論文 / 著書情報
Article / Book Information

題目(和文)	
Title(English)	An Architecture of Distributed Pub/Sub Systems Using Structured Overlay Networks
著者(和文)	坂野遼平
Author(English)	Ryohei Banno
出典(和文)	学位:博士(理学), 学位授与機関:東京工業大学, 報告番号:甲第10942号, 授与年月日:2018年9月20日, 学位の種別:課程博士, 審査員:首藤 一幸,増原 英彦,南出 靖彦,脇田 建,松浦 知史,秋山 豊和
Citation(English)	Degree:Doctor (Science), Conferring organization: Tokyo Institute of Technology, Report number:甲第10942号, Conferred date:2018/9/20, Degree Type:Course doctor, Examiner:,,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

An Architecture of Distributed Pub/Sub Systems Using Structured Overlay Networks

A dissertation presented by
Ryohei Banno

supervised by
Kazuyuki Shudo

for the degree of
Doctor of Science

Department of Mathematical and Computing Sciences
Graduate School of Information Sciences and Engineering
Tokyo Institute of Technology
Tokyo, Japan

September 2018

An Architecture of Distributed Pub/Sub Systems Using Structured Overlay Networks

Abstract

In the context of event-driven application development, techniques of pub/sub messaging are promising candidates. Its paradigm provides not only real time dissemination by push-based delivery, but also high tolerance to the transition of relations between publishers and subscribers by their decoupling, e.g., each publisher does not need to be concerned about the location or status of subscribers that will receive its message. However, since a typical architecture has a centralized broker which gathers all published messages and forwards them to corresponding subscribers, it could be a bottleneck and a single point of failure. Especially when we consider handling IoT data, their characteristic of low value density makes the above issue much harder. Namely, a tremendous amount of data is concentrated on the broker with oppressing the network bandwidth, even though most of it will be discarded.

For overcoming the problem, this dissertation focuses on topic-based pub/sub systems which are one of the best known and widely used type of pub/sub systems, and introduces structured overlay networks into them. We assume an architecture, in which many brokers are placed on the edge of a wide area network and they cooperate with each other by composing a structured overlay network. This edge-based architecture makes latency lower as well as avoiding congestion on cloud resources, especially when producing and consuming data have geographical locality. We discuss about the architecture from the following three points of view.

At first, we propose a topic-based pub/sub method using Skip Graph, which is one of the algorithms of structured overlay networks supporting range queries. Although there are some existing studies of topic-based pub/sub messaging based on structured overlay networks, they have the problem of wasting network resources because of lacking adaptability to the characteristic of low value density in IoT data. In particular, the behavior that each publisher node continues to forward messages to a relay node even if there are no subscribers causes redundant messages between brokers. The proposed method regulates publishers and subscribers of the same topic to compose connected subgraphs so that publishers can detect the absence of subscribers and suspend sending messages. This mechanism reduces the redundant messages and thereby can minimize

the load of wide area networks. We confirm the effectiveness of the proposed method quantitatively by simulation experiments.

Secondly, this dissertation focuses on the latency. The above proposed method involves the increase of the required time from publishers to subscribers, even though it brings about high scalability. This could impair the advantage of real time dissemination of pub/sub messaging. We discuss about the improvement of the latency from two aspects. The first one is routing algorithms of Skip Graph. There are several existing algorithms for handling range queries, but they are inefficient regarding the latency. We propose a new algorithm named Split-Forward Broadcasting (SFB), and indicate that it can improve the latency by reducing the average number of hops. The second one is client assignment strategies. The latency is widely influenced by how to assign subscribers to each broker. There are two possible approaches: the intensive assignment by which subscribers having a same topic are accommodated on a same broker, and the extensive assignment by which the subscribers are accommodated on different brokers as far as possible. We formulate the difference of the latency between these two approaches, and discuss about the optimization of subscriber assignment.

Finally, this dissertation gives a discussion about developing the proposed method as a middleware, including the capability of a practical protocol. We focus on MQTT, which is one of the promising protocols for exchanging IoT data. Considering the edge-based architecture, heterogeneity could be a vital issue, i.e., an appropriate product of the MQTT broker could vary according to the different environment of each network edge. We propose Interworking Layer of Distributed MQTT brokers (ILDM), which enables arbitrary kinds of MQTT brokers to cooperate with each other. ILDM provides APIs which facilitate rapid development of variety of cooperation algorithms, including the proposed method. To clarify the feasibility of ILDM, we show two primitive cooperation algorithms which use the APIs, and evaluate on an actual environment. The evaluation is conducted by a benchmark method which we design to have the capability of measuring both a single broker and multiple brokers. Experimental results show that the throughput of multiple brokers running together by ILDM is improved significantly than that of a single broker.

Contents

Listing of figures

Listing of tables

Acknowledgments

First of all, I would like to express my profound gratitude to my supervisor, Prof. Kazuyuki Shudo. He gave me conscientious advices not only on the study in this dissertation, but also on my career path, to broaden my horizons. Also, I would like to thank his group members for their discussion on my research. The dissertation committee consisted of Prof. Toyokazu Akiyama, Prof. Hidehiko Masuhara, Prof. Satoshi Matsuura, Prof. Yasuhiko Minamide, Prof. Kazuyuki Shudo and Prof. Ken Wakita. I deeply appreciate the discerning feedbacks offered by them.

I extend special thanks to Tomoyuki Fujino, Takashi Kambayashi, Tetsuo Kawano, Masato Matsuo, Jingyu Sun, Michiharu Takemoto, and Susumu Takeuchi. They were my colleagues in NTT Network Innovation Laboratories. They provided varied constructive ideas useful for the development of my study, which were grounded on their high expertise and extensive knowledge. In addition, I would like to express my deep appreciation for Kenji Umakoshi, who was also my colleague. If I had not been given his thoughtful encouragement, I would have struggled to overcome various difficulties of my work. The advice and comments offered by Masahiro Fujita were also of great help regarding my study.

Finally, I would like to offer my deepest gratitude to my family, especially to my wife Arisa. They always backed me up in the choices of my life, including the entrance into the Ph.D. program. I would not be where I am today without their support.

1 Introduction

Development of information and communications technology has brought about several paradigm shifts. In the early days of the Internet, the World Wide Web has enabled people to acquire various information helpful for their life from all over the world. In the past two decades, information originating has become easier gradually; after the prevalence of personal web pages among certain people, blogs have become popular, and then micro-blogs and social networking services have followed. These services achieved that everyone can originate his/her information quite easily. And nowadays, our life is steadily getting programmable. A variety of smart devices have appeared such as smart phones, wearable devices, and smart speakers. These devices provide us the opportunity to choose applications and install on them according to lifestyles. Arising of many kinds of frameworks which facilitate quick development of cooperative applications, e.g., flow-based application platforms [Node-RED], are accelerating the programmability.

Adding to that, the number of things having network connectivity is estimated to reach 100 billion by 2020 [Hodges et al. 2013]. In the near future, it is expected that massive volume of information is exchanged between such things so that various useful smart services not only visualizing information but also actuating things in physical space are provided. This sort of services composed of cooperative things are typically

event-driven, i.e., things are actuated by real time information.

To develop such event-driven applications, the techniques of pub/sub messaging [Eugster et al. 2003] is promising candidates. In pub/sub systems, senders of messages do not specify receivers directly. That is, senders make messages to include some information about the contents of themselves, while receivers specify interests in advance. Here, senders and receivers are called publishers and subscribers. This scheme provides convenient decoupling between publishers and subscribers, e.g., each publisher has no concern with the location or the status of subscribers that will receive a published message.

Typical pub/sub systems have a server called a "broker" [Snyder et al. 2008; Videla and Williams 2012; HiveMQ], whereas publishers and subscribers are called clients. The broker gathers all published messages and forwards them to subscribers according to their interests. In other words, these systems form a centralized architecture. Since such architecture is easy to implement, it is used for a wide variety of applications e.g., web syndication, disaster prevention systems, and social networking services.

However, in this sort of architecture, the broker could be a bottleneck and a single point of failure. Especially when we consider the future smart service systems as mentioned above, there is a difficulty of handling so-called "data exhaust", which is predicted to occupy most of the IoT data [Manyika et al. 2011]. The characteristics of data exhaust can be summarized as follows:

Low value density

Data are generated as byproducts and often without specific uses. These data have low or no value most of the time, but at times they are highly useful.

High generation frequency

Data are automatically and continuously generated by IoT devices, unlike the traditional Internet in which people generate most of the content.

Breadth of generation area

Data are generated over a wide area, since those are from things in the physical space.

Namely, a tremendous amount of data is concentrated on the broker with oppressing the network bandwidth. This is unprofitable because the data arriving at the broker are mostly discarded due to its low value density as depicted in Figure 1.1. Such

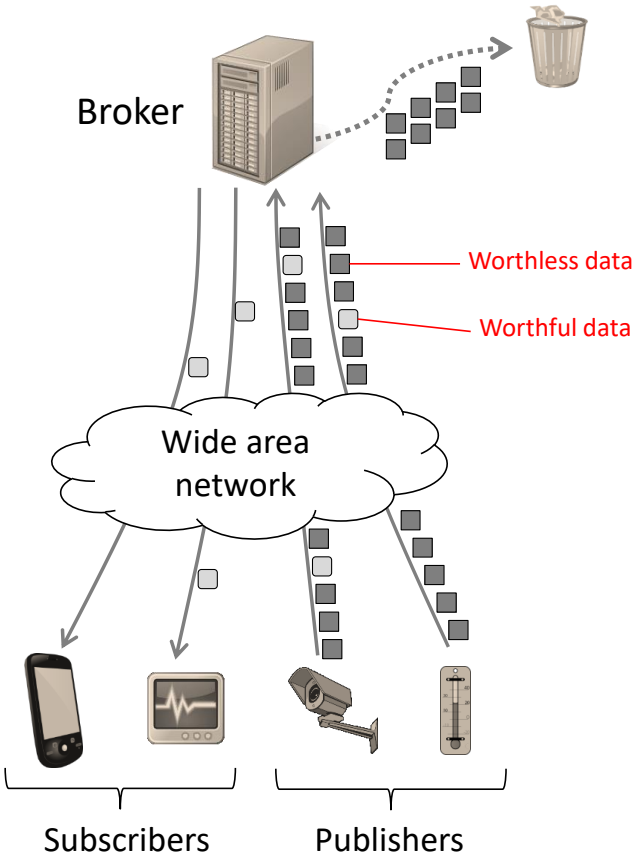

Figure 1.1: Centralized architecture of pub/sub messaging.

characteristics of IoT data are also called “edge-heavy” [Okanohara et al. 2013], and they become an obstacle to achieving future large scale cooperative systems.

Accordingly, we suppose an efficient architecture using edge brokers as shown in Figure 1.2. Edge brokers are placed at the edge of the wide area network, i.e., they are installed over a wide area. Publishers and subscribers connect to the closest one. In this architecture, locally consumed data which are both sent from and received by the clients connected to a same edge broker do not encroach on the wide area network. In addition, by exchanging only worthful data among edge brokers, this architecture can prevent imprudent forwarding of data exhaust to the wide area network. Such concepts of focusing on placing computational resources at the outer edge of wide area networks has become increasingly important as can be seen from the proposal of “Edge

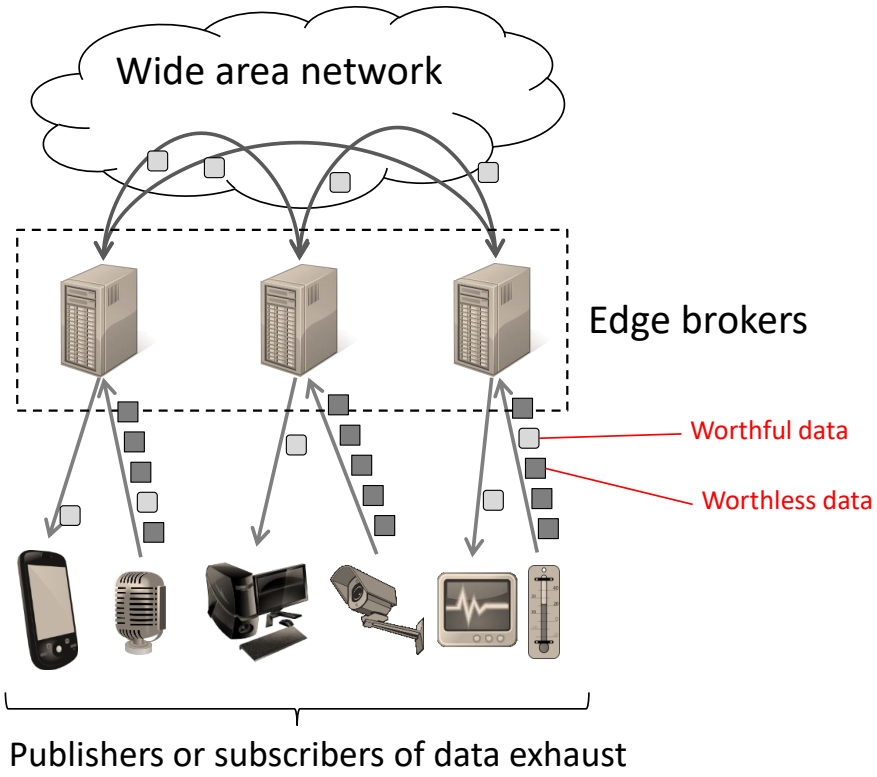

Figure 1.2: Edge-based architecture.

computing" [Shi et al. 2016]. Our basic idea in this dissertation is to compose an autonomous distributed system by the edge brokers so that it has high scalability by avoiding a single point of failure and a performance bottleneck. Note that an edge broker plays the role of a publisher or a subscriber in the network of edge brokers, if it has a publisher or a subscriber as its client. Hereafter, a publisher or a subscriber means an edge broker, except when we explicitly mention the user end client like an IoT device.

We focus on topic-based pub/sub which is the best known and the most widely used pattern of pub/sub messaging, and aim for realizing the edge-based architecture. Figure 1.3 shows example software components of an IoT application based on the architecture. Data consumer denotes a software module which receives data from other IoT devices via edge broker(s), and actuates things and/or visualizes by the results of processing the data. Data producer denotes a software module which sends out data

towards other IoT devices via edge broker(s). Interest handler manages what kind of data is required by the local IoT device, namely the interest of the local IoT device, and converts the interest as subscribe requests.

We assume that an application vendor deploys the distributed pub/sub middleware to the edge brokers which belong to it, while the IoT devices belong to the end users. The edge brokers may be prepared by the vendor or lent out by other service vendors as like IaaS (Infrastructure as a Service) of the present day. To operate the deployed distributed pub/sub system, the application vendor decides system specification such as the definition of topic names, the format of messages, and so on. We also assume that the vendor develops the software components depicted on the IoT device in the figure as a client application. This application may be provided to users as an embedded device or a software program which they can install it on their devices.

This dissertation aims to provide techniques required for realizing the distributed pub/sub middleware in the figure. The middleware should achieve high throughput and low latency, since the assuming IoT services consist of vast number of cooperative things and are typically event-driven. Considering such middleware is applicable to constructing not only a single integrated system, but also a platform like distributed data flow platform [Teranishi et al. 2017], scalability regarding the throughput and the latency are quite significant. As mentioned above, we focus on topic-based pub/sub, and discuss from three points of view: getting high throughput by introducing structured overlay networks, improvement of latency, and development as a middleware including the capability of a practical protocol.

At first, we introduce structured overlay networks into distributed pub/sub systems. They have not only scalability effective for high throughput, but also suitable properties such as robustness, elimination of a single point of failure, etc. Methods of topic-based pub/sub using structured overlay networks have already been proposed [Castro et al. 2002; Zhuang et al. 2001; Ratnasamy et al. 2001b]. However, these methods do not work efficiently for data exhaust; they are not adaptive to the value of data, so that they waste network resources by gratuitously exchanging worthless data. This could make overall throughput lower.

