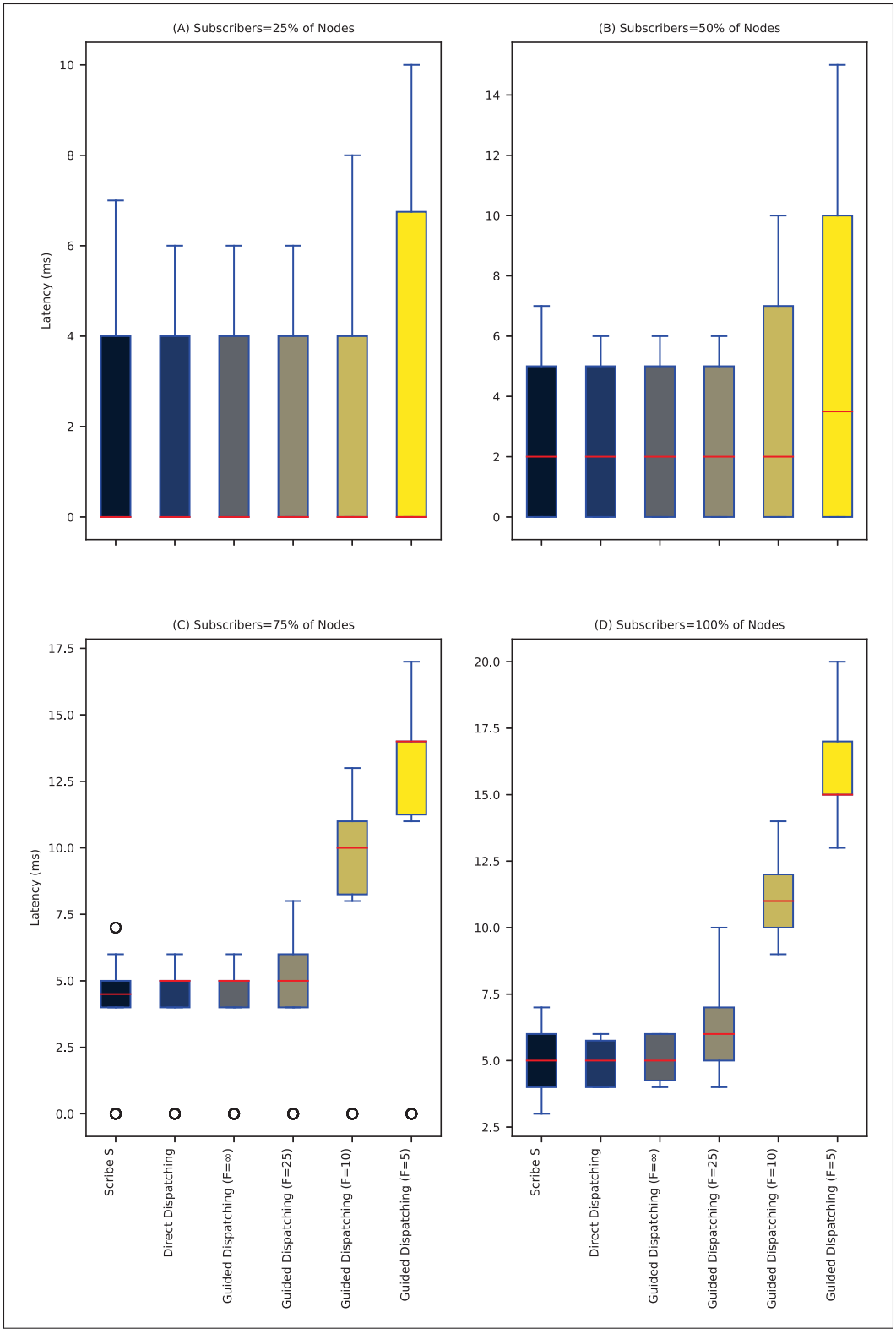


Figure 4.1 Results of increasing the number of subscribers for average delivery latency per node (experiment one)

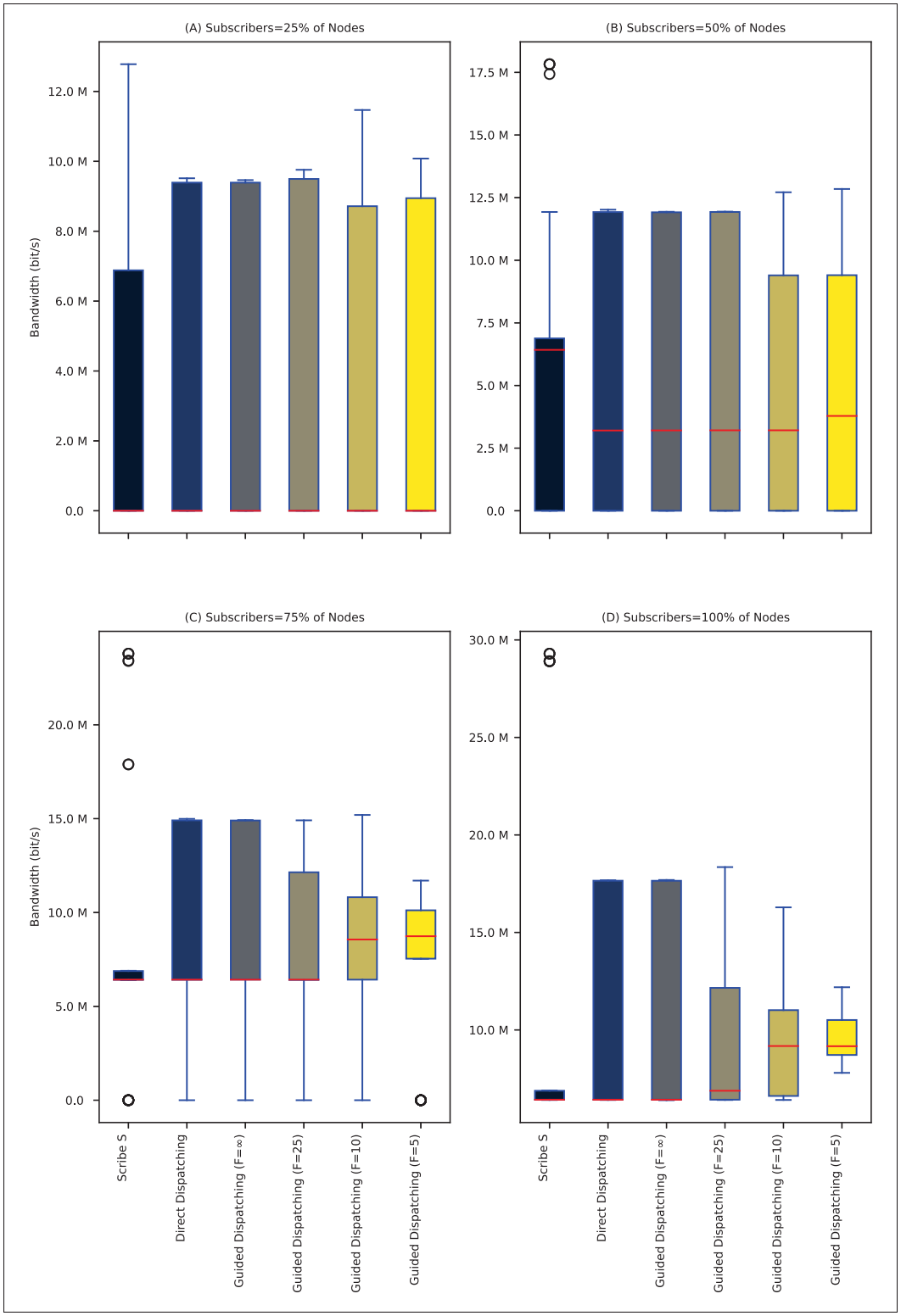


Figure 4.2 Results of increasing the number of subscribers for bandwidth usage per node (experiment one)

4.2.2 Experiment two - Increasing the number of publishers

In this experiment, we examined the impact of increasing the publisher count from 2 percent to 28 percent of the total number of nodes within the overlay network on MQTT2EdgePeer's dynamicity and load distribution management. The distribution of nodes for average delivery latency and bandwidth usage can be seen in Figures 4.3 and 4.4, respectively.

Figure 4.3 shows the results of experiment two for average delivery latency. As can be seen, at publishers with 2 percent of nodes level, Direct Dispatching and Guided Dispatching approaches have more nodes with higher average delivery latency above the median line compared to Scribe S. This is attributed to the possibility of achieving a more evenly distributed load and randomized latency between roots and subscribers when using Scribe S, particularly at lower publisher count levels. However, by increasing the number of publishers, Direct Dispatching and Guided Dispatching (F=∞) offer the same average delivery latency as Scribe S at publishers with 28% of nodes level. It can also be observed that increasing fan-out value and publisher counts leads to a decrease in maximum value and an improvement of node distribution for average delivery latency.