For overcoming the inefficiency, we propose a topic-based pub/sub method using Skip Graph [Aspnes and Shah 2007], which is one of the algorithms of structured over-

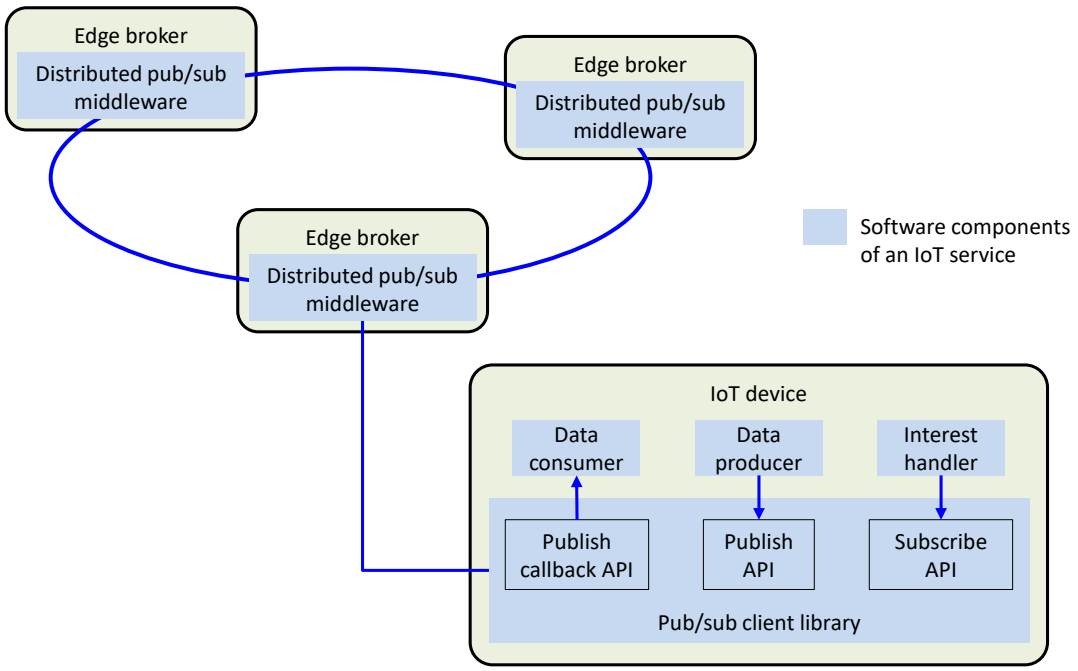

Figure 1.3: Example of software components of edge-based architecture.

lay networks supporting range queries. The proposed method regulates publishers and subscribers of the same topic to compose connected subgraphs so that publishers can detect the absence of subscribers and suspend sending messages. To evaluate the proposed method, we implemented simulated programs for the proposed method and one of the existing methods, and conducted some experiments with up to approximately $100,000$ nodes. We confirmed that the proposed method can reduce consumption of network resources by suspending publish messages adaptively, so that consequently can increase the overall throughput.

Secondly, this dissertation focuses on the latency. The above proposed method involves the increase of the required time from publishers to subscribers, even though it brings about high scalability. This could impair the advantage of real time dissemination of pub/sub messaging. We discuss about the improvement of latency from the following two aspects:

Routing algorithms of Skip Graph

There are existing routing algorithms for range queries, but they are inefficient regarding the latency or the traffic volume. We propose a new algorithm named Split-Forward Broadcasting (SFB), and indicate it can reduce the average number of hops.

Client assignment strategies

The latency is widely influenced by a way to assign subscribers to each broker. We show two possible strategies: the intensive assignment by which subscribers having a same topic are accommodated on a same broker, and the extensive assignment by which the subscribers are accommodated on different brokers as far as possible. We formulate the difference of the latency between these two strategies, and discuss about the optimization of subscriber assignment.

Finally, this dissertation gives considerations about developing the proposed method as a middleware, including the capability of a practical protocol. We focus on MQTT, which is standardized and one of the promising protocols for exchanging IoT data. There are a lot of MQTT broker products: open source, proprietary, embedded appliance, etc. Utilizing existing products is a reasonable way to provide high quality implementation, but heterogeneity could be a vital issue when we consider the edge-based architecture. In other words, an appropriate product of MQTT broker could vary according to the different environment of each network edge. To address this issue, we propose Interworking Layer of Distributed MQTT brokers (ILDM), which enables arbitrary kinds of MQTT brokers to cooperate with each other. For clarifying the feasibility, we provide two basic cooperation algorithms, including the way to furnish MQTT-specific functions such as QoS and Retain. We also formulate a benchmark method which can be used for both a single broker and multiple brokers. Experimental results show that the throughput of five brokers running together by ILDM is improved 4.3 times at maximum than that of single broker.

1.1 Structure of This Dissertation

The contributions of this dissertation are threefold:

- First, we propose a topic-based pub/sub method using Skip Graph for achieving

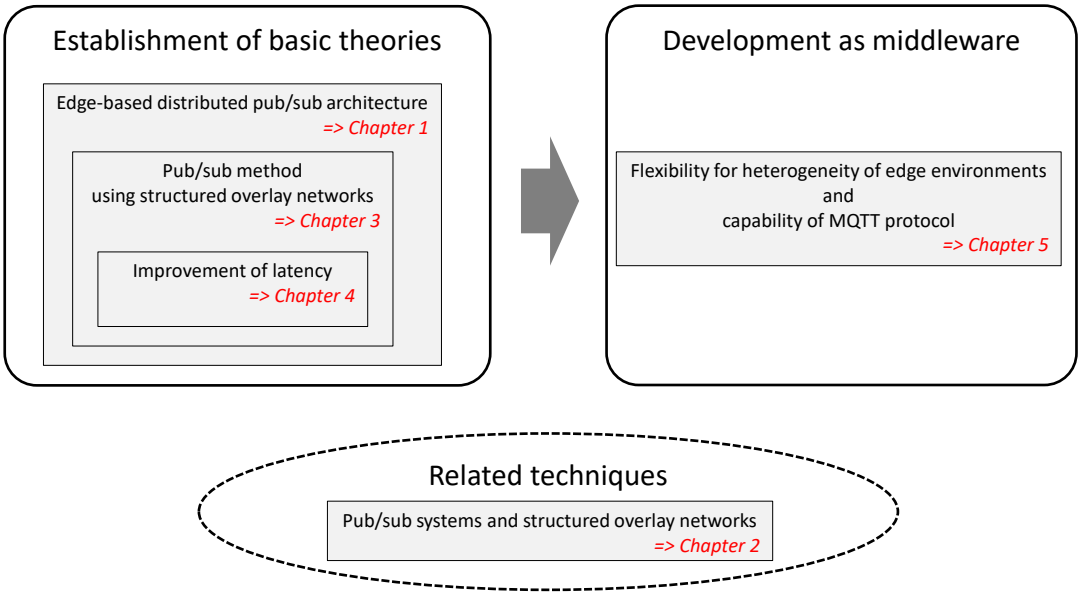

Figure 1.4: Relation between chapters.

high throughput. It enables publishers to detect the absence of subscribers and suspend sending messages.

- Second, we present two techniques for improving the latency in the distributed pub/sub architecture; an efficient routing algorithm of Skip Graph, and a formulation regarding client assignment strategies.
- Third, we show the result of feasibility study towards developing the proposed method as a middleware. We provide a new mechanism which enables arbitrary kinds of MQTT brokers to cooperate with each other. We also formulate a benchmarking method.

Figure 1.4 shows the relation between chapters in this dissertation. In Chapter 2, we explain some techniques commonly related to the research topics composing this dissertation. In Chapter 3, we illustrate our method using structured overlay networks for achieving high throughput. In Chapter 4, we introduce techniques for improving the latency. In Chapter 5, we show the approaches towards developing as a middleware. Finally, we summarize and conclude this dissertation in Chapter 6.

2

Pub/Sub Systems and Structured Overlay Networks

In this chapter, we explain some techniques commonly related to our studies in this dissertation.

2.1 Pub/Sub Systems

As previously stated, pub/sub systems can provide convenient decoupling between publishers and subscribers. Different from the traditional request-reply paradigm, publishers do not specify subscribers directly, and subscribers also do not specify publishers directly. Alternatively, subscribers specify their interests regarding the contents of messages in advance. When a publisher sends a message, it involves some information about the contents of the message. The message will be delivered to subscribers whose interests are matched to the information.

Eugster et.al., [Eugster et al. 2003] says the paradigm of pub/sub messaging provides the following three decoupling dimensions between publishers and subscribers:

- Space decoupling: publishers and subscribers need not to know each other.

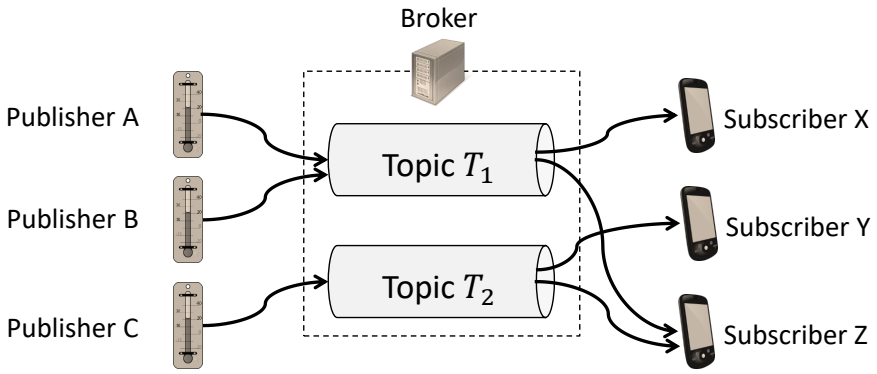

Figure 2.1: Message flow of topic-based pub/sub.

- Time decoupling: publishers and subscribers need not to be actively participating at the same time.
- Synchronization decoupling: publishers and subscribers process messages asynchronously.

2.1.1 Patterns of Pub/Sub Messaging

There exist two principle patterns of pub/sub messaging: topic-based pub/sub and content-based pub/sub. Topic-based pub/sub uses logical channels called "topics" for exchanging messages as shown in Figure 2.1. Publishers are required to give a topic to each message they send out. Subscribers specify topics of interest, and receive messages published on those topics. On the other hand, in content-based pub/sub, subscribers can specify their interests more flexibly than in topic-based pub/sub. For example, Gryphon [Aguilera et al. 1999] provides a mechanism of matching tree so that subscribers can specify their interests as a set of conditions regarding attributes such as "$(city = NewYork) \wedge (temperature < 40)$".

In this dissertation, we focus on topic-based pub/sub. This is because topic-based pub/sub is more primitive, so that it has well applicability for distributed environments. Namely, it is suitable for the large scale systems we are assuming as mentioned in Section 1. Although content-based pub/sub is more flexible for expressing the interests

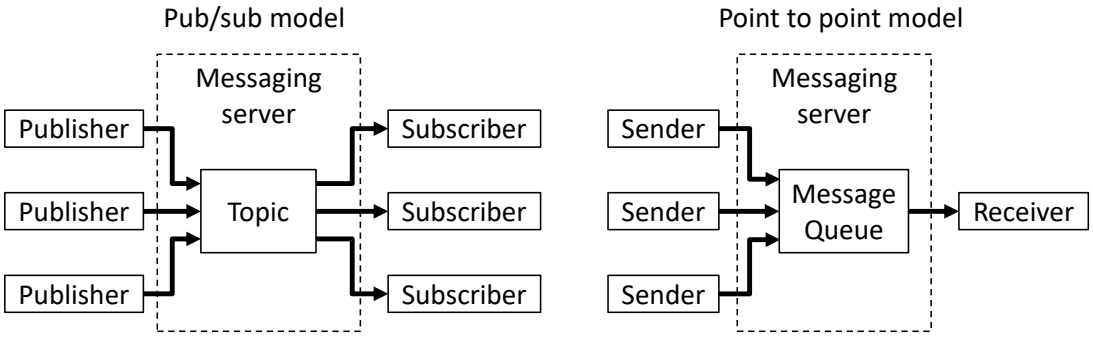

Figure 2.2: Messaging models in JMS.

of subscribers, the prevalence of key-value stores in recent years reflects that handling data by using labels like topics is enough versatile.

Topic-based pub/sub systems are most often implemented over TCP/IP. In particular, there exist some studies of implementing topic-based pub/sub as overlay networks. We explain about these studies later in Section 3.1. On the other hand, there is another approach of implementing it in lower network layers. LIPSIN [Jokela et al. 2009] is one of such studies. LIPSIN is a multicast forwarding fabric for large-scale topic-based pub/sub messaging. It achieves efficient energy consumption by its simple forwarding decisions and small forwarding tables. In this dissertation, we assume implementing over TCP/IP, since it is tolerant to the change of lower layer networks and has a wider application range.

2.1.2 Existing Pub/Sub Systems

There are some well-known protocols of topic-based pub/sub: JMS [Deakin 2015], AMQP [Vinoski 2006], MQTT [Banks and Gupta 2015] and so on.

JMS is a specification describing a common way of messaging for Java language. It supports two messaging models as shown in Figure 2.2. Pub/sub model is the pattern of topic-based pub/sub messaging. On the other, point to point (PTP) model is somewhat different. In PTP, there is only a single receiver for one message queue, even though there might be multiple senders. A receiver needs to read actively from the queue.

AMQP is an open standard protocol standardized by OASIS. It is fundamentally

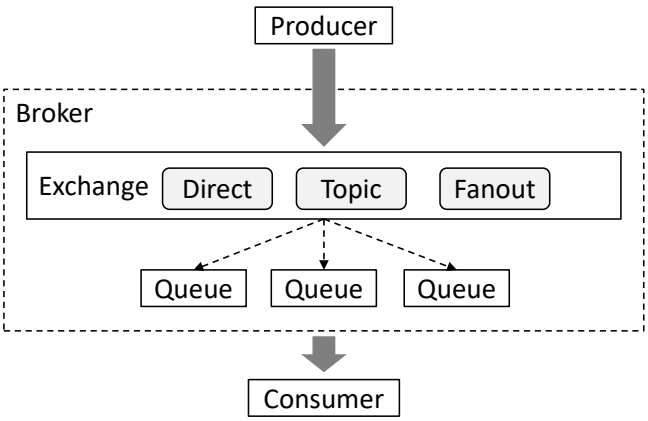

Figure 2.3: Message flow in AMQP.

based on the concept of message queues, but can realize various patterns by using a module called "exchange" (Figure 2.3). At first, a sender, called "producer" in AMQP, sends a message to an exchange. The exchange which receives the message is responsible for routing the message towards queues bound to it. The behavior of routing is determined by the type of the exchange. There are the following three types:

Direct Delivers the message to queues which have exactly the same label called "key" of the message.

Topic Delivers the message to queues which have exactly or partially match the key of the message.

Fanout Delivers the message to all queues bound to the exchange.

Eventually the messages in queues are consumed by receivers, called "consumer" in AMQP.

Regarding MQTT, we describe it later in this section.

The above protocols are implemented in many middleware, so-called Message-Oriented Middleware (MOM), such as ActiveMQ [Snyder et al. 2008], RabbitMQ [Videla and Williams 2012], MessageSight [Chen et al. 2014], etc. There are also implementations assuming specific use, e.g., Kafka [Kreps et al. 2012] is focusing on processing a large amount of log data.

In recent years, some cloud-based messaging services have appeared [Google Cloud pub/sub; Amazon Simple Notification Service]. These services are easy to use, but have a difficulty in handling data exhaust as described in Section 1. Redundant forwarding of worthless data to the cloud broker lowers the throughput, and the latency also tends to be higher compared to the edge-based architecture.

MQTT

MQTT [Banks and Gupta 2015] is a protocol of topic-based pub/sub, standardized by OASIS. It is known for lightweight design such as a minimum of two bytes header size. Below is an example flow of using MQTT.

1. A client X sends $CONNECT$ message to a broker. This establishes a connection between X and the broker.
2. X sends $SUBSCRIBE$ message to the broker. This message informs the topics of interest of X to the broker.
3. Another client sends $PUBLISH$ message to the broker, with specifying a topic. If the topic is included in the above topics of interest, this message is forwarded to X by the broker.
4. X sends $DISCONNECT$ message to the broker, to terminate the connection.

MQTT provides several useful functions for clients, such as "QoS", "Retain" and "Will".

QoS provides capability of configuring the level of delivery confirmation. A client and a broker try to confirm the delivery of a $PUBLISH$ message and resend it if needed, according to the QoS level. Three levels are defined: "At most once delivery", "at least once delivery", and "exactly once delivery".

Retain is for delivering a latest message in the past to a new subscriber. A $PUBLISH$ message has a flag of Retain. If the flag is set to *true*, a broker stores the message until a new $PUBLISH$ message whose Retain flag is *true* of the same topic arrives. This stored message will be forwarded to new subscribers of the topic.

Will enables to inform unexpected close of a connection. $CONNECT$ message has a flag of Will. If the Will Flag is set to *true*, a broker stores a Will message and Will

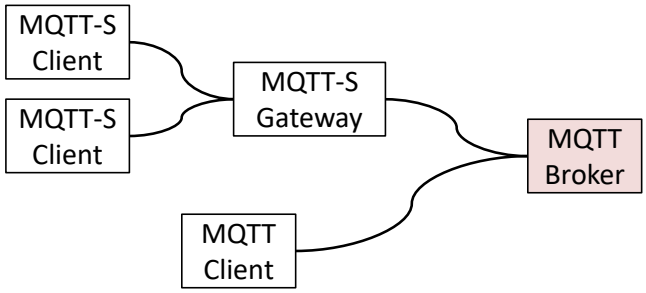

Figure 2.4: Architecture of MQTT-S.

topic which are also included in the *CONNECT* message. The Will message will be published from the broker, when it detects the connection with the client is unexpectedly closed.

In addition to the general MQTT protocol, MQTT-S (also called MQTT-SN) has been proposed as a specialized extension for wireless sensor networks [Hunkeler et al. 2008]. MQTT-S is oriented toward more lightweight design, and has the following features:

- It can be used with transport protocols other than TCP, such as UDP and ZigBee.
- The size of a PUBLISH message can be reduced by replacing a topic name to a two-byte long "topic ID".

MQTT-S assumes the architecture as shown in Figure 2.4. MQTT-S clients connect to an MQTT broker via an MQTT-S gateway. An MQTT-S gateway may or may not be integrated with an MQTT broker.

Such protocol is helpful to enable various small devices to participate in systems constructed with the edge-based architecture.

2.2 Structured Overlay Networks

A structured overlay network is a kind of overlay networks, i.e., a computer network constructed on top of another computer network (Figure 2.5). Each node has a routing table, and the neighborhood relations defined by the table form a structural topology.

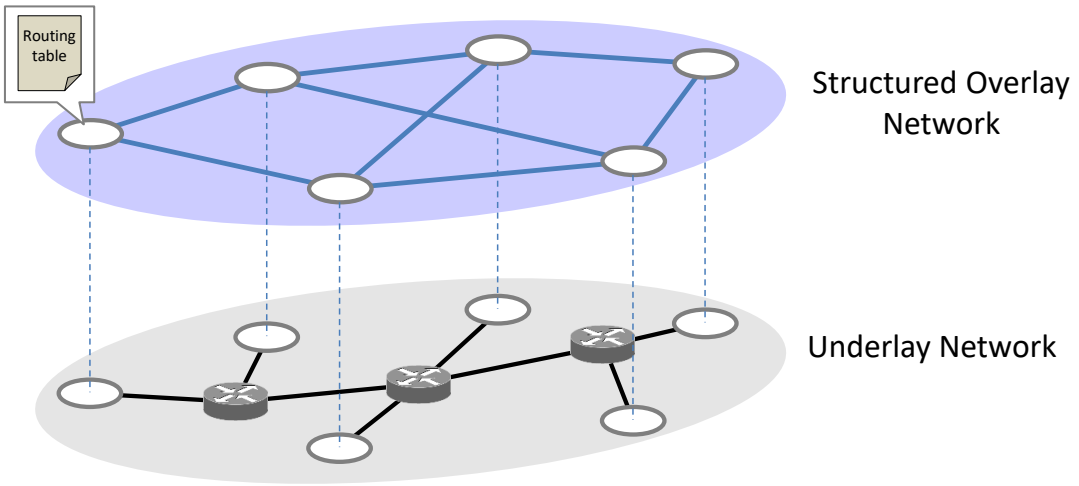

Figure 2.5: Structured overlay network.

One of the best known types of structured overlay networks is distributed hash table (DHT). DHT builds a hash table over multiple nodes. Namely, the place where each key-value pair is stored in is determined by the hash value of the key. Nodes have node IDs which are also hash values calculated from such as their IP addresses, and they are responsible for a part of the range of hash values according to their IDs.

By using cryptographic hash functions such as MD5 [Rivest 1992] and SHA-1 [FIPS PUB180-1 1995], keys and node IDs are mapped uniformly over the range of hash values as shown in Figure 2.6, so that the number of key-value pairs stored on each node is almost equalized.

2.2.1 Chord

Chord [Stoica et al. 2001] is one of the earliest algorithms of DHT. Chord uses a ring-style one-dimensional ID space. A key-value pair with a key k is assigned to the first node whose ID is equal to or follows $hash(k)$ in the ID space. This node is denoted as $successor(hash(k))$.

In the case that the size of ID space is 2^m, a node with an ID x has a routing table

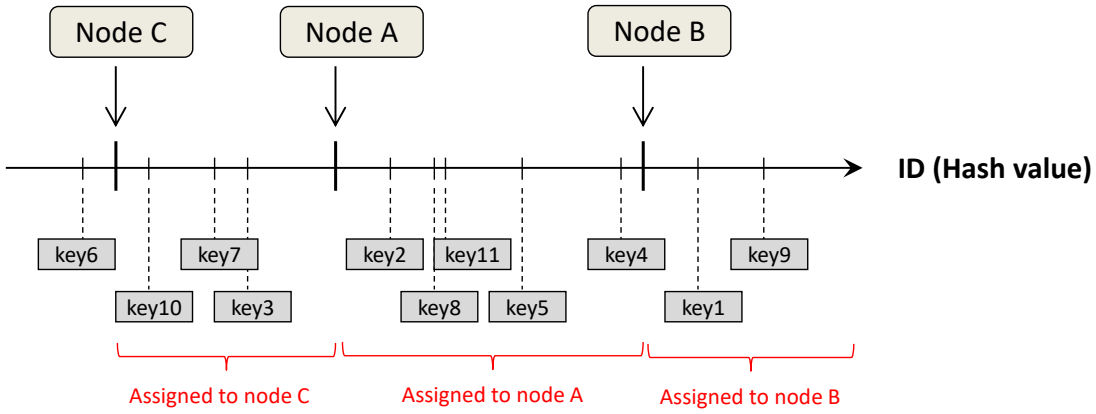

Figure 2.6: Example of ID space of DHT.

composed of the following entries:

$$\{successor(x + 2^i \bmod 2^m)\} \quad \text{where} \quad 0 \leq i \leq m - 1.$$

When the node wants to find a key-value pair with a key k, it issues a query to the node who has largest ID smaller than $hash(k)$ in the routing table. Note that "large" and "small" mean distant and nearby respectively in the direction of successors in the circular ID space.

By recursively forwarding the query in the same manner, arbitrary node can be found with $O(\log N)$ messages where N is the number of nodes. The size of the routing table on each node is also suppressed to $O(\log N)$.

For example, we assume an ID space with its size of 2^4 as shown in Figure 2.7. The routing table of the node of ID 0 is

$$\{successor(1), successor(2), successor(4), successor(8)\} = \{2, 6, 9\}.$$

When the node 0 want to find a key 12, it issues a query to the node 9 because it has largest ID smaller than 12 in the routing table. The routing table of the node of ID 9 is

$$\{successor(10), successor(11), successor(13), successor(1)\} = \{11, 13, 2\}.$$

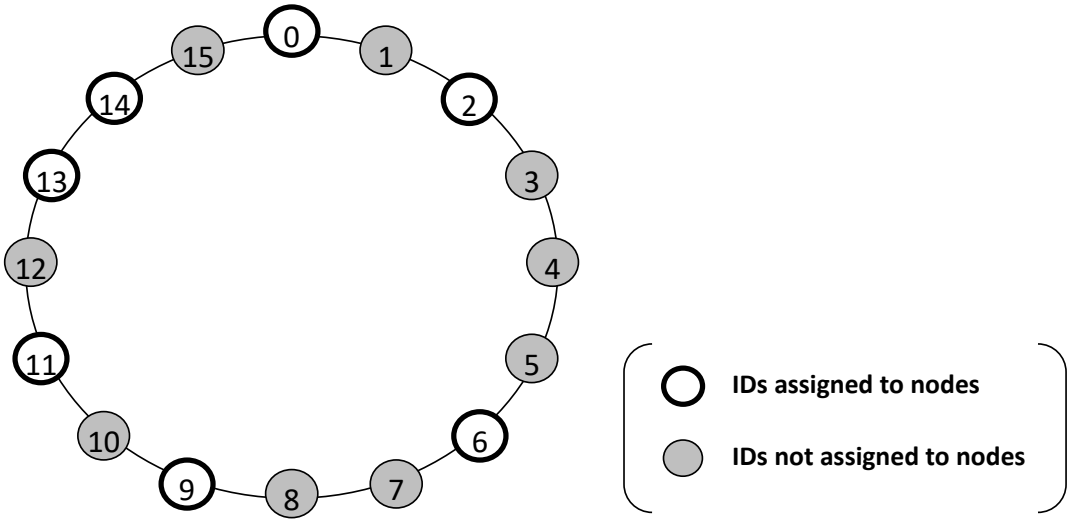

Figure 2.7: Example of ID space of Chord.

Node 9 forwards the query to the node 11. Finally, the successor of the node 11, i.e., node 13, is the node who stores the key-value pair with the key 12.

2.2.2 Pastry

Pastry [Rowstron and Druschel 2001] is an algorithm of DHT based on Plaxton's algorithm [Plaxton et al. 1999]. It uses an ID space of 128 bits, and IDs are expressed as sequences of numbers which have 2^b as their base. For example, if $b = 4$, IDs are expressed as 32-digit hexadecimal numbers. Pastry enables to find an arbitrary node with at most $\lceil \log_{2^b} N \rceil$ hops, where N is the number of nodes.

A routing table of a node consists of $\lceil \log_{2^b} N \rceil$ rows, and each of them has $2^b - 1$ entries. Entries in nth row have the same n-digit prefixes as the ID of the node and also have the different $n+1$th digit from that of the ID of the node. If there is no appropriate node, the corresponding entry becomes empty. Figure 2.8 shows an example of routing table where the ID space is 2^{16}, parameter b is 2, and the ID of the local node is 10233102.

Finding nodes in Pastry is processed by gradually extending the length of the common prefix of IDs. We denote the length of the common prefix between id_1 and id_2 as $L(id_1, id_2)$. We also denote the target ID as id_t. When a node x which has

	Routing Table (ID of local node: 10233102)			
0th row	02212102	1	22301203	31203203
1st row	0	11301233	12230203	13021022
2nd row	10031203	10132102	2	10323302
3rd row	10200230	10211302	10222302	3
4th row	10230322	10231000	10232121	3
5th row	10233001	1	10233232	
6th row	0		10233120	
7th row			2	

Figure 2.8: Example of routing table of Pastry.

an ID id_x receives the query, it selects a node y which has an ID id_y and satisfies $L(id_t, id_y) = L(id_t, id_x) + 1$ from its routing table. Subsequently, the node x forwards the query to the node y.

Figure 2.9 shows an example of finding nodes when ID space is 3-digit quaternary numbers. Note that the original design of Pastry uses a 128-bit ID space, but we assume a smaller ID space in this example for simplicity. In this example, the node S (ID: 312) issues a query of finding ID 312. It is processed as follows:

1. Node S checks the 0th row of its routing table, and finds the node A which has an ID beginning at 3.

2. Node A checks the 1st row of its routing table, and finds the node B which has an ID beginning at 31.

3. Node B checks the 2nd row of its routing table, and finds the node C which has an ID beginning at 312.

The size of routing tables in Pastry is approximately $\lceil \log_{2^b} N \rceil \cdot (2^b - 1)$, and the required number of hops for finding nodes is $\lceil \log_{2^b} N \rceil$.

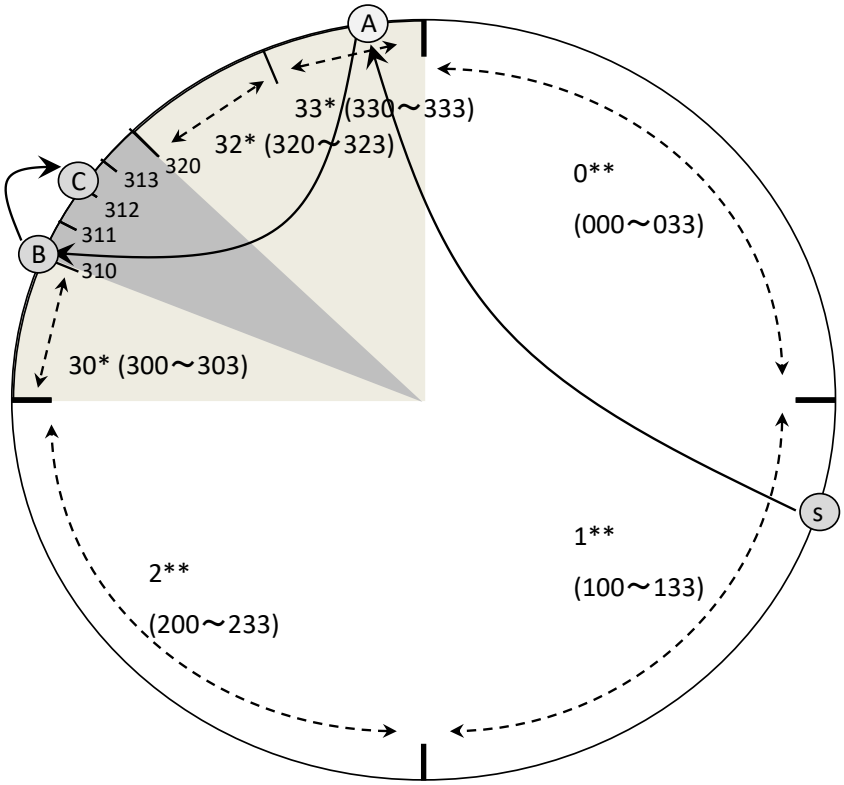

Figure 2.9: Pastry.

2.2.3 Other DHT algorithms and their extensions

Other than Chord and Pastry, many algorithms of DHT are proposed; CAN [Ratnasamy et al. 2001a] based on N-dimensional torus, Kademlia [Maymounkov and Mazieres 2002] based on binary tree, Koorde [Kaachoek and Karger 2003] based on De Bruijn Graph [de Bruijn and Erdős 1948], Symphony [Manku et al. 2003] based on small world network [Watts and Strogatz 1998], and so on. A common feature of most of these algorithms is that each node has dense information for nearby nodes and sparse information for distant nodes so that a query can be processed by gradually narrowing down the range including the target hash value.

There are some studies of extensions of DHT, such as query bundling mechanisms [Mizutani et al. 2012; Shudo 2017] and methods for handling partial match queries [Huebsch 2008; Zhuge and Feng 2008]. There are also some studies which try to have capabil-

ity of handling range queries, e.g., Prefix Hash Tree [Ramabhadran et al. 2004] and Mercury [Bharambe et al. 2004].