The results of experiment two for bandwidth usage are observable in Figure 4.4. The results demonstrate that at lower publisher levels, Scribe S exhibits the potential to offer better load distribution and lower maximum bandwidth usage with root nodes compared to Direct Dispatching and Guided Dispatching with publisher nodes. However, increasing the number of publishers leads to an increase in the maximum bandwidth usage, number of outliers, and deviation for Scribe S while Direct Dispatching and all Guided Dispatching approaches experience lower maximum usage of bandwidth, fewer outliers, and better consistency and spread of load. Moreover, Guided Dispatching (F=5) provides on average 45% lower maximum bandwidth usage and 64% better consistency and spread of load compared to Scribe S for all publisher levels.

The results of Experiment Two confirmed that increasing the number of publishers caused Scribe S to experience high loads at root nodes, as demonstrated in Figures 4.3 and 4.4. Outlier

nodes exhibited up to 7 ms of average delivery latency and up to 29 megabit bandwidth usage at publishers with 28% of nodes level. In contrast, the Direct Dispatching and Guided Dispatching ($F=\infty$) approaches showed considerably lower maximum bandwidth usage and the same average delivery latency at the highest publisher counts level, indicating MQTT2EdgePeer's ability to handle load distribution by increasing the number of publishers. Furthermore, it is evident that increasing the number of publishers and decreasing the fan-out value for Guided Dispatching approaches enhances load distribution and decreases the maximum bandwidth usage value. However, this also leads to an increase in message delivery latency.

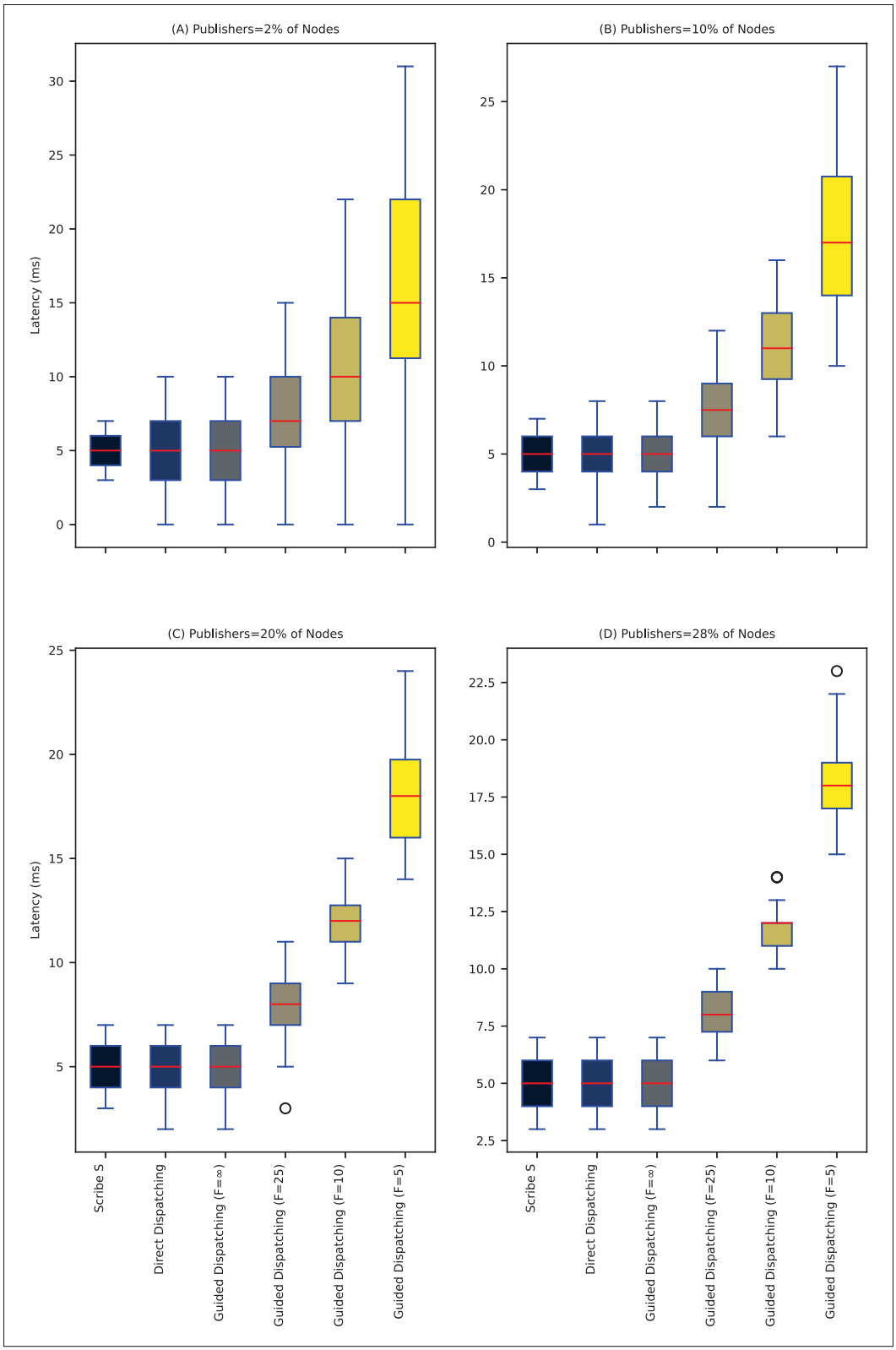


Figure 4.3 Results of increasing the number of publishers for average delivery latency per node (experiment two)

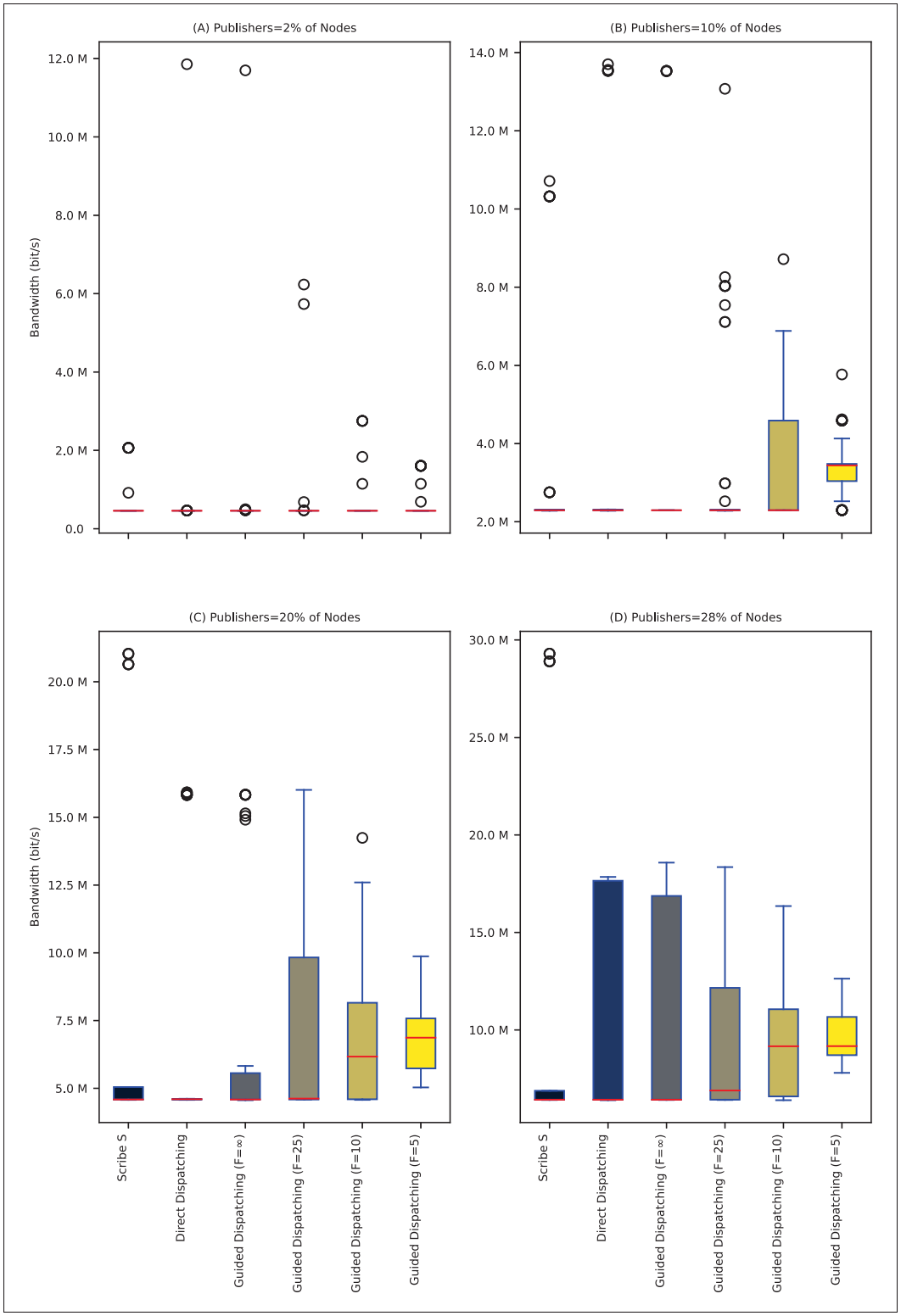


Figure 4.4 Results of increasing the number of publishers for bandwidth usage per node (experiment two)

4.2.3 Experiment three - Increasing the number of topics

In this experiment, we increased the number of topics in order to examine how well MQTT2EdgePeer handles dynamicity and load distribution management across nodes within a structured P2P overlay.