2.2.4 Skip Graph

Skip Graph is an algorithm of structured overlay networks providing the capability of handling range queries. Each node has a key and can issue a query by specifying a range or a value in the key space. Issued queries are delivered to nodes whose keys are included in the range or exactly matched with the value.

Skip Graph is based on skip list [Pugh 1990]. Skip list is a data structure which has hierarchical lists as shown in Figure 2.10. The lowest list, called Level 0 list, includes all elements in a sorted order. An element in Level i also exists in Level $i+1$ with a predefined probability. Finding an element starts from the first element in the topmost level and processed as follows:

1. Find the largest element in elements smaller than the target element in the topmost level.
2. If the corresponding element is not the target element, move one level down.
3. Repeat the above two steps until reaching the target element.

This algorithm provides $O(\log N)$ time to find an element where N is the number of elements.

Skip Graph composes a multiplex structure of skip list as depicted in Figure 2.11. Level 0 is a doubly linked list that consists of all nodes sorted in the order of keys. Each node has a random sequence in base b* called a membership vector, and composes a doubly linked list with nodes whose membership vectors have the same first i digits in level i.

When a node issues a query, the search process starts from the maximum level of the node. The query is forwarded among nodes in the same manner as the skip list, i.e., skips long distance at the higher level and gradually moves down to level 0.

The size of the routing table that each node must have is $\mathcal{O}(\log N)$ of the N participants, while the path length of forwarding queries is also $\mathcal{O}(\log N)$.

* In this dissertation, we consider the case of binary digits except when we explicitly mention other cases.

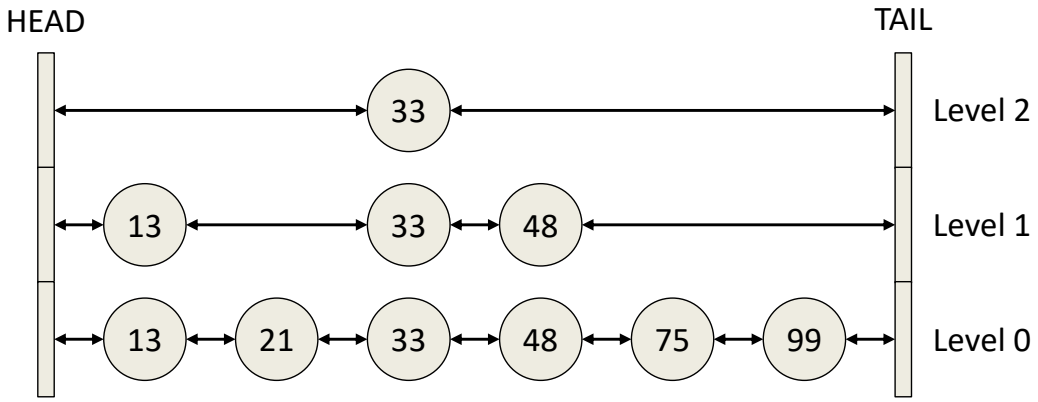

Figure 2.10: Example of Skip list

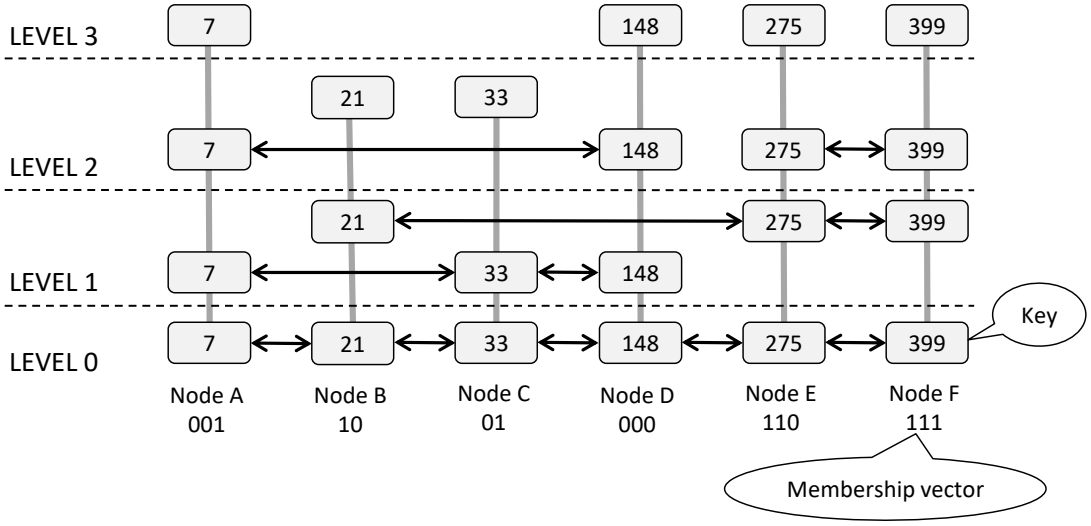

Figure 2.11: Example of Skip Graph

There are some studies of extensions of Skip Graph: FRT-Skip Graph [Hojo et al. 2016] is a method to constructing routing tables more flexibly with keeping the range queriable feature of Skip Graph. Self-Refining Skip Graph [Kawaguchi et al. 2016] adds a functionality of refining routing tables dynamically so that the path length becomes shorter.

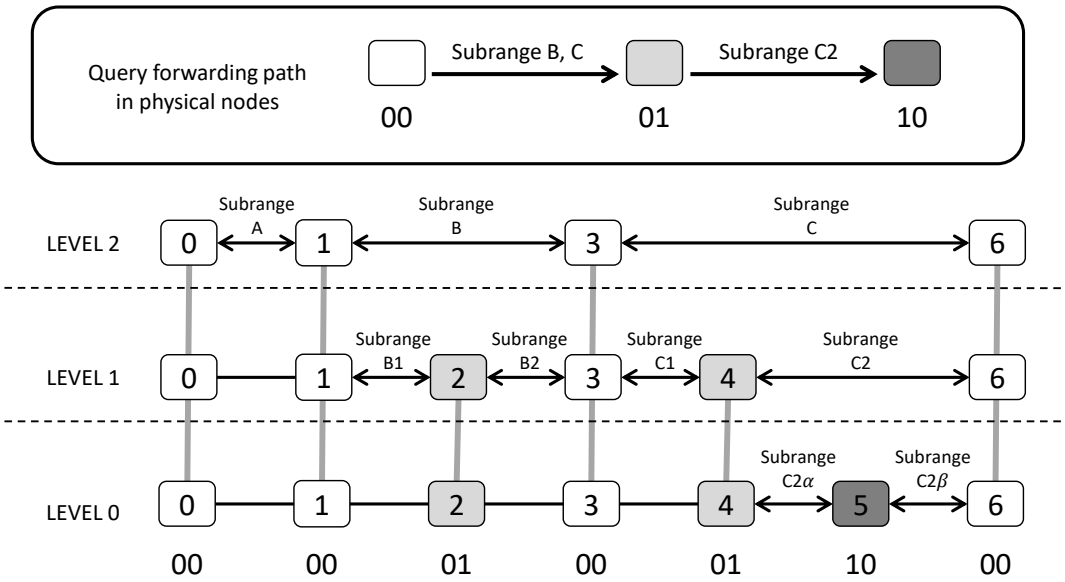

Figure 2.12: Query forwarding by multi-range forwarding

2.2.5 Multi-key Skip Graph

Multi-key Skip Graph [Konishi et al. 2008] is an expansion of Skip Graph, which enables nodes to possess multiple keys. Each node (hereafter, called a physical node) inserts its keys onto Skip Graph as virtual nodes. Virtual nodes created from the same physical node have an equivalent membership vector, namely membership vectors are unique to physical nodes.

If a query is forwarded among virtual nodes in the same way as normal Skip Graph, there is a possibility that the query passes through one physical node multiple times. To avoid an increase in hops by such possibility, Multi-key Skip Graph includes an efficient routing mechanism called multi-range forwarding.

In multi-range forwarding, a query with its target range R is forwarded as follows: when a virtual node whose key is outside R receives the query, the virtual node selects one from the virtual nodes of its physical node on the basis of proximity to R, and hands the query over to it. If the nearest is itself, it processes forwarding in the same way as normal Skip Graph. When a virtual node whose key is within R receives the query, the virtual node divides R into subranges by the keys of its physical node. The query is

duplicated, and forwarded with each subrange instead of R to other physical nodes.

Figure 2.12 shows an example. There are three physical nodes whose membership vectors are 00, 01, and 10. When the virtual node, whose key is 0, receives a query of target range $0 \leq key \leq 6$, the range is divided into three subranges: A, B, and C. Subrange B and C are forwarded to physical node 01 from 00, then are divided into subranges: B into B1 and B2, C into C1 and C2. Finally, subrange C2 is forwarded to physical node 10 from 01 and divided into C2α and C2β.

By these rules, each physical node having virtual nodes within a target range receives the same query only once. The path length of forwarding queries is $\mathcal{O}(\log N)$, where N is the number of physical nodes but not virtual nodes.

3

Distributed Pub/Sub Method Using Structured Overlay Networks

In this chapter, we illustrate our method of distributed pub/sub messaging using structured overlay networks.

Although there are some existing studies of topic-based pub/sub systems based on structured overlay networks, they have the problem of wasting network resources because of lacking adaptability to the "data exhaust". The problem contains two aspects. One is that each publisher node continues to forward messages to a relay node even if there are no subscribers. The other is that excessively large multicast trees are constructed for topics which have only a small number of subscribers.

Our method regulates publishers and subscribers of the same topic to compose connected subgraphs. This provides not only shrunk multicast trees but also enabling publishers to detect the absence of subscribers and suspend sending messages. We confirm the effectiveness of the proposed method quantitatively by simulation experiments.

3.1 Related Work

In this section, we provide an explanation of current methods of topic-based pub/sub systems using structured overlay networks [Castro et al. 2002; Zhuang et al. 2001; Ratnasamy et al. 2001b; Zhao et al. 2013]. These methods use DHTs in common and have been proposed as application layer multicast (ALM), where multicast groups correspond to topics in topic-based pub/sub.

3.1.1 Topic-based Pub/Sub Method using DHT

Scribe [Castro et al. 2002] is a method of topic-based pub/sub using Pastry [Rowstron and Druschel 2001]. In Scribe, nodes form a tree for every topic. Each topic has a unique ID computed from a topic name by using a hash function, e.g., SHA-256. A node responsible for the ID on the Pastry network becomes the root node of the tree of that topic. The root node is called the rendezvous point while the other nodes of the tree are called forwarders. A node that attempts to subscribe to a topic sends a **JOIN** message towards the rendezvous point, according to the routing protocol of Pastry described in Section 2.2.2. A node that receives the message adds the information of the sender node to its children table. If it had not been a forwarder of the topic before receiving the message, it forwards the **JOIN** message towards the rendezvous point. Therefore, the multicast path from the rendezvous point to the joining node is constructed in the reverse order of the routing path of Pastry as shown in Figure 3.1. Publishers of the topic send messages towards the rendezvous point whose address can be found by using the normal routing protocol of Pastry. Publishers can cache the address and send messages to the rendezvous point directly. Published messages are forwarded along the tree and delivered to all the corresponding subscribers.

Bayeux [Zhuang et al. 2001] is built on top of Tapestry [Zhao et al. 2004] and realizes topic-based pub/sub in a similar way to Scribe. The primary difference is that Bayeux uses the forward-path forwarding scheme, while Scribe uses the reverse-path forwarding scheme [Zhang and Hu 2003]. In Bayeux, a node attempting to subscribe to a topic sends a **JOIN** message towards the root node, and each intermediate node in the path from the joining node to the root node simply forwards the message. When the root node receives the message, it sends a **TREE** message towards the joining node. Each

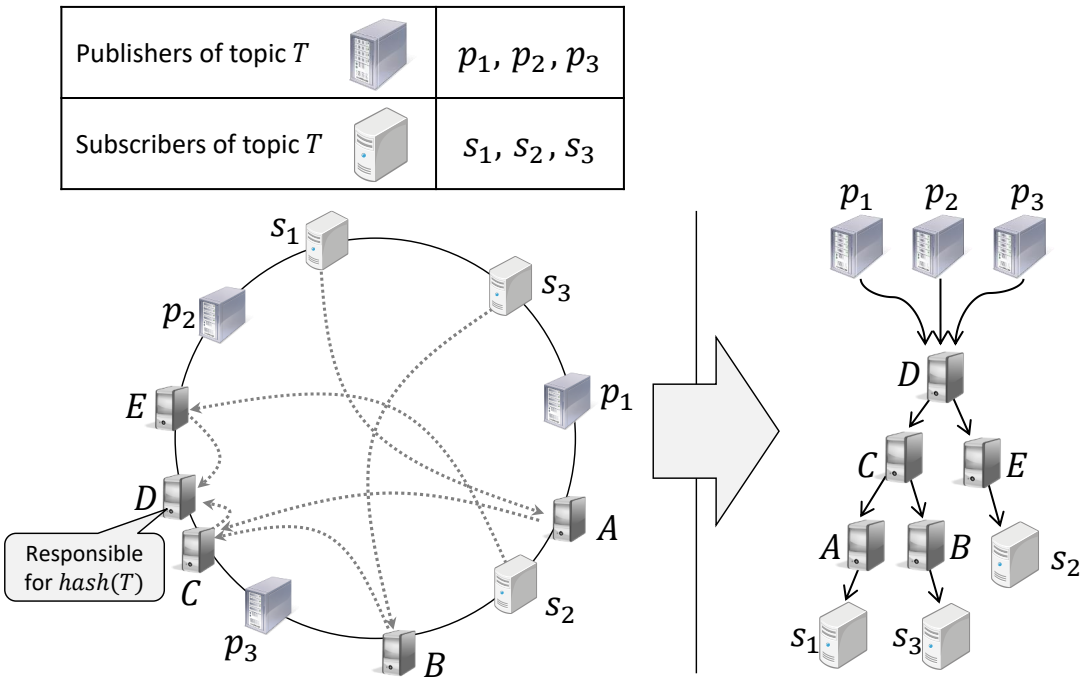

Figure 3.1: Topic-based pub/sub by Scribe.

node in the path from the root node to the joining node registers the joining node in its table.

CAN-MC [Ratnasamy et al. 2001b], built on top of CAN [Ratnasamy et al. 2001a], has presented a somewhat different style compared to the above two methods. CAN-MC consists of two types of CAN networks: the entire CAN and the mini CAN. The entire CAN is joined by all of the nodes and provides the function of looking up an introducer node, which is specific for each topic. The mini CAN is constructed for each topic independently and joined by the nodes of the topic. A published message is delivered by flooding over the corresponding mini CAN as follows:

- A publisher sends a message to all its neighbors.
- Each node receiving the message from its neighbor along dimension i forwards it to nodes as follows: the neighbors along dimension 1 to $(i-1)$, and those along dimension i in the opposite of the receiving direction.
- Each node does not forward the message along a particular dimension if it has al-

ready traversed at least half-way across the space from the publisher's coordinates along the dimension.

- Each node caches the sequence number of messages and does not process a duplicated message.

DYNATOPS [Zhao et al. 2013] is an approach to extend rendezvous-based pub/sub methods like Scribe or Bayeux by using following two dynamic mapping algorithms.

- Similarity-based user placement
- Broker network reconfiguration

The former is an algorithm which aims to map users with similar subscriptions to nearby brokers. It is useful for reducing the subscription management overhead at brokers. This algorithm can also be applied to our approach, but it is out of the scope of this dissertation. The latter is an algorithm for dynamic reconfiguration of overlay networks of brokers, which aims to reduce the unrelated relay brokers on topic routing trees. The goal of this algorithm is somewhat similar to our research which succeeds in eliminating unrelated relay brokers. One of the major differences is that DYNATOPS drives the reconfiguration process with overhead subsequent to the establishment of topic routing trees.