Figure 4.5 depicts the results of increasing topics for the distribution of nodes over average delivery latency. Direct Dispatching and Guided Dispatching (F=∞) experience lower average delivery latency at all levels compared to Scribe S. Additionally, reducing the fan-out value for Guided Dispatching leads to increasing latency regardless of topic level. Overall, Direct Dispatching provides on average 7 percent lower maximum value and 26 percent better consistency and distribution of nodes for average delivery latency compared to Scribe S at all levels.

The results of experiment two for the distribution of nodes over bandwidth usage are illustrated in Figure 4.6. A cursory glance reveals that Direct Dispatching and Guided Dispatching with all fan-out values maintained lower maximum values and better node distribution and consistency with no outliers at all levels compared to Scribe S. Reducing the fan-out value consistently impacts the distribution of load, Guided Dispatching with (F=5) having the lowest maximum value and highest consistency and distribution of loads over bandwidth usage. On average, this leads to a 75 percent improvement in maximum value and an 89 percent improvement in consistency and distribution compared to Scribe S at all topic counts.

Based on the experimental findings, we can confirm that the system is capable of efficiently managing dynamicity and distributing the load. Furthermore, we found that the fan-out value for Guided Dispatching involves a trade-off between average delivery latency and bandwidth usage. Reducing the value increases latency and decreases bandwidth, while increasing the fan-out value leads to opposite results for latency and bandwidth.

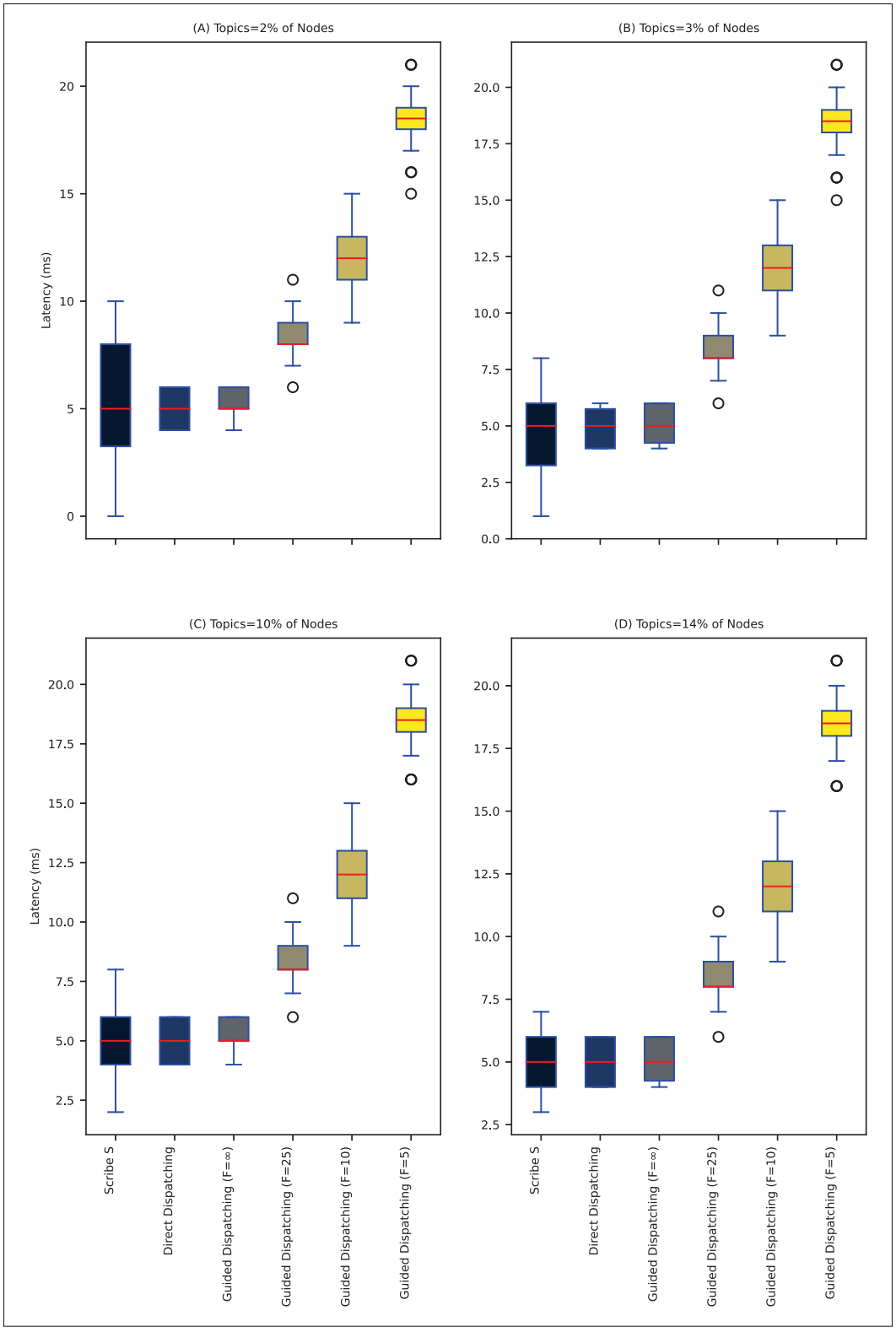


Figure 4.5 Results of increasing the number of topics for average delivery latency per node (experiment three)