3.1.2 Inadequacy for Handling Data Exhaust

In the edge-based architecture, it is preferable that the pub/sub messaging works efficiently even when the number of subscribers is small or zero, because data exhaust has low value densities as described in Chapter 1. However, conventional methods have the following inefficiencies for handling data exhaust, though they achieve high scalability.

Forwarding Messages Towards Absent Subscribers

Conventional methods cannot suspend publishing even if there are no subscribers. In Scribe and Bayeux, publishers have no way to detect the absence of subscribers, and have to continue to constantly send messages towards the root node of the corresponding topic. Even in DYNATOPS, there is the same problem. In CAN-MC, nodes join the

mini CAN of the topic of interest without any distinctions between subscribers and publishers. As a result, each publisher is forced to continue to flood messages as long as there are other publishers.

Excessively Long Forwarding Path for Small Topic

Scribe and Bayeux construct a multicast tree for each topic. The path length from the root node to each subscriber is the same as the path length of finding other nodes in Pastry and Tapestry, namely $\mathcal{O}(\log N)$ where N is the number of entire nodes. Because this length does not depend on the number of nodes that are of the topic, published messages are forced to be forwarded along the excessively long path even if there are only few subscribers. This wastes network resources and increases the delay time of delivery.

Figure 3.2 illustrates this problem. As a primitive consideration, a heavy load is applied to the root node if it undertakes forwarding messages to all corresponding subscribers, as shown in the upper left of the figure. Scribe and Bayeux construct multicast trees to avoid this heavy load, as shown in the lower left of the figure. However, when there are only few subscribers, these methods force messages to be forwarded along the trees that have the same depth as the case of numerous subscribers (see the lower right of the figure). Regarding the case of few subscribers, the zero-hop delivery described in the upper right of the figure can forward more economically. It is also thought that most of the topics usually have few subscribers because of the low value density of data exhaust.

Thus, an efficient method is preferred, which achieves both economical forwarding for few subscribers and load distribution for numerous subscribers.

3.2 Topic-based Pub/Sub Method Using Skip Graph

We first clarify the requirements for overcoming the problems described in Section 3.1.2.

- To prevent the forwarding towards absent subscribers, it is required that publishers can detect the switching between corresponding subscribers' absence and presence.

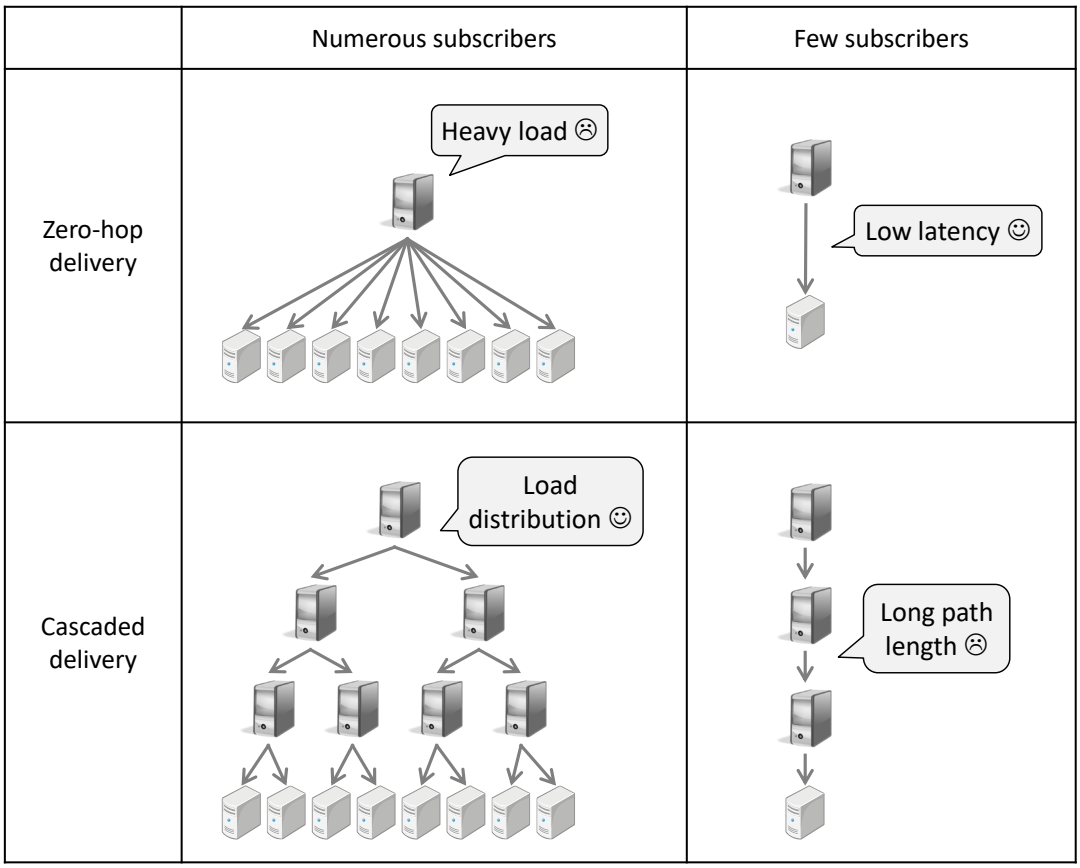

Figure 3.2: Difference in the load and latency of forwarding by the number of subscribers.

- To prevent the excessively long forwarding path, it is required to shorten the path length for a small number of subscribers, while maintaining efficient load distribution of dissemination for a large number of subscribers.

Each requirement can be met in simple ways, such as broadcasting queries to determine the presence of subscribers, switching delivery mechanisms as shown in Figure 3.2 for each topic, and so on. However these ways lack the global perspective and lead to other inefficiencies, e.g., negative effect on scalability.

In this section, we propose a method for constructing overlay networks that satisfy the above requirements by using Skip Graph [Aspnes and Shah 2007]. Subsequently, we describe the mechanism of suspending publishing on the constructed overlay networks.

3.2.1 Overview

As mentioned in Section 2.2.4, nodes in Skip Graph are sorted by keys, i.e., the set of keys must be totally ordered set. In this sections, we use the notation below for the key of ith node:

$$key_i = \{subkey_j \mid j = 0, 1, 2, \ldots, k\} \quad \text{where} \quad k \geq 0$$

The order of keys is determined by the order of $subkey_j$. That with a smaller value of j has higher priority to decide the order. For instance, if $k = 2$ and each $subkey_j$ is a character string whose order is lexicographic order, we can use a character string "$subkey_0$_$subkey_1$_$subkey_2$" as the key.

We first assume that each node possesses the names of topics of interest as a subscriber or a publisher. By using the names as $subkey_0$ and constructing Multi-key Skip Graph, topic-based pub/sub is possible. Publishers can deliver messages to subscribers by range queries of Multi-key Skip Graph.

We also use the type of nodes, i.e., a subscriber or a publisher, as $subkey_1$. Accordingly, in addition to the fact that the units of every topic are sorted, units of every node type (subscriber or publisher) are also sorted inside the topic units at level 0 as shown in Figure 3.3. In this figure, $G^S_{t_i}$ denotes a subgraph induced by nodes which are subscribers of the topic t_i, whereas $G^P_{t_i}$ denotes a subgraph induced by nodes which are publishers*. $rp(t_i)$ denotes the rendezvous point, described later in Section 3.2.1.

Furthermore, to make keys being totally ordered set, keys of nodes need to have $subkey_2$ which are unique among nodes. For example, this is possible by using the IP address or the MAC address of each node.

Note that nodes possessing multiple combinations of topics and types can participate in this overlay network by having multiple virtual nodes corresponding to the combinations. On the other hand, nodes possessing no combinations of them can participate in this overlay network by having a virtual node with predefined key like "default_key_< *random number* >". This key must be defined so that it will not conflict with other keys.

Since the proposed method utilizes the topology of Skip Graph, we can easily add or remove nodes by using the join/leave mechanisms of Skip Graph. This feature con-

* Strict definitions are described in Section 3.3.2.

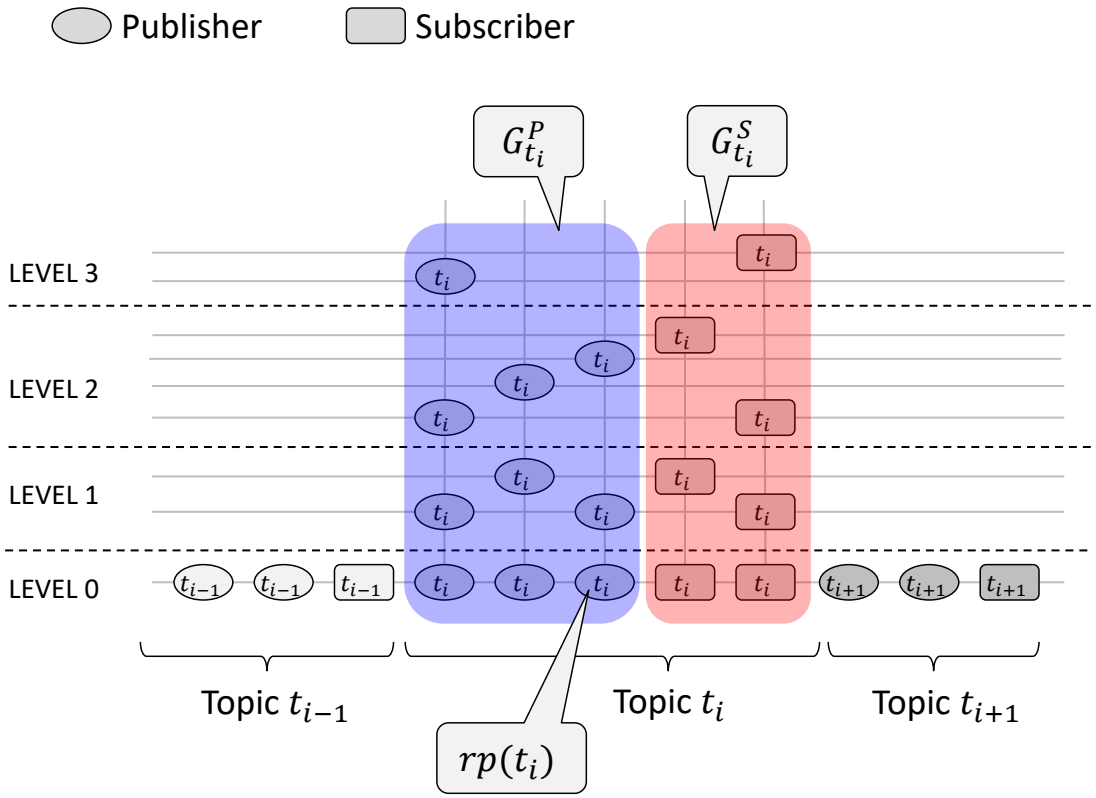

Figure 3.3: Ordered relation of nodes in the proposed method

tributes to that a service vendor using our method can increase or decrease the number of edge brokers in response to the changes in the service scale.

Publish, Subscribe, Unsubscribe

When a publisher sends a message to a topic t, the range search mechanism of Multi-key Skip Graph is used with the target range of $G^S_{t_i}$. The process of subscribing/unsubscribing is possible by the insertion/deletion mechanisms of virtual nodes in Multi-key Skip Graph (essentially similar to those of Skip Graph).

Definition of Rendezvous Point

For $\forall t \in T$, it is ensured that there exists a unique publisher contiguous to G^S_t at level 0 as long as one or more publishers of t exist. We call this publisher a "rendezvous

point" with an expression of $rp(t)$. For example, the position of the rendezvous point of topic t_i is illustrated in Figure 3.3. $rp(t)$ can conclude whether subscribers are absent without any meta information about topic t. Specifically, if the only neighbor v of $rp(t)$ at level 0 in the direction of G_t^S satisfies

$$v \in Sub(t),$$

there are one or more subscribers. Otherwise, there are no subscribers.

When a new publisher is inserted between $rp(t)$ and G_t^S, the new publisher takes the place of the rendezvous point thereafter. On the other hand, when an existing $rp(t)$ leaves topic t and there are other publishers, the neighbor of $rp(t)$ at level 0 in the direction of G_t^P takes over the position of the rendezvous point.

Suspending and Resuming

Suspending and resuming according to the switching between subscribers' absence and presence is possible by the rendezvous point, which is responsible for detecting the switching and notifying other publishers. Figure 3.4 and Figure 3.5 show flow charts regarding the behavior of $rp(t)$.

When all subscribers of topic t leave, $rp(t)$ can detect it passively by using a handler which catches the update of routing tables of Multi-key Skip Graph. When $rp(t)$ detects the absence of subscribers, it sends a signal dictating suspension to publishers, by using the range search mechanism with the target range of G_t^P, as described in Figure 3.4.

Conversely, when new subscribers appear in topic t, which has had no subscribers, $rp(t)$ can detect it passively in the same way and is responsible for sending a signal dictating resuming to publishers (see Figure 3.5).

Inserting Publishers

Newly joining publishers need to conclude whether they should start publishing immediately after finishing participation[†].

[†] Concerning a node which is both a subscriber and publisher of a topic, such node just needs to insert a virtual node only into G_t^S and continue publishing towards G_t^S. The reason is that there exist both subscribers and publishers as long as the node itself is alive. Therefore, such node can be irrelevant to suspending/resuming mechanisms.

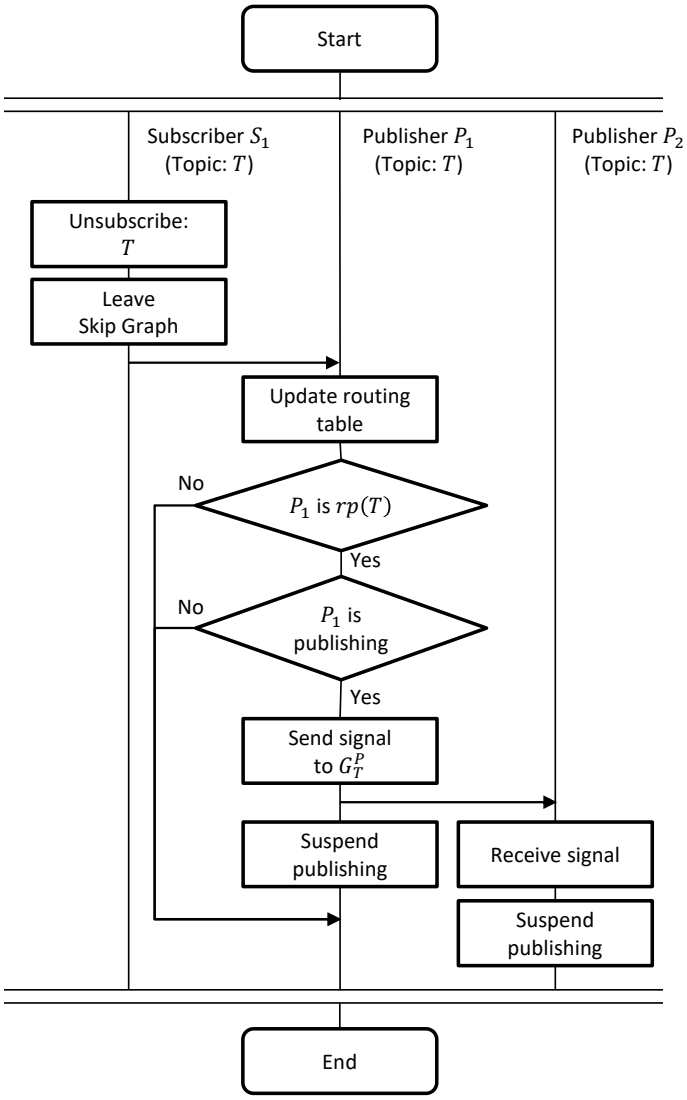

Figure 3.4: Flow chart with respect to switching publishers' behavior (Suspending).

In case that there has been any publisher of the corresponding topic, the joining publisher is certain to exchange messages with at least one existing publisher in the process of inserting in Multi-key Skip Graph. In the proposed method, information about the suspending status is piggy-backed on the messages, and the joining publisher determines the correct status by checking it.

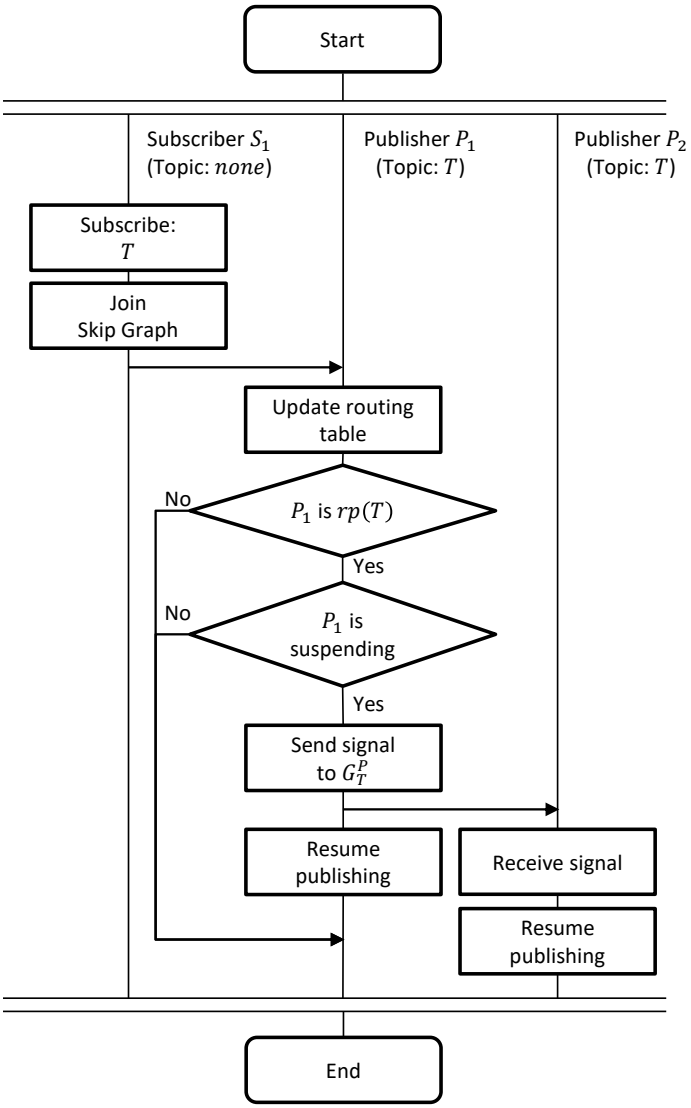

Figure 3.5: Flow chart with respect to switching publishers' behavior (Resuming).

If there were no publishers, the joining publisher will be the rendezvous point. Hence, it can determine the correct status by itself after finishing insertion.

Figure 3.6 is a flow chart regarding node insertions including the above mechanisms.

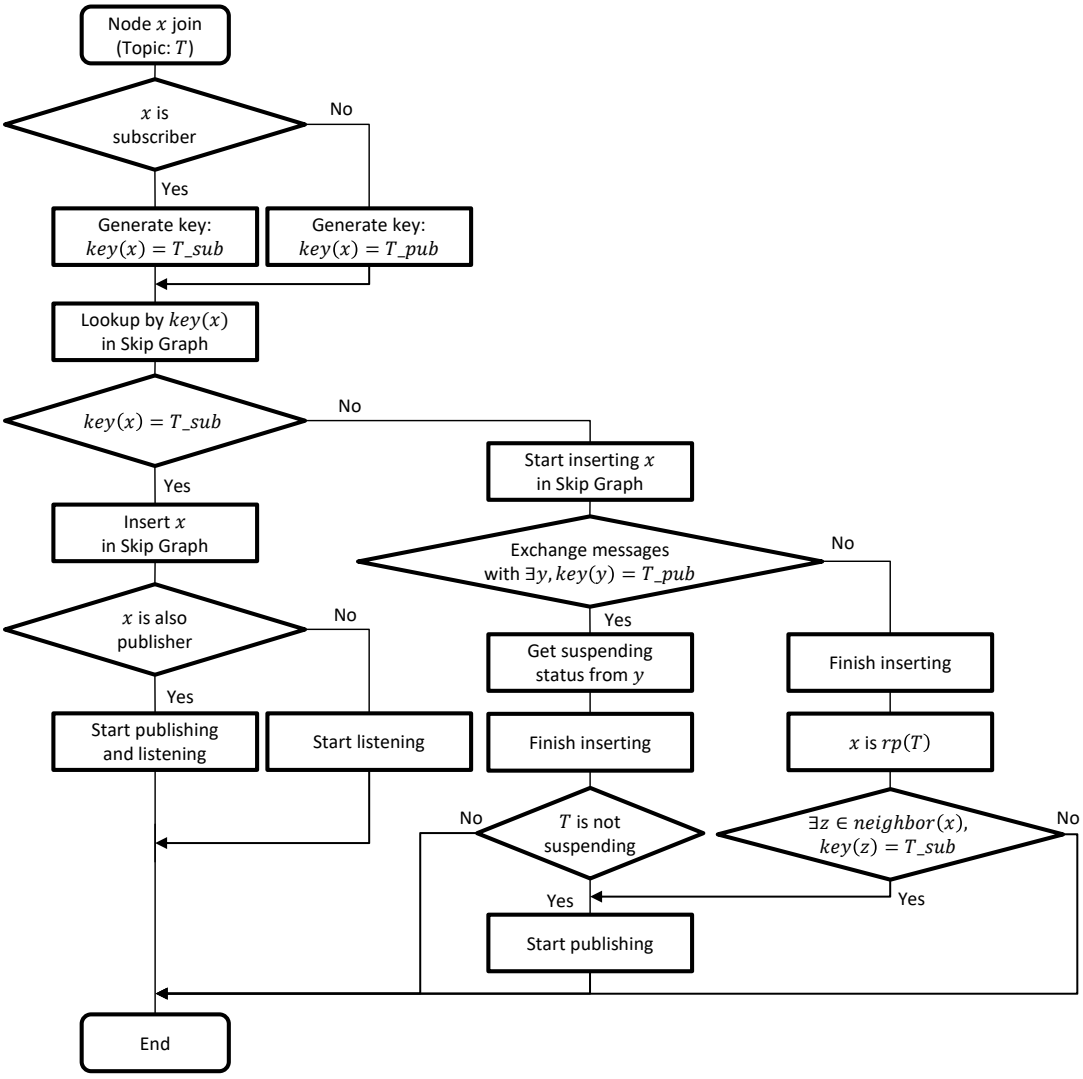

Figure 3.6: Flow chart of node insertion.

Eliminating Inconsistencies

In the proposed method, two types of inconsistencies described below can occur by the undelivered signals from rendezvous points caused by the sudden disappearance of nodes on the notifying path.

1. There exists a publisher continuing to publish even if there are no subscribers.
2. There exists a publisher suspending even if there are subscribers.

Table 3.1: Analytical comparison with existing methods.

	Suspend publish	Path length	Storage cost
Proposed	✓	$\mathcal{O}(\log(pub_t + sub_t))$	$\mathcal{O}(\frac{pub_t+sub_t}{N} M \log N)$
Scribe/Bayeux	✗	$\mathcal{O}(\log N)$	$\mathcal{O}(\frac{pub_t}{N} M + \frac{sub_t}{N} M \log N)$
CAN-MC	✗	$\mathcal{O}(d(pub_t + sub_t)^{1/d})$	$\mathcal{O}(\frac{pub_t+sub_t}{N} Md + d)$

These inconsistencies can occur on every publisher, except for rendezvous points. We describe the solutions for the inconsistencies.

In 1, a published message from a publisher of topic t is certain to pass through $rp(t)$ as long as there are no subscribers. When $rp(t)$ receives a message from other publishers during suspend, $rp(t)$ checks the absence of subscribers and sends a signal dictating suspension to the source publisher.

In 2, on the other hand, each publisher that is suspending actively confirms the status concerning the suspension of the neighbor at level 0 by periodically sending a dedicated message. As a result of the confirmation, if there is a conflict between the status of the neighbor and itself, the publisher sends a reporting message towards $rp(t)$. When $rp(t)$ receives the report, it checks the presence of subscribers and sends a signal dictating resuming to G_t^P.

3.2.2 Analytical Assessment

We give an analytical assessment of the proposed method in comparison with the methods described in Section 3.1.1[‡]. We assume the following notations: M denotes the number of topics, N denotes the number of nodes, pub_t denotes the number of publishers per topic, sub_t denotes the number of subscribers per topic, and d denotes the number of dimensions in CAN. To simplify, we also assume that each node is not both a subscriber and publisher of the same topic.

Table 3.1 shows the comparison of the methods. "Suspend publish" denotes the ability to suspend publish messages. Only the proposed method has this ability.

"Path length" denotes the maximum length of paths from publishers to subscribers.

[‡] In this section, we refer to Scribe, Bayeux, and CAN-MC. As for DYNATOPS, the fundamental property is considered to be same as Scribe and Bayeux.

The proposed method needs

$$\mathcal{O}(\log(pub_t + sub_t)),$$

because a published message of topic t is forwarded over the subgraph which consists of G_t^S and G_t^P. Scribe/Bayeux uses lookup paths of DHTs, so the path length is

$$\mathcal{O}(\log N).$$

CAN-MC requires

$$\mathcal{O}(d(pub_t + sub_t)^{1/d}),$$

due to flooding over the corresponding mini CAN. Because the path length of the proposed method does not depend on N unlike Scribe/Bayeux, it can reduce the consumption of network resources and the delay time of delivery, especially regarding topics having a small number of participants. CAN-MC also excludes N, and its path length depends on d which can adjust the tradeoff between the path length and storage cost.

"Storage cost" denotes the average size of routing tables of all nodes. This cost affects the consumption of memory and the maintenance overhead on each node. With the proposed method, each publisher or subscriber is inserted onto Multi-key Skip Graph as a virtual node which must have

$$\mathcal{O}(\log N)$$

neighbors in the routing table. Here, the total number of virtual nodes is

$$(pub_t + sub_t)M,$$

which means that each physical node has

$$\frac{(pub_t + sub_t)M}{N}$$

virtual nodes on average. Thus, the average size of routing tables is

$$\mathcal{O}(\frac{pub_t + sub_t}{N} M \log N).$$

Regarding Scribe/Bayeux, each subscriber forces intermediate nodes on the forwarding path to possess children tables. This storage cost is

$$\mathcal{O}(\frac{sub_t}{N} M \log N)$$

on average. Besides this, each publisher caches the root node of the corresponding topic, then the average cost is

$$\mathcal{O}(\frac{pub_t}{N} M).$$

Accordingly, the total average cost is

$$\mathcal{O}(\frac{pub_t}{N} M + \frac{sub_t}{N} M \log N).$$

With CAN-MC, each publisher or subscriber is inserted onto mini CAN with the cost of

$$\mathcal{O}(d),$$

so the average cost is

$$\mathcal{O}(\frac{pub_t + sub_t}{N} Md).$$

Each node also composes the entire CAN, thus

$$\mathcal{O}(\frac{pub_t + sub_t}{N} Md + d)$$

is required as a whole. The proposed method requires slightly large cost compared to Scribe/Bayeux, but is not extremely inferior. Concerning CAN-MC, the cost depends on d.

There is another viewpoint that should be discussed. Node(s) that are responsible for the storage cost are different between the methods. Regarding the proposed method and CAN-MC, when a subscriber or publisher is added, the joining subscriber or publisher itself is responsible for the storage cost. Some other nodes are forced to update routing tables, but no one is basically forced to increase the size of its routing table except for the joining node. On the other hand, with Scribe or Bayeux, the joining of a subscriber or publisher forces intermediate nodes on the forwarding path to take responsibility of the storage cost. This seems to be preferable from the viewpoint of load distribution,

but it also means that each node cannot predict its forwarding load. In other words, the fact that multiple nodes are responsible for storage cost may lead to inconvenience in terms of the load predictability. The details will be discussed later with experimental results, in Section 3.4.3.

3.3 Desirable Design of Overlay Networks

In this section, we give discussions about the desirable design of overlay networks for distributed pub/sub systems, based on the idea of the proposed method.

3.3.1 Relay-free Property

A property called "relay-free", also called "Topic-connected Overlay", has been proposed as an effective design of overlay networks [Chockler et al. 2007]. It is primarily discussed in studies based on unstructured overlay networks [Setty et al. 2012]. The definition is as follows:

> Given a set of nodes V and a set of topics T, we define sets of nodes $Pub(t) \subseteq V$ and $Sub(t) \subseteq V$ with a topic $t \in T$ as input. $Pub(t)$ is a set of nodes which are publishers of a topic t, whereas $Sub(t)$ is a set of nodes which are subscribers of a topic t. We also define a set of nodes $Int(t) \subseteq V$ as follows:
>
> $$Int(t) = Pub(t) \cup Sub(t).$$
>
> Given an overlay network:
>
> $$G = (V, E) \qquad \text{where} \quad E \subseteq V \times V,$$
>
> G is relay-free if a subgraph
>
> $$G_t = (V_t, E_t) \qquad \text{where} \quad V_t = Int(t) \quad \text{and} \quad E_t = V_t \times V_t \subseteq E$$
>
> is connected for all $t \in T$.

In overlay networks that satisfy the relay-free property, a published message is forwarded only between nodes that are interested in the corresponding topic. It is expected

that this property can contribute to shortening the path length for topics that have a small number of subscribers, because the diameter of the subgraph corresponding to each topic should become shorter in response to the decrease in subscribers.

The proposed method is relay-free, because it is ensured that subscribers and publishers joining the same topic are contiguous at level 0 in Skip Graph. Out of the existing methods explained in Section 3.1.1, CAN-MC also satisfies this property.

3.3.2 Strong Relay-free Property

Satisfying the relay-free property provides suitability for shortening path length, but it is still difficult for publishers to determine whether there is any corresponding subscriber. By introducing the distinction between subscribers and publishers into the relay-free property, we can express the essentials of the ability to suspend publish messages in the proposed method. Accordingly, we newly define a desirable property called "strong relay-free" as an expansion of the relay-free property.

The definition is as follows, where $T, Pub(t), Sub(t), G, E, G_t$ have the same meanings as above.

> An overlay network G is strong relay-free if all the following three conditions are satisfied for all $t \in T$:
>
> - A subgraph
>
> $$G_t^S = (V_t^S, E_t^S) \quad \text{where } V_t^S = Sub(t) \text{ and } E_t^S = V_t^S \times V_t^S \subseteq E$$
>
> is connected.
> - A subgraph
>
> $$G_t^P = (V_t^P, E_t^P) \quad \text{where } V_t^P = Pub(t) \text{ and } E_t^P = V_t^P \times V_t^P \subseteq E$$
>
> is connected.
> - A subgraph G_t is connected.

In the overlay network constructed by the proposed method, subscribers and publishers of each topic are contiguous respectively at level 0, and the subgraph of subscribers

and that of publishers are also contiguous. Thus, it satisfies the strong relay-free property.

In overlay networks satisfying the strong relay-free property, subscribers of each topic compose a connected subgraph. This means that the presence or absence of subscribers is synonymous with that of one subgraph. This is suitable for detecting the absence of subscribers under the constraint that each publisher has information only on its neighbors. Specifically, a publisher at the connection boundary between G_t^S and G_t^P seems possible to conclude whether subscribers are absent by checking only its neighbors. Furthermore, publishers of each topic also compose a connected subgraph, so one can easily disseminate the information about the absence of subscribers to others.

3.4 Evaluation

We evaluated our method through experiments with a simulation program implemented in Java. This section gives the details of each experiment and its results.