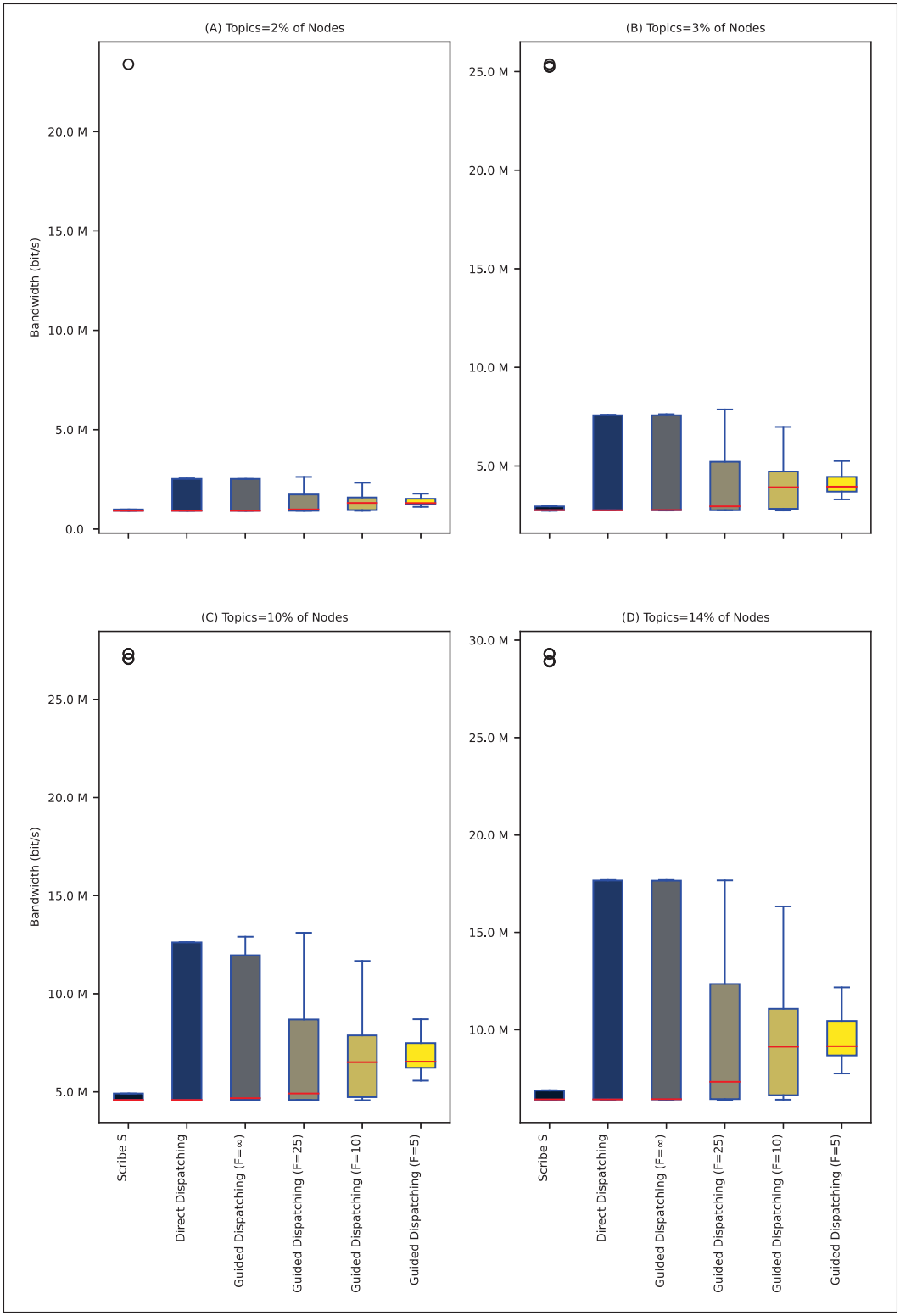


Figure 4.6 Results of increasing the number of topics for bandwidth usage per node (experiment three)

4.2.4 Experiment four: Increasing the number of nodes

In this experiment, we evaluated the scalability of the MQTT2EdgePeer by increasing the number of nodes within the structured P2P overlay network.

Figure 4.7 presents the distribution of nodes over the average delivery latency of publication messages as the number of nodes is increased. The data show that Direct Dispatching and Guided Dispatching (with F values of 25 and ∞, respectively) exhibit lower maximum values for average delivery latency at all levels, compared to Scribe S. Additionally, the results indicate that reducing the fan-out value for Guided Dispatching leads to higher maximum values for average delivery latency. In terms of performance, Direct Dispatching outperforms Scribe S, with an average reduction of 17% in the maximum value of latency and a 16% improvement in the distribution and consistency of nodes across all node counts.

Figure 4.8 presents the distribution of nodes based on their bandwidth usage in experiment four. The figure shows that Direct Dispatching and Guided Dispatching experience lower maximum bandwidth usage at all levels compared to Scribe S. With the number of nodes in the overlay increasing, the maximum bandwidth usage of Scribe S decreases. This can be attributed to two factors: the involvement of non-interested nodes as message forwarders in the dissemination process and the additional hops required to route publications from publishers to roots and then propagate from roots to subscribers. However, non-interested newly joined nodes are not involved in message dissemination for Direct Dispatching and Guided Dispatching, resulting in better consistency across all levels. Direct Dispatching performs better than Scribe S, with an average reduction of 44% in the maximum bandwidth usage and a 28% improvement in distribution and consistency.

The data indicate that MQTT2EdgePeer can be scaled up by adding new nodes to the structured P2P overlay network while maintaining consistency in load distribution. When new nodes are added to the overlay network, they may not be immediately employed in message dissemination between publishers and subscribers, resulting in minimum values of zero for average delivery

latency and bandwidth usage. However, these new nodes can still serve as a coordinator or a shadow.

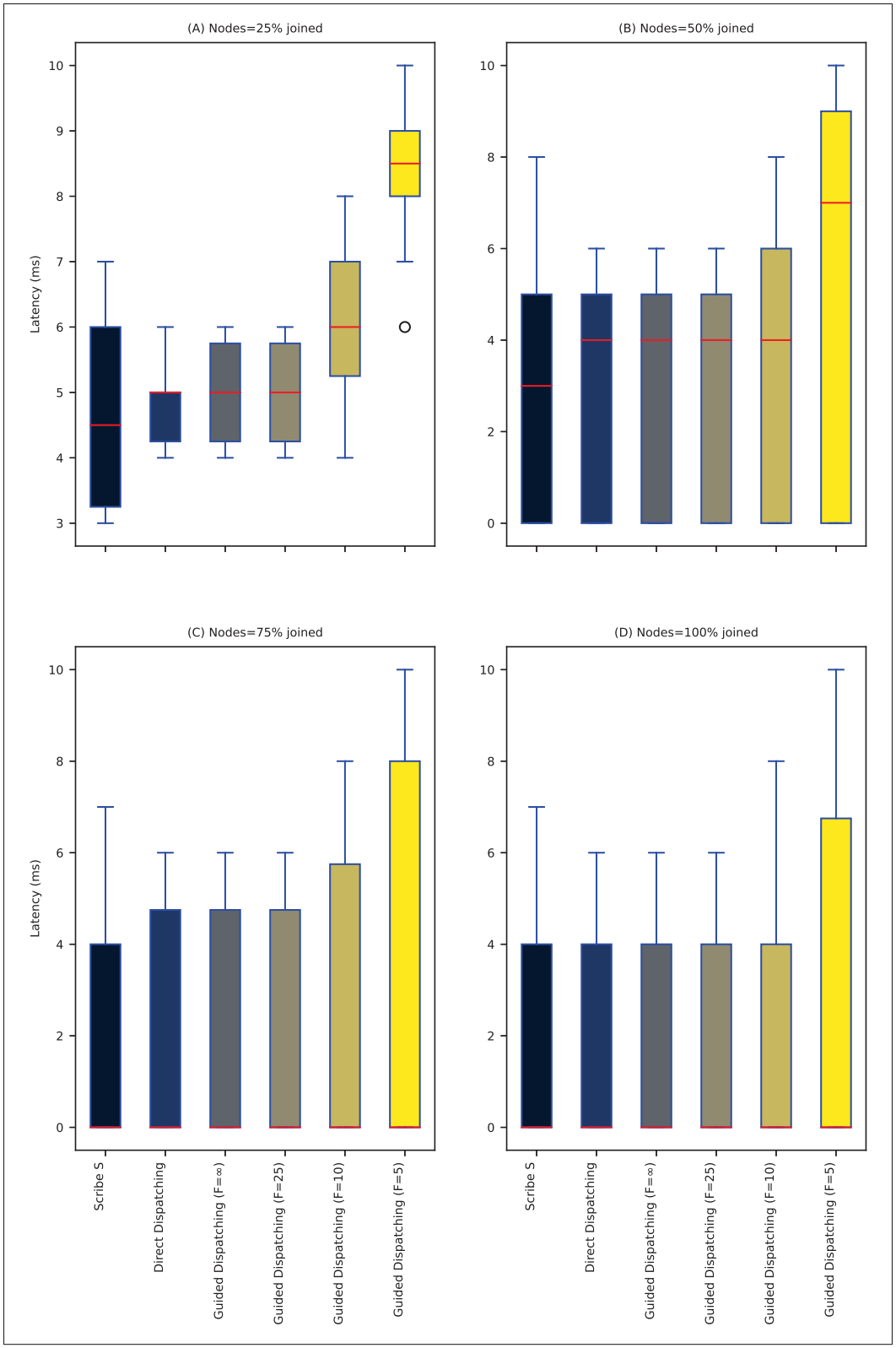


Figure 4.7 Results of increasing the number of nodes for average delivery latency per node (experiment four)