As mentioned in Section 1, we are aiming to provide techniques for the distributed pub/sub middleware which requires high throughput. Therefore, we evaluate the proposed method from two viewpoints having a large influence on the throughput: the number of messages exchanged on the overlay network, and the length of the forwarding path. The less messages required for processing a one publish message, the more messages per unit time (i.e., the throughput) we can expect that the whole distributed system composed by the middleware can handle. Regarding this relation between the required number of messages and the throughput, we also discuss later in Section 5.6. The length of the forwarding path is also important because it directly affects the number of messages exchanged on the overlay network. In addition, we also evaluate regarding the load of each node by measuring the correlation between the number of sending/receiving and forwarding messages, and the size of routing tables.

We chose Scribe as the comparison target in these experiments, mainly because its idea can be applied on top of any DHTs. Skip Graph can be used to construct a DHT by using a kind of routing which is referred to as "Routing by Numeric ID" in SkipNet [Harvey et al. 2003]. Indeed, this type of DHT is implemented by PIAX [Teranishi 2009]. Using a DHT on top of Skip Graph is convenient for harmonizing the experimen-

tal conditions such as the size of routing tables with the proposed method[§]. Therefore, we implemented Scribe on top of Skip Graph in the simulation program.

Note that our experiments were aimed to show essential tendencies because the actual performance is affected by parameters, e.g., the radix of membership vectors can adjust the tradeoff between the path length and size of routing tables.

3.4.1 Number of Messages Associated with Publishing

To confirm the ability to suspend publishing, we measured the number of forwarded messages on the overlay network with several numbers of subscribers including zero. The simulator generated $100,000$ nodes and constructed an overlay network regarding the proposed method and Scribe respectively.

We determined this number of nodes by considering a rough estimation of large scale systems: The largest services today have billions of users. If we assume each user send or receive one message per second, the required throughput is billions of messages per second. As shown in Section 5.6.1, existing products of the broker of a lightweight protocol can handle tens of thousands of messages per second. Therefore, we set $100,000$ nodes to compose the overlay network.

We set a topic with the following two patterns:

- A topic has 100 publishers and $1,000$ subscribers.
- A topic has 10 publishers and $1,000$ subscribers.

The simulator made subscribers unsubscribe in turns. At the timing of that the number of subscribers matches $1,000$, 100, 10 and 0, the simulator forced publishers of the topic to publish a message and counted the number of messages forwarded on the overlay network.

Figure 3.7 and Figure 3.8 shows the results of the averages of five repeated measurements. The term "p/t" in them denotes the number of publishers per topic. From

[§] Bayeux can also be built on top of any DHTs, unlike that CAN-MC is specialized for using CAN. But Bayeux has almost the same characteristics as Scribe from the viewpoint of the experiments described in this section, i.e., the number of messages, the length of the forwarding path and the correlation between the number of sending/receiving and forwarding messages. Therefore, we have chosen Scribe which was proposed later than Bayeux.

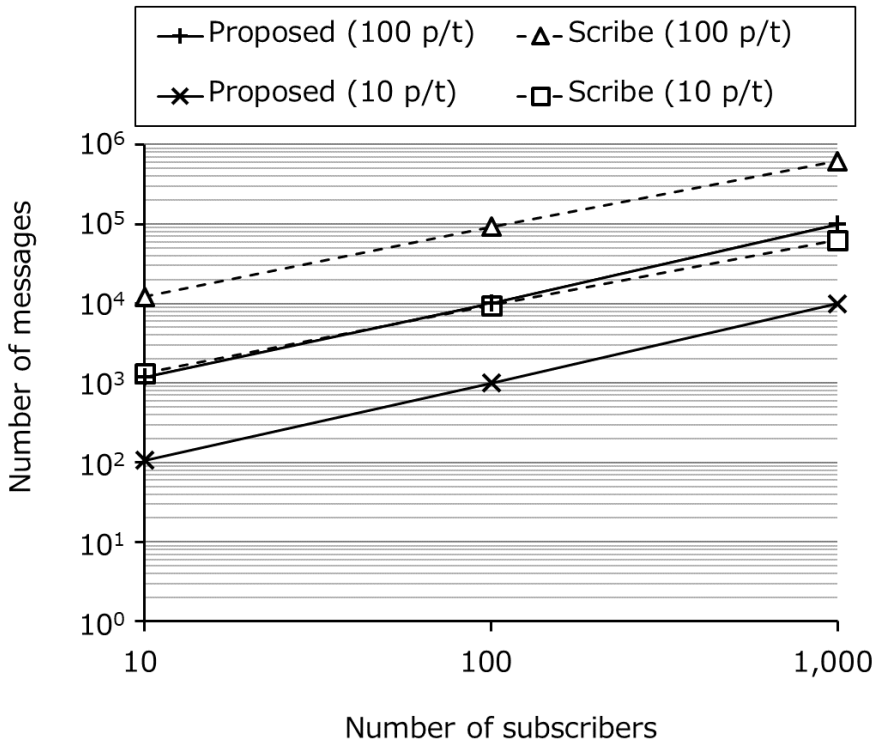

Figure 3.7: Number of messages associated with publishing.
(Number of subscribers is more than 0)

Figure 3.7, the number of messages becomes large according to the number of publishers or subscribers in both methods. The number of messages of the proposed method is smaller by approximately one digit than Scribe. This result is due to the difference of the length of forwarding path. In the proposed method, the length depends on the number of participants of the corresponding topic. On the other hand, the length in Scribe depends on the number of whole nodes, thus longer paths which cause a lot of messages are constructed. Experiments for confirming the difference of the path length will be described in Section 3.4.2.

Figure 3.8 is in the case of the number of subscribers is 0, and it indicates the number of messages drops to 0 regarding both patterns of the proposed method. Regarding Scribe, messages are forwarded even if the number of subscribers is 0.

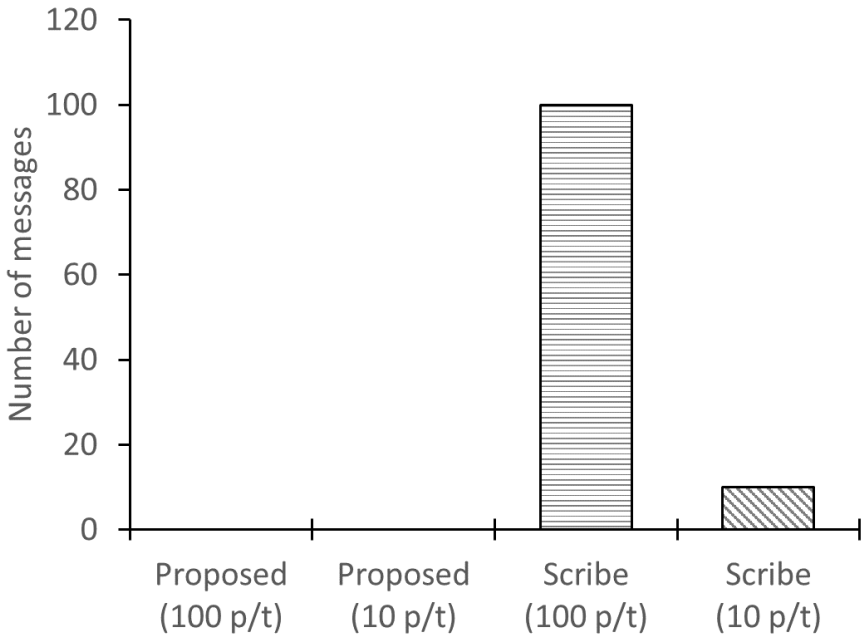

Figure 3.8: Number of messages associated with publishing. (No subscriber)

Table 3.2: Patterns of the experiment.

	pub_t	sub_t	Number of total nodes
α	10	990	$1,000$ or $10,000$ or $100,000$
β	500	500	$1,000$ or $10,000$ or $100,000$
γ	990	10	$1,000$ or $10,000$ or $100,000$
δ	1	9	$1,000$ or $10,000$ or $100,000$
ϵ	5	5	$1,000$ or $10,000$ or $100,000$
ζ	9	1	$1,000$ or $10,000$ or $100,000$

3.4.2 Length of Forwarding Path of Publishing

We also evaluated the effectiveness against gratuitous forwarding mentioned in Section 3.1.2. In this experiment, we calculated the average length of paths from each publisher to each subscriber. Here pub_t and sub_t denote the same as described in Section 3.2.2.

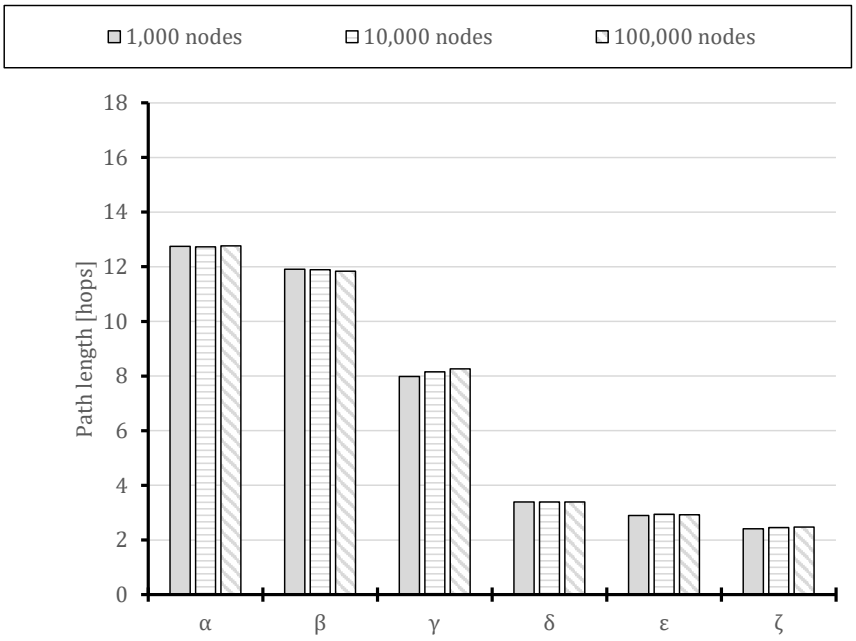

Figure 3.9: Length of forwarding path of publishing (Proposed method).

In this experiment, every topic was the same size, i.e., the sum of the number of publishers and subscribers was equivalent. We assumed two topic-sizes: small and large. We also assumed three combinations of pub_t and sub_t: $pub_t < sub_t$, $pub_t = sub_t$, and $pub_t > sub_t$. Thus, we set six patterns in total, as listed in Table 3.2. Each pattern had three different node amounts, $1,000$, $10,000$ and $100,000$. The number of topics in each pattern was keyed to the number of nodes, i.e., it can be obtained by dividing "Number of total nodes" by the sum of pub_t and sub_t. For example, the number of topics in pattern α was 10 when the number of nodes was $10,000$.

The simulator constructed overlay networks for every pattern and calculated the average length of the forwarding path from a publisher to every corresponding subscriber for all publishers.

Figure 3.9 and Figure 3.10 show the results. Regarding the proposed method, Figure 3.9 illustrates that the path length was not affected by the total number of nodes and was decreased in response to the reduction of the size of topics. This means the proposed method has high scalability for the increase in the total number of nodes and can prevent gratuitous forwarding. On the other hand, the results for Scribe shown in Figure 3.10

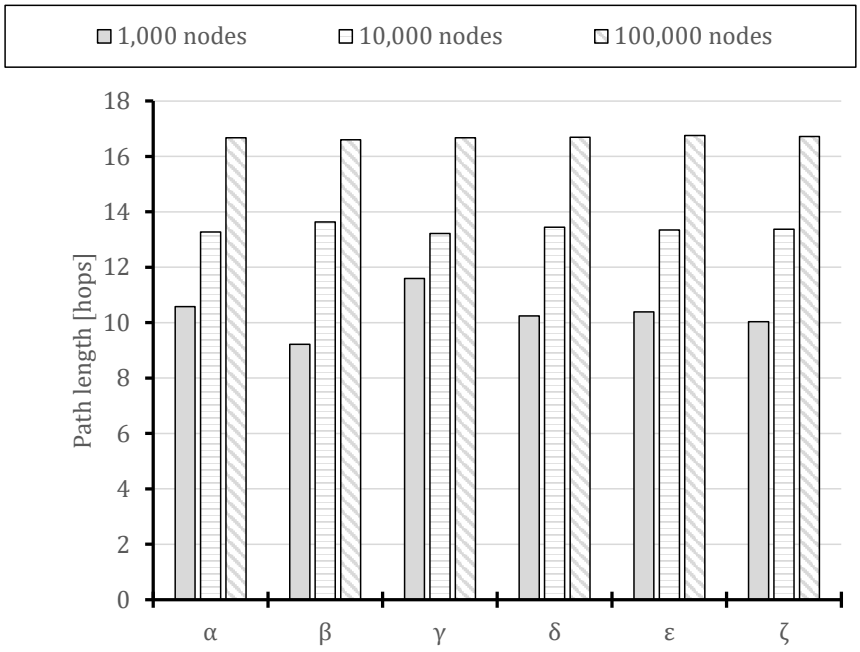

Figure 3.10: Length of forwarding path of publishing (Scribe).

indicate that the path length is not affected by the size of topics, which causes gratuitous forwarding.

For example, when focusing on patterns of δ, ϵ and ζ with $100,000$ nodes, the path length is less than 4 hops in the proposed method while Scribe requires more than four times the hops (16 hops). The path length affects the latency between publishers and subscribers, and also the number of messages as shown in Section 3.4.1.

Focusing on the pattern α of which the path lengths are most similar between the proposed method and Scribe, we also confirmed the cumulative distribution function (CDF) of path length as shown in Figure 3.11. It can be seen that there was no significant difference between the two curves, i.e., the above convenient features of the proposed method do not cause a serious undesirable effect on the distribution of path length.

3.4.3 Predictability of Load of Edge Brokers

We focused on the correlation between the number of sending/receiving and forwarding messages. This viewpoint is important for predicting the load of each node, namely

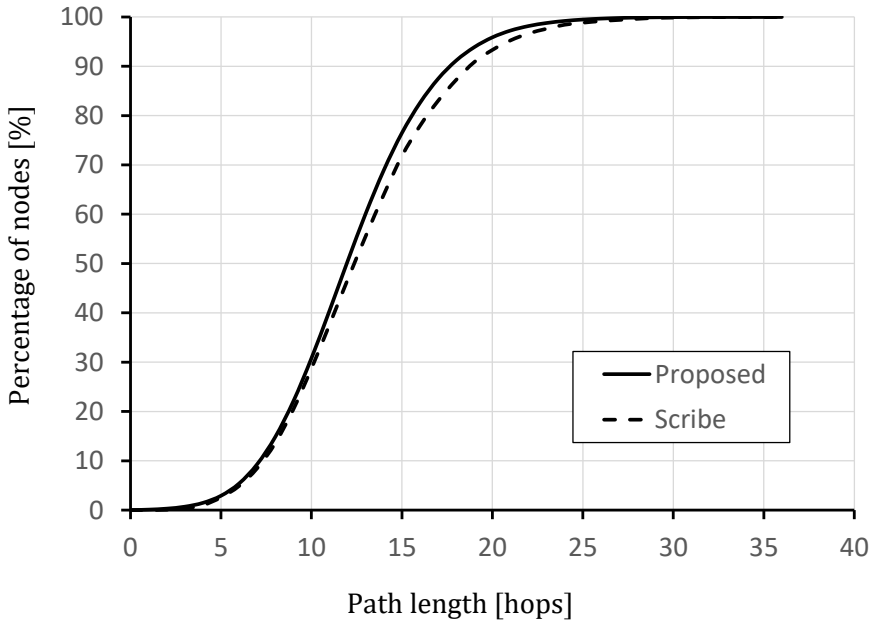

Figure 3.11: CDF of path length of pattern α.

edge broker.