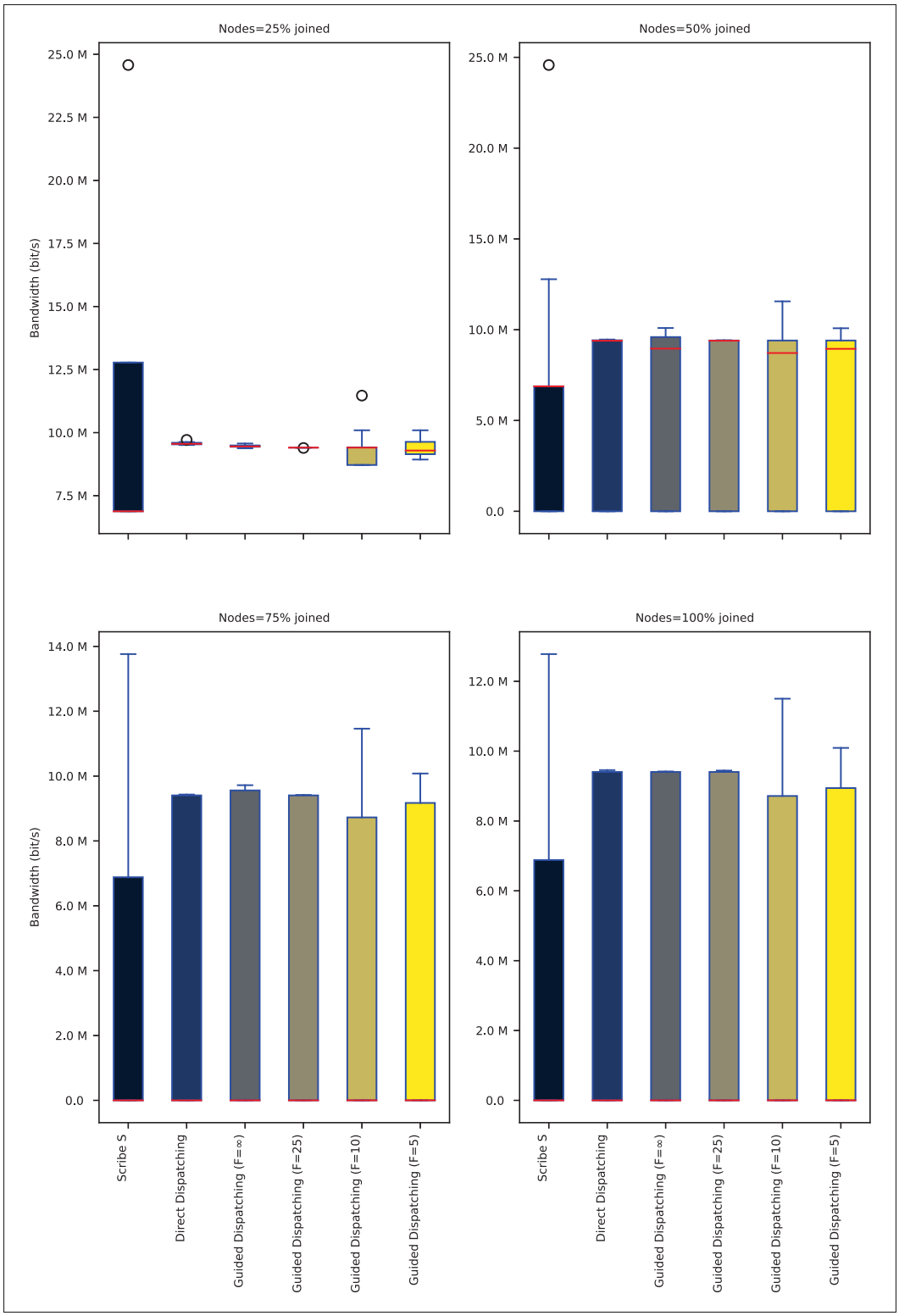


Figure 4.8 Results of increasing the number of nodes for bandwidth usage per node (experiment four)

4.2.5 Experiment five - failure of nodes (coordinator/root)

In this experiment, we tested MQTT2EdgePeer's capability to deliver publication messages in the event of coordinator node failure within the overlay network. The system was configured with a coordinator shadow count of 3 for the purpose of this evaluation.

Figure 4.9 shows the results of node failures on message delivery over time. An analysis of Figure 4.9 (B) reveals that all approaches have the same stable delivery rate for 30 seconds, during which time they drop to different levels after the failure of all root (for Scribe S) and coordinator (for Direct Dispatching and Guided Dispatching) nodes. Direct Dispatching and Guided Dispatching with all fan-out values provide faster recovery and stabilize the delivery rate of messages in a much shorter period of time compared to Scribe S, due to the fault-tolerance mechanism of MQTT2EdgePeer that maintains shadow coordinators to recover the system in case of coordinator failures. It is also observable that lower fan-out values for Guided Dispatching lead to slower stabilization of the delivery rate, as more nodes are involved in message dissemination at lower fan-out values.

Figure 4.9 (A) illustrates the results of total delivered publication messages over the entire experiment period. Upon examining the results of delivered messages, it becomes clear that Direct Dispatching and Guided Dispatching approaches consistently provide higher levels of total delivered messages compared to Scribe S. These findings are consistent with delivery rate and confirm that the faster reconfiguration of Direct Dispatching and Guided Dispatching approaches leads to higher counts of delivered messages. Comparing all approaches with Scribe S in terms of delivery rate and total message delivery counts, Guided Dispatching ($F=\infty$) shows approximately 94 percent faster recovery and about 48 percent higher counts of delivered messages over the experiment period.

Based on the empirical findings presented in this experiment, it is concluded that MQTT2EdgePeer maintains a high level of fault-tolerance and robustness in the event of coordinator failures.

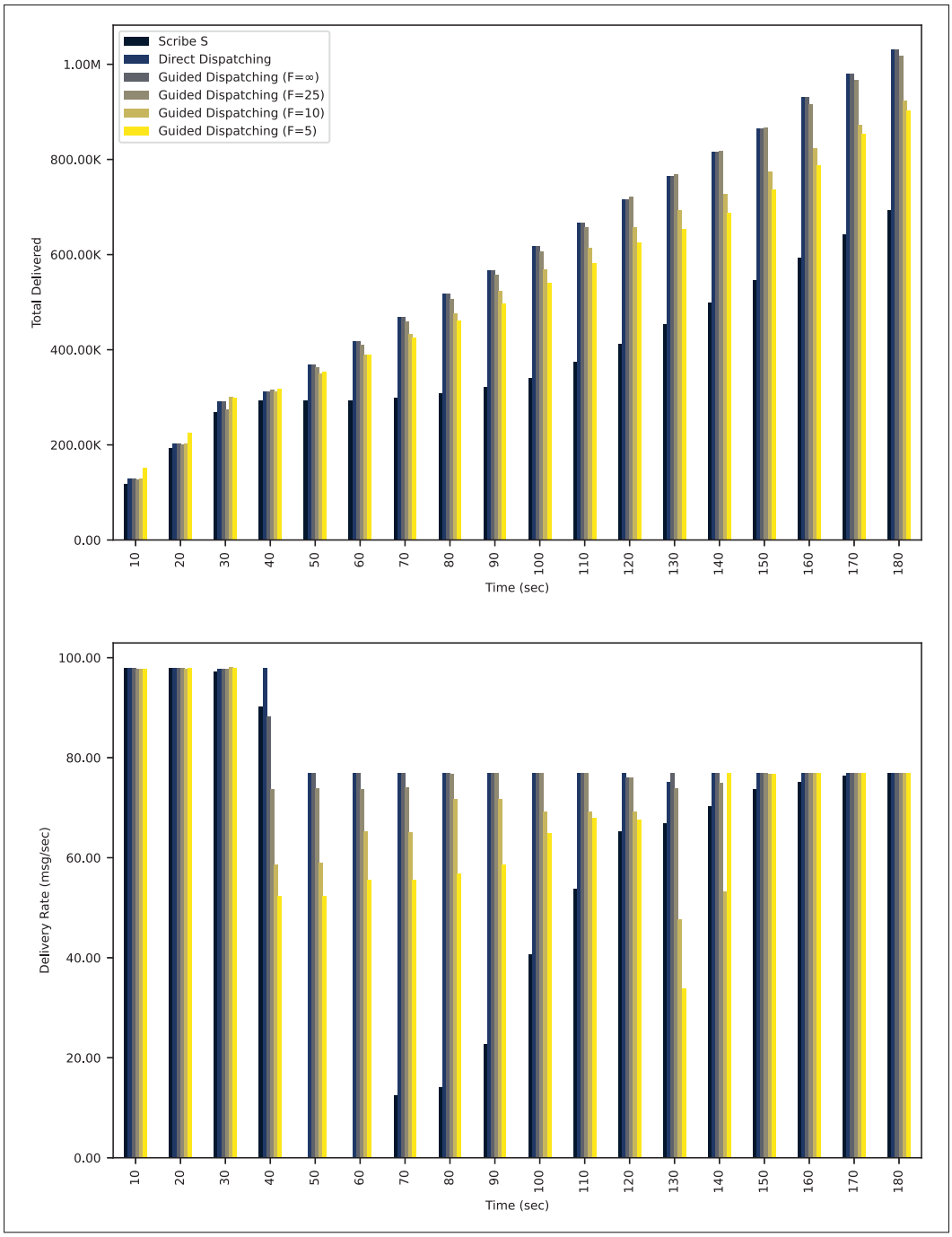


Figure 4.9 Results of the failure of nodes for message delivery (experiment five)

CONCLUSION AND RECOMMENDATIONS

In this thesis, we presented MQTT2EdgePeer a robust and scalable topic-based peer-to-peer communication infrastructure for edge/IoT solutions. MQTT2EdgePeer propagates pub/sub messages using MQTT brokers that are built on top of a structured peer-to-peer overlay network. While most of the other MQTT-based distributed edge-IoT solutions use managed static configuration or clustering approaches, MQTT2EdgePeer utilizes a fully unmanaged peer-to-peer structured overaly network which is one of its kind in this area. Within the overlay network, unlike conventional single root per topic approaches, MQTT2EdgePeer introduces novel coordinator-based approaches for message dissemination. MQTT2EdgePeer avoids a high load of message propagation from a single rendezvous point (root) by coordinating the communications between publishers and subscribers with Direct Disptaching and Guided Dispatching methods. Direct Dispatching and Guided Dispatching approaches distribute message dissemination load by involving publishers instead of roots, therefore, provide lower average delivery latency and bandwidth usage.

Additionally, the system benefits from a built-in fault-tolerance mechanism. This mechanism ensures the delivery of messages despite any coordinator node failures, thereby increasing the overall robustness of the system. Experimental results from the deployment of the system at the edge confirm the benefits of MQTT2EdgePeer properties. Compared to an approach with a single root per topic, our solution presents the best result under dynamic rates of subscriptions, publications, and topics. Furthermore, our experiments demonstrate the scalability and fault-tolerance of the system, as it can effortlessly scale up by joining new nodes to the structured overlay network, and can immediately recover the delivery rate in case of coordinator node failures.

In the future, we plan to implement advanced features of the MQTT protocol such as quality of service, retained messages, wildcards, and last will. Additionally, we aim to leverage AI

techniques to optimize the fan-out value of the Guided Dispatching approach for load distribution improvement and reduced bandwidth usage without compromising latency. We also plan to investigate the use of other overlay networks, such as structured, unstructured, and hybrid, and assess their impact on the performance of MQTT2EdgePeer. These efforts will be part of our ongoing work to further enhance the capabilities and scalability of our peer-to-peer communication infrastructure for edge/IoT solutions.

BIBLIOGRAPHY

Abdelwahab, S. & Hamdaoui, B. (2016). Fogmq: A message broker system for enabling distributed, internet-scale iot applications over heterogeneous cloud platforms. *arXiv preprint arXiv:1610.00620*.

Aekaterinidis, I. & Triantafillou, P. (2006). Pastrystrings: A comprehensive content-based publish/subscribe dht network. *26th IEEE International Conference on Distributed Computing Systems (ICDCS'06)*, pp. 23–23.

Ai, Y., Peng, M. & Zhang, K. (2018). Edge computing technologies for Internet of Things: a primer. *Digital Communications and Networks*, 4(2), 77–86.

Aiomqtt. (2023). Aiomqtt. Retrieved on 2023-04-03 from: https://github.com/mossblaser/aiomqtt/.

Alam, T. (2018). A reliable communication framework and its use in internet of things (IoT). *CSEIT1835111| Received*, 10, 450–456.

Alaya, M. B., Banouar, Y., Monteil, T., Chassot, C. & Drira, K. (2014). OM2M: Extensible ETSI-compliant M2M service platform with self-configuration capability. *Procedia Computer Science*, 32, 1079–1086.

Amoretti, M., Pecori, R., Protskaya, Y., Veltri, L. & Zanichelli, F. (2020). A scalable and secure publish/subscribe-based framework for industrial IoT. *IEEE Transactions on Industrial Informatics*, 17(6), 3815–3825.

Arantes, L., Potop-Butucaru, M. G., Sens, P. & Valero, M. (2010). Enhanced dr-tree for low latency filtering in publish/subscribe systems. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, pp. 58–65.

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I. et al. (2009). *Above the clouds: A berkeley view of cloud computing*.

Aurenhammer, F. (1991). Voronoi diagrams—a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3), 345–405.

Baehni, S., Eugster, P. & Guerraoui, R. (2004). Data-aware multicast. *International Conference on Dependable Systems and Networks, 2004*, pp. 233–242. doi: 10.1109/DSN.2004.1311893.

Baldoni, R. & Virgillito, A. (2005). Distributed event routing in publish/subscribe communication systems: a survey. *DIS, Universita di Roma La Sapienza, Tech. Rep*, 5.

Baldoni, R., Beraldi, R., Quema, V., Querzoni, L. & Tucci-Piergiovanni, S. (2007). TERA: topic-based event routing for peer-to-peer architectures. *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, pp. 2–13.

Banks, A., Briggs, E., Borgendale, K. & Gupta, R. (2019). MQTT Version 5.0. *OASIS Standard*, 7, 102.

Banno, R. & Shudo, K. (2020). Adaptive Topology for Scalability and Immediacy in Distributed Publish/Subscribe Messaging. *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*, pp. 575–583.

Banno, R., Takeuchi, S., Takemoto, M., Kawano, T., Kambayashi, T. & Matsuo, M. (2015). Designing overlay networks for handling exhaust data in a distributed topic-based pub/sub architecture. *Journal of Information Processing*, 23(2), 105–116.

Banno, R., Sun, J., Fujita, M., Takeuchi, S. & Shudo, K. (2017). Dissemination of edge-heavy data on heterogeneous MQTT brokers. *2017 IEEE 6th International Conference on Cloud Networking (CloudNet)*, pp. 1–7.

Bermbach, D., Pallas, F., Pérez, D. G., Plebani, P., Anderson, M., Kat, R. & Tai, S. (2017). A research perspective on fog computing. *International Conference on Service-Oriented Computing*, pp. 198–210.

Bhola, S., Strom, R., Bagchi, S., Zhao, Y. & Auerbach, J. (2002). Exactly-once delivery in a content-based publish-subscribe system. *Proceedings International Conference on Dependable Systems and Networks*, pp. 7–16.

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2000). Achieving scalability and expressiveness in an internet-scale event notification service. *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*, pp. 219–227.

Carzaniga, A., Rosenblum, D. S. & Wolf, A. L. (2001). Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3), 332–383. doi: 10.1145/380749.380767.

Castro, M., Druschel, P., Kermarrec, A.-M. & Rowstron, A. I. (2002). SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in communications*, 20(8), 1489–1499.

Chand, R. & Felber, P. (2004). XNET: a reliable content-based publish/subscribe system. *Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, 2004.*, pp. 264–273. doi: 10.1109/RELDIS.2004.1353027.

Chand, R. & Felber, P. (2005). Semantic Peer-to-Peer Overlays for Publish/Subscribe Networks. *Euro-Par 2005 Parallel Processing*, (Lecture Notes in Computer Science), 1194–1204. doi: 10.1007/11549468_130.

Chen, C. & Tock, Y. (2015). Design of routing protocols and overlay topologies for topic-based publish/subscribe on small-world networks. *Proceedings of the Industrial Track of the 16th International Middleware Conference*, pp. 1–7.

Chen, C., Jacobsen, H.-A. & Vitenberg, R. (2016). Algorithms Based on Divide and Conquer for Topic-Based Publish/Subscribe Overlay Design. *IEEE/ACM Transactions on Networking*, 24(1), 422-436. doi: 10.1109/TNET.2014.2369346.

Chen, C., Tock, Y. & Girdzijauskas, S. (2018). Beaconvey: Co-design of overlay and routing for topic-based publish/subscribe on small-world networks. *Proceedings of the 12th ACM International Conference on distributed and event-based systems*, pp. 64–75.

Chen, C., Vitenberg, R. & Jacobsen, H.-A. (2021). Building fault-tolerant overlays with low node degrees for topic-based publish/subscribe. *IEEE Transactions on Dependable and Secure Computing*.

Chockler, G., Melamed, R., Tock, Y. & Vitenberg, R. (2007). Constructing scalable overlays for pub-sub with many topics. *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*, pp. 109–118.

CORBA, O. & Specification, I. (1999). Object management group. *Joint revised submission OMG document orbos/99-02*.

Costa, P. & Picco, G. P. (2005). Semi-probabilistic content-based publish-subscribe. *25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, pp. 575–585.

Costa, P., Migliavacca, M., Picco, G. P. & Cugola, G. (2003). Introducing reliability in content-based publish-subscribe through epidemic algorithms. *Proceedings of the 2nd international workshop on Distributed event-based systems*, pp. 1–8.

Cugola, G., Di Nitto, E. & Fuggetta, A. (2001). The JEDI event-based infrastructure and its application to the development of the OPSS WFMS. *IEEE transactions on Software Engineering*, 27(9), 827–850.

Dabek, F., Cox, R., Kaashoek, F. & Morris, R. (2004). Vivaldi: A decentralized network coordinate system. *ACM SIGCOMM Computer Communication Review*, 34(4), 15–26.

de Araujo, J. P., Arantes, L., Duarte Jr, E. P., Rodrigues, L. A. & Sens, P. (2019). VCube-PS: A causal broadcast topic-based publish/subscribe system. *Journal of Parallel and Distributed Computing*, 125, 18–30.

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. & Vogels, W. (2007). Dynamo: amazon's highly available key-value store. *ACM SIGOPS Operating Systems Review*, 41(6), 205–220. doi: 10.1145/1323293.1294281.

Dedousis, D., Zacheilas, N. & Kalogeraki, V. (2018). On the fly load balancing to address hot topics in topic-based pub/sub systems. *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pp. 76–86.

Demers, A., Gehrke, J., Hong, M., Riedewald, M. & White, W. (2006). Towards expressive publish/subscribe systems. *International Conference on Extending Database Technology*, pp. 627–644.

Diro, A., Reda, H., Chilamkurti, N., Mahmood, A., Zaman, N. & Nam, Y. (2020). Lightweight authenticated-encryption scheme for Internet of Things based on publish-subscribe communication. *IEEE Access*, 8, 60539–60551.

Duarte, E. P., Bona, L. C. & Ruoso, V. K. (2014). VCube: A provably scalable distributed diagnosis algorithm. *2014 5th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, pp. 17–22.

Eclipse. (2023). PahoMQTTPython. Retrieved on 2023-04-03 from: https://github.com/eclipse/paho.mqtt.python/.

eMQTT Bench. (2023). eMQTT-Bench. Retrieved on 2023-04-03 from: https://github.com/emqx/emqtt-bench/.

Esposito, C., Cotroneo, D. & Russo, S. (2013). On reliability in publish/subscribe services. *Computer Networks*, 57(5), 1318–1343. doi: 10.1016/j.comnet.2012.10.023.

Eugster, P. T. & Guerraoui, R. (2002). Probabilistic multicast. *Proceedings International Conference on Dependable Systems and Networks*, pp. 313–322.

Eugster, P. T., Guerraoui, R. & Sventek, J. [Number: REP_WORK]. (2000). Type-Based Publish/Subscribe. Retrieved on 2021-07-12 from: https://infoscience.epfl.ch/record/52358.

Eugster, P. T., Guerraoui, R. & Damm, C. H. (2001). On objects and events. *ACM SIGPLAN Notices*, 36(11), 254–269. doi: 10.1145/504311.504301.

Eugster, P. T., Felber, P. A., Guerraoui, R. & Kermarrec, A.-M. (2003). The many faces of publish/subscribe. *ACM computing surveys (CSUR)*, 35(2), 114–131.

FIWARE. (2023). FIWARE. Retrieved on 2023-01-04 from: https://www.fiware.org/.

Garcia Lopez, P., Montresor, A., Epema, D., Datta, A., Higashino, T., Iamnitchi, A., Barcellos, M., Felber, P. & Riviere, E. (2015). Edge-Centric Computing: Vision and Challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5), 37–42. doi: 10.1145/2831347.2831354.

Garey, M. R. & Johnson, D. S. (2002). Computers and Intractability, vol. 29. wh freeman New York.

Gargees, R. S. & Scott, G. J. (2018). Dynamically scalable distributed virtual framework based on agents and pub/sub pattern for IoT media data. *IEEE Internet of Things Journal*, 6(1), 599–613.

Gascon-Samson, J., Garcia, F.-P., Kemme, B. & Kienzle, J. (2015a). Dynamoth: A scalable pub/sub middleware for latency-constrained applications in the cloud. *2015 IEEE 35th International Conference on Distributed Computing Systems*, pp. 486–496.

Gascon-Samson, J., Kienzle, J. & Kemme, B. (2015b). Dynfilter: Limiting bandwidth of online games using adaptive pub/sub message filtering. *2015 International Workshop on Network and Systems Support for Games (NetGames)*, pp. 1–6.

Gupta, H., Landle, T. C. & Ramachandran, U. (2021). ePulsar: Control Plane for Publish-Subscribe Systems on Geo-Distributed Edge Infrastructure. *2021 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 228–241.

Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P. & Satyanarayanan, M. (2014). Towards wearable cognitive assistance. *Proceedings of the 12th annual international conference on Mobile systems, applications, and services*, pp. 68–81.

Hasenburg, J. & Bermbach, D. (2020). GeoBroker: Leveraging geo-contexts for IoT data distribution. *Computer Communications*, 151, 473–484.

Hasenburg, J., Stanek, F., Tschorsch, F. & Bermbach, D. (2020). Managing latency and excess data dissemination in fog-based publish/subscribe systems. *2020 IEEE international conference on fog computing (ICFC)*, pp. 9–16.

HiveMQ. (2023). HiveMQ. Retrieved on 2023-01-04 from: https://www.hivemq.com/.

Hmissi, F. & Ouni, S. (2022). TD-MQTT: Transparent Distributed MQTT Brokers for Horizontal IoT Applications. *2022 IEEE 9th International Conference on Sciences of Electronics, Technologies of Information and Telecommunications (SETIT)*, pp. 479–486.

Hunkeler, U., Truong, H. L. & Stanford-Clark, A. (2008). MQTT-S — A publish/subscribe protocol for Wireless Sensor Networks. *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE '08)*, pp. 791–798. doi: 10.1109/COMSWA.2008.4554519.

Jamali, J., Bahrami, B., Heidari, A., Allahverdizadeh, P. & Norouzi, F. (2020). *Towards the internet of things*. Springer.

Jayaram, K., Jayalath, C. & Eugster, P. (2010). Parametric subscriptions for content-based publish/subscribe networks. *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 128–147.

Khare, S., Sun, H., Zhang, K., Gascon-Samson, J., Gokhale, A., Koutsoukos, X. & Abdelaziz, H. (2018). Scalable Edge Computing for Low Latency Data Dissemination in Topic-Based Publish/Subscribe. *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pp. 214-227. doi: 10.1109/SEC.2018.00023.

Kreps, J., Narkhede, N., Rao, J. et al. (2011). Kafka: A distributed messaging system for log processing. *Proceedings of the NetDB*, 11, 1–7.

Ledlie, J., Gardner, P. & Seltzer, M. I. (2007). Network Coordinates in the Wild. *NSDI*, 7, 299–311.

Li, G. & Gao, S. (2011). DRScribe: An Improved Topic-Based Publish-Subscribe System with Dynamic Routing. *Web-Age Information Management*, pp. 226–237.

Li, G., Muthusamy, V. & Jacobsen, H.-A. (2008). Adaptive content-based routing in general overlay topologies. *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 1–21.

Li, M., Ye, F., Kim, M., Chen, H. & Lei, H. (2011). A scalable and elastic publish/subscribe service. *2011 IEEE International Parallel & Distributed Processing Symposium*, pp. 1254–1265.

Light, R. A. (2017a). Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2(13), 265. doi: 10.21105/joss.00265.

Light, R. A. (2017b). Mosquitto: server and client implementation of the MQTT protocol. *Journal of Open Source Software*, 2(13), 265.

Longo, E., Redondi, A. E., Cesana, M., Arcia-Moret, A. & Manzoni, P. (2020). MQTT-ST: a spanning tree protocol for distributed MQTT brokers. *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pp. 1–6.

Lv, P., Wang, L., Zhu, H., Deng, W. & Gu, L. (2019). An IoT-oriented privacy-preserving publish/subscribe model over blockchains. *IEEE Access*, 7, 41309–41314.

Lwin, C. H., Mohanty, H. & Ghosh, R. (2004). Causal ordering in event notification service systems for mobile users. *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, 2, 735–740.

Manyika, J., Chui, M., Brown, B., Bughin, J., Dobbs, R., Roxburgh, C., Hung Byers, A. et al. (2011). *Big data: The next frontier for innovation, competition, and productivity*. McKinsey Global Institute.

Moquette. (2023). Moquette. Retrieved on 2023-04-03 from: https://moquette-io.github.io/.

MQTT. (2023). MQTT. Retrieved on 2023-01-04 from: https://mqtt.org/.

Mühl, G. (2002). *Large-scale content-based publish-subscribe systems*. (Ph.D. thesis, Technische Universität).

Muthusamy, V. & Jacobsen, H.-A. (2005). Small Scale Peer-to-Peer Publish/Subscribe. *P2PKM*.

Muthusamy, V. & Jacobsen, H.-A. (2013). Infrastructure-free content-based publish/subscribe. *IEEE/ACM Transactions on Networking*, 22(5), 1516–1530.

Mühl, G. (2002). *Large-Scale Content-Based Publish-Subscribe Systems*. (phd, Technische Universität, Darmstadt). Retrieved from: http://elib.tu-darmstadt.de/diss/000274.

Nakayama, H., Duolikun, D., Aikebaiery, A., Enokidoz, T. & Takizaw, M. (2014). Causal Order of Application Events is P2P Publish/Subscribe Systems. *2014 17th International Conference on Network-Based Information Systems*, pp. 444–449.

Nakayama, H., Duolikun, D., Enokido, T. & Takizawa, M. (2016). Reduction of unnecessarily ordered event messages in peer-to-peer model of topic-based publish/subscribe systems. *2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 1160–1167.

Okanohara, D., Hido, S., Kubota, N., Unno, Y. & Maruyama, H. (2013). Krill: An Architecture for Edge-Heavy Data. *Third Workshop on Architectures and Systems for Big Data, Tel Aviv*.

Oki, B., Pfluegl, M., Siegel, A. & Skeen, D. (1993). The Information Bus: an architecture for extensible distributed systems. *Proceedings of the fourteenth ACM symposium on Operating systems principles*, (SOSP '93), 58–68. doi: 10.1145/168619.168624.

OM2M. (2023). OM2M. Retrieved on 2023-01-04 from: https://www.eclipse.org/om2m/.

Onus, M. & Richa, A. W. (2011). Minimum maximum-degree publish–subscribe overlay network design. *IEEE/ACM Transactions on Networking*, 19(5), 1331–1343.

Onus, M. & Richa, A. W. (2016). Parameterized maximum and average degree approximation in topic-based publish-subscribe overlay network design. *Computer Networks*, 94, 307–317.

Panagiotou, N., Zygouras, N., Katakis, I., Gunopulos, D., Zacheilas, N., Boutsis, I., Kalogeraki, V., Lynch, S. & O'Brien, B. (2016). Intelligent urban data monitoring for smart cities. *Joint European conference on machine learning and knowledge discovery in databases*, pp. 177–192.

Pedrosa, F. & Rodrigues, L. (2021). Reducing the subscription latency in reliable causal publish-subscribe systems. *Proceedings of the 36th Annual ACM Symposium on Applied Computing*, pp. 203–212.

Pereira, C., Cardoso, J., Aguiar, A. & Morla, R. (2018). Benchmarking Pub/Sub IoT middleware platforms for smart services. *Journal of Reliable Intelligent Environments*, 4(1), 25–37. doi: 10.1007/s40860-018-0056-3.

Pereira, C. M., Lobato, D. C., Teixeira, C. A. & Pimentel, M. G. (2008). Achieving causal and total ordering in publish/subscribe middleware with dsm. *Proceedings of the 3rd workshop on Middleware for service oriented computing*, pp. 61–66.

Pietzuch, P. R. & Bacon, J. M. (2002). Hermes: A distributed event-based middleware architecture. *Proceedings 22nd international conference on distributed computing systems workshops*, pp. 611–618.

Prakash, R., Raynal, M. & Singhal, M. (1996). An efficient causal ordering algorithm for mobile computing environments. *Proceedings of 16th International Conference on Distributed Computing Systems*, pp. 744–751.

Ratnasamy, S., Francis, P., Handley, M., Karp, R. & Shenker, S. (2001). A scalable content-addressable network. *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172.

Rausch, T., Nastic, S. & Dustdar, S. (2018). Emma: Distributed qos-aware mqtt middleware for edge computing applications. *2018 IEEE International Conference on Cloud Engineering (IC2E)*, pp. 191–197.

Razzaque, M. A., Milojevic-Jevric, M., Palade, A. & Clarke, S. (2015). Middleware for internet of things: a survey. *IEEE Internet of things journal*, 3(1), 70–95.

Redondi, A. E., Arcia-Moret, A. & Manzoni, P. (2019). Towards a scaled iot pub/sub architecture for 5g networks: the case of multiaccess edge computing. *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 436–441.

Rice. (2009). freepastry. Retrieved on 2023-04-03 from: https://www.freepastry.org//.

Rowstron, A. & Druschel, P. (2001). Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 329–350.

Sanlı, C. & Lambiotte, R. (2015). Local variation of hashtag spike trains and popularity in twitter. *PloS one*, 10(7), e0131704.

Santos, V. & Rodrigues, L. (2019). Localized reliable causal multicast. *2019 IEEE 18th International Symposium on Network Computing and Applications (NCA)*, pp. 1–10.

Satyanarayanan, M. (2017). The emergence of edge computing. *Computer*, 50(1), 30–39.

Satyanarayanan, M., Bahl, P., Caceres, R. & Davies, N. (2009). The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4), 14–23.

Satyanarayanan, M., Lewis, G., Morris, E., Simanta, S., Boleng, J. & Ha, K. (2013). The role of cloudlets in hostile environments. *IEEE Pervasive Computing*, 12(4), 40–49.

Satyanarayanan, M., Schuster, R., Ebling, M., Fettweis, G., Flinck, H., Joshi, K. & Sabnani, K. (2015a). An open ecosystem for mobile-cloud convergence. *IEEE Communications Magazine*, 53(3), 63–70.

Satyanarayanan, M., Simoens, P., Xiao, Y., Pillai, P., Chen, Z., Ha, K., Hu, W. & Amos, B. (2015b). Edge analytics in the internet of things. *IEEE Pervasive Computing*, 14(2), 24–31.

Segall, B., Arnold, D., Boot, J., Henderson, M. & Phelps, T. (2000). Content based routing with elvin4. *Proceedings of AUUG2K*, (39).

Setty, V., Steen, M. v., Vitenberg, R. & Voulgaris, S. (2012). Poldercast: Fast, robust, and scalable architecture for P2P topic-based pub/sub. *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 271–291.

Shi, W. & Dustdar, S. (2016). The promise of edge computing. *Computer*, 49(5), 78–81.

Sisinni, E., Saifullah, A., Han, S., Jennehag, U. & Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE transactions on industrial informatics*, 14(11), 4724–4734.

Staglianò, L., Longo, E. & Redondi, A. E. (2021). D-MQTT: design and implementation of a pub/sub broker for distributed environments. *2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS)*, pp. 1–6.

Taivalsaari, A. & Mikkonen, T. (2017). A Roadmap to the Programmable World: Software Challenges in the IoT Era. *IEEE Software*, 34(1), 72–80. doi: 10.1109/MS.2017.26. Conference Name: IEEE Software.

Tam, D., Azimi, R. & Jacobsen, H.-A. (2004). Building content-based publish/subscribe systems with distributed hash tables. *International Workshop on Databases, Information Systems, and Peer-to-Peer Computing*, pp. 138–152.

Terry, D. B., Theimer, M. M., Petersen, K., Demers, A. J., Spreitzer, M. J. & Hauser, C. H. (1995). Managing update conflicts in Bayou, a weakly connected replicated storage system. *ACM SIGOPS Operating Systems Review*, 29(5), 172–182.

Vinoski, S. (2006). Advanced Message Queuing Protocol. *IEEE Internet Computing*, 10(6), 87–89. doi: 10.1109/MIC.2006.116. Conference Name: IEEE Internet Computing.

Yamamoto, Y. & Hayashibara, N. (2017). Merging topic groups of a publish/subscribe system in causal order. *2017 31st International Conference on Advanced Information Networking and Applications Workshops (WAINA)*, pp. 172–177.

Yu, W., Liang, F., He, X., Hatcher, W. G., Lu, C., Lin, J. & Yang, X. (2018). A Survey on the Edge Computing for the Internet of Things. *IEEE Access*, 6, 6900–6919. doi: 10.1109/ACCESS.2017.2778504. Conference Name: IEEE Access.

Zacheilas, N., Kalogeraki, V., Nikolakopoulos, Y., Gulisano, V., Papatriantafilou, M. & Tsigas, P. (2017). Maximizing determinism in stream processing under latency constraints. *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pp. 112–123.

Zhao, B. Y., Huang, L., Stribling, J., Rhea, S. C., Joseph, A. D. & Kubiatowicz, J. D. (2004a). Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1), 41–53.

Zhao, Y., Kim, K. & Venkatasubramanian, N. (2013). Dynatops: A dynamic topic-based publish/subscribe architecture. *Proceedings of the 7th ACM international conference on Distributed event-based systems*, pp. 75–86.

Zhao, Y., Sturman, D. & Bhola, S. (2004b). Subscription propagation in highly-available publish/subscribe middleware. *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 274–293.

Zhuang, S. Q., Zhao, B. Y., Joseph, A. D., Katz, R. H. & Kubiatowicz, J. D. (2001). Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. *Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video*, pp. 11–20.

Zilhão, L., Morla, R. & Aguiar, A. (2018). A Modular Tool for Benchmarking loT Publish-Subscribe Middleware. *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, pp. 14–19. doi: 10.1109/WoWMoM.2018.8449774.