In distributed pub/sub using structured overlay networks, each node is responsible for forwarding messages to relevant succeeding destinations. The forwarding load is determined according to properties of topics which the node is on the paths of. For example, the load must be heavy regarding a node responsible for forwarding messages of a topic whose publishers frequently send large amounts of data. The forwarding load is closely related to routing tables, which store the forwarding path information on each node. The information is registered on nodes in a different way for every method, as described in Section 3.2.2.

If the forwarding load correlates with the transmission load as publishers or receiving load as subscribers, each edge broker can easily predict the necessary specifications of hardware resources. For instance, if there is a device attempting to subscribe to a topic of video streaming, an edge broker which the device joins will be under a heavy load and should be strengthened.

Conversely, if the forwarding load does not correlate, it is difficult to predict from local information. Such a case is unsuitable when it is assumed that edge brokers

compose autonomous distributed networks such as the Internet, namely edge brokers are not managed by a single enterprise intensively but are arbitrarily added/removed by various enterprises or individuals.

In this regard, we conducted an experiment in which the simulator counts the number of forwarding and sending/receiving messages for every node. Precisely, "number of forwarding" is the number of times that a physical node forwards a message to others, including the initial hops from publishers. "number of sending" is the number of times that a publisher sends a message created on itself. "number of receiving" is the number of times that a subscriber receives a message associated with the topic the subscriber is subscribing to.

The conditions of this experiment are as follows: pub_t was 1 and sub_t was $1,000$. The number of topics was 100, thus the total number of nodes was $100,100$. The 100 publishers joining different topics were assigned different time intervals of sending. The intervals were calculated so as to make the number of transmissions during the simulation period become $1, 2, ..., 100$.

The simulator constructed overlay networks with the above conditions, and forced publishers to publish at respective intervals. After completion of counting the number of forwarding and sending/receiving, we normalized the data. The normalizing function for a data x in a data set X is as follows:

$$Normalize(x) = \frac{x - min(X)}{max(X) - min(X)}$$

Figure 3.12 shows the results obtained by plotting all publishers, and Figure 3.13 shows those by plotting $1,000$ subscribers which are randomly selected from all of subscribers. Regarding the proposed method, both the number of sending and receiving messages are clearly correlated with the number of forwarding messages. In Figure 3.13, nodes of the proposed method are plotted linearly on three different angled lines. This is because of the characteristic of multi-range forwarding in Multi-key Skip Graph, which forces each node to forward at most twice for each dissemination of a published message[¶]. In contrast, the results of Scribe indicate that there are no correlations.

[¶] Specifically, the forwarding paths in Multi-key Skip Graph compose incomplete binary trees. Each root node or intermediate node has one or two children, and each leaf node has no child. This is why each node forwards a message at most twice.

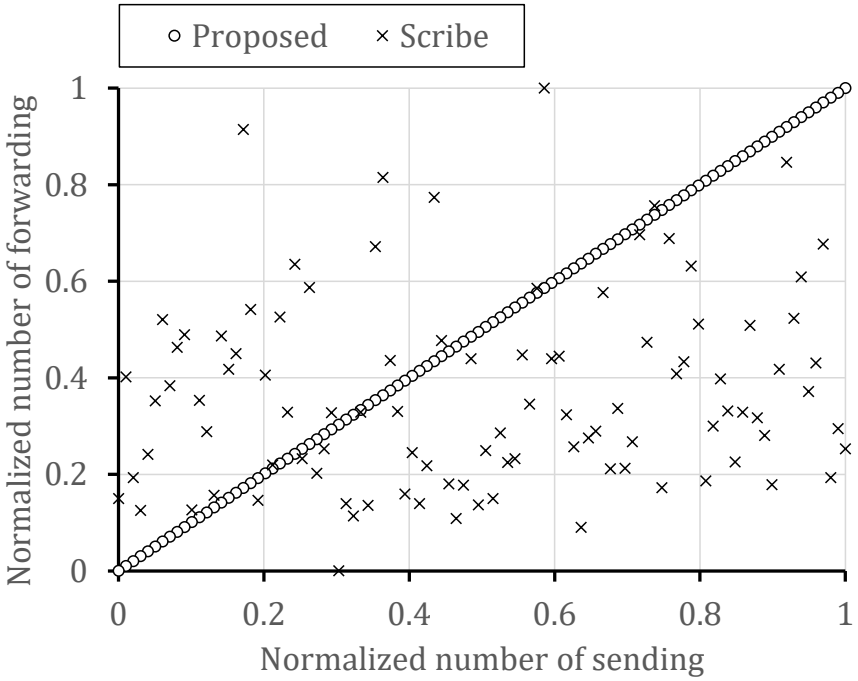

Figure 3.12: Correlation between the number of sending and forwarding.

The correlation coefficients and their confidence intervals at the 99% level were calculated[‖], as shown in Table 3.3. Note that the confidence interval of proposed method in Figure 3.12 is written as N/A because the Fisher transformation cannot be applied on the correlation coefficient value of 1.0.

Considering practical applications, it is natural that the frequency of publishing is unbalanced. For example, Twitter is a famous and large service based on pub/sub messaging. It has been reported that the number of tweets for every user follows the power law distribution, and 20% of users account for 84% of tweets [Welhuis]. Scribe or similar methods receive a negative effect from such an imbalance in terms of load predictability. In fact, there was a node that was forced to forward more than one thousand times the number of receiving count regarding Scribe in the experiment. In contrast, nodes in the proposed method forwarded at most twice the number of receiving count.

[‖] We used original data before normalization. Regarding Figure 3.13, we used all of the data, not sampled data.

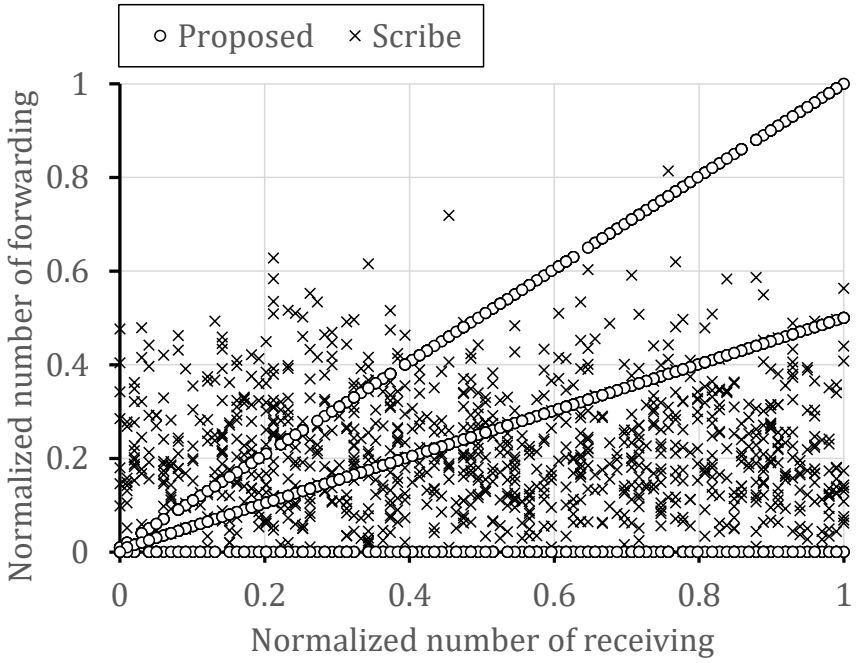

Figure 3.13: Correlation between the number of receiving and forwarding.

Table 3.3: Correlation coefficients and confidence intervals.

	Correlation coefficients	99% Confidence intervals
Figure 3.12 - proposed	1.0	N/A
Figure 3.12 - Scribe	0.1310	$-0.1290 \leq \rho \leq 0.3742$
Figure 3.13 - proposed	0.5483	$0.5426 \leq \rho \leq 0.5540$
Figure 3.13 - Scribe	-0.0045	$-0.0126 \leq \rho \leq 0.0037$

3.4.4 Size of Routing Tables

Furthermore, we conducted experiments for observing the size of routing tables on each node. Routing table size affects maintenance cost including the consumption of memory space.

At first, we focused on the transition of the average size of routing tables in response to the number of topics which each node publishes or subscribes to. The simulator constructed overlay networks with $100,000$ nodes. Half of them joined as publishers,

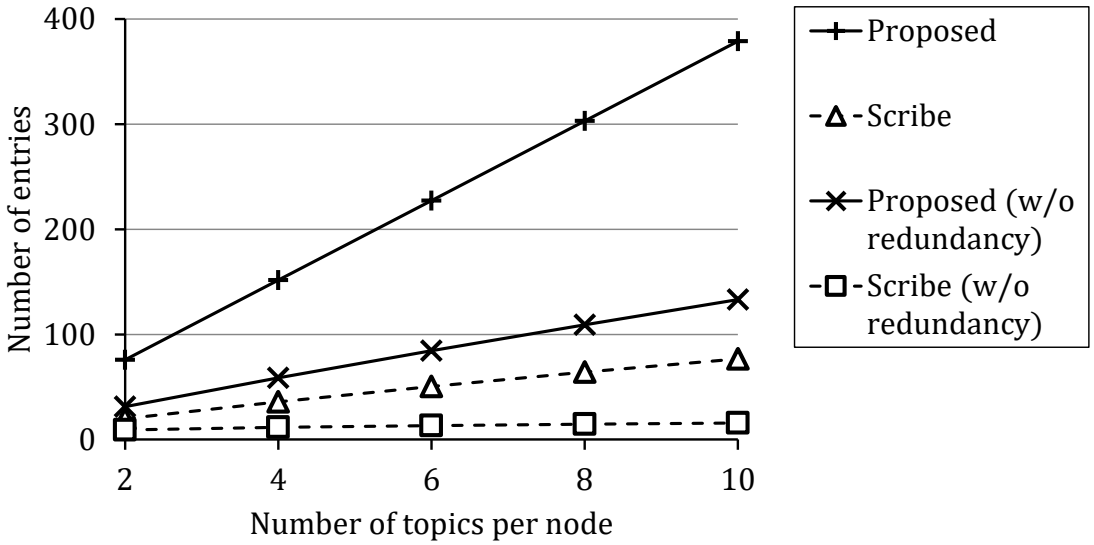

Figure 3.14: Size of routing tables.

and the remaining half joined as subscribers. Each node published/subscribed to specific number of topics uniformly: 2, 4, 6, 8 and 10. The topics are randomly selected from predefined $1,000$ topics. For each number of topics, the simulator calculated the average number of entries in routing tables of each node.

Figure 3.14 shows the result. For each method ("Proposed" or "Scribe"), the simulator counted two kinds of numbers: allow or disallow redundancy. In the legend of the graph, "w/o redundancy" denotes that the simulator counted the number of unique physical nodes in routing tables. The other data series which "w/o redundancy" is not attached to show the results of counting the number of entries in routing tables by allowing redundancy. The number of unique physical nodes in routing tables affects a kind of maintenance cost, when each node actively checks the existence of neighbors, i.e., periodically sends messages to them for confirming their activity.

In the Figure, the absolute values of the vertical axis of the proposed method are larger than Scribe, but the values do not increase intensively because they change linearly with the number of topics just like Scribe. Note that the proposed method based on Skip Graph can reduce the size of routing tables with a trade-off of the increase of the path length, by configuration of the base number of a logarithm.

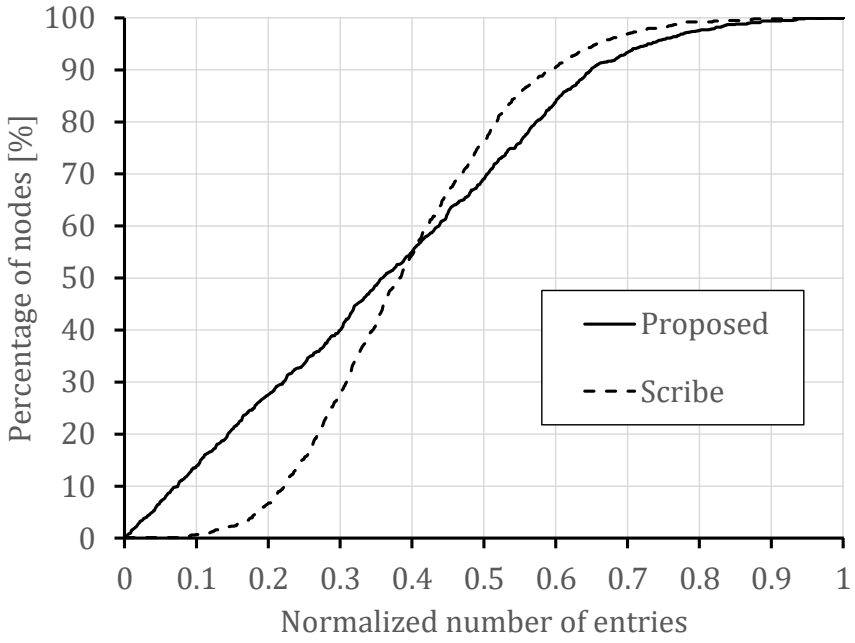

Figure 3.15: CDF of size of routing tables.

Subsequently, we focused on the CDF of the size of routing tables, to confirm the difference of the responsibility for the storage cost which is described in Section 3.2.2 and 3.4.3. The simulator constructed overlay networks with $1,000$ nodes, which all joined as subscribers. The number of topics to which each node subscribed was decided by selecting one from 1 to $1,000$ without duplication. The topics on each node were different one by one. The simulator counted the number of entries in routing tables of each node by allowing redundancy.

The obtained data were normalized by the function described in Section 3.4.3, and plotted as shown in Figure 3.15. It can be seen that the percentage of nodes in the proposed method is linearly increased compared to Scribe. This is caused by a characteristic by which the storage cost of a node is sensitive to the number of topics to which the node subscribes. On the other hand, in Scribe, the storage cost is shared among nodes. It leads to difficulty regarding load predictability as described in Section 3.2.2.

3.5 Conclusion of This Chapter

In this chapter, we proposed a method using Skip Graph which regulates publishers and subscribers of the same topic to compose connected subgraphs so that publishers can detect the absence of subscribers and suspend sending messages.

The proposed method utilizes a single overlay network to provide the pub/sub functionality, so that it is relatively easy to implement. There could be another approach where an overlay network is used as an infrastructure and a small overlay network for every topic is constructed onto it, but such approach makes implementation more complex. In particular, the mechanism of recovery from failure of nodes becomes complicated, since the influence on all of the overlay networks and connections among them must be considered. This leads to the lowering of fault tolerance or the increase of the maintenance cost, such as replicating some information additionally. In the case that above demerits are allowed, the approach of constructing multiplex overlay networks may have some advantages, e.g., flexibility in the topology. Since the easiness of implementation, the fault tolerance and the maintenance cost are generally essential for developing applications, clarifying the effectiveness of this sort of approach is out of the scope in this dissertation; it is one of the open issues.

From the results of simulation experiments, we confirmed that the above characteristics worked effectively with the proposed method and the path length of the proposed method was shorter than that of Scribe. It was also shown that the proposed method could predict the forwarding load, which Scribe could not.

As described in Section 1, the middleware we are aiming at is required to have high throughput. The proposed method in this chapter enables the middleware on the edge brokers to compose an autonomous distributed pub/sub system, and can achieve higher throughput than existing methods.

4

Improvement of Latency

In this chapter, we discuss issues of the latency. Although the proposed method described in Chapter 3 brings about high scalability and throughput, it involves the increase of the required time from publishers to subscribers. This could impair the advantage of real time dissemination of pub/sub messaging.

The distributed pub/sub middleware we are aiming at is expected to achieve low latency in addition to high throughput as mentioned in Section 1. Therefore, we present two techniques for improvement of latency; a routing algorithm of Skip Graph, and a formulation regarding client assignment strategies.

4.1 Routing Algorithm of Skip Graph

In Skip Graph, each node has a key and can issue a query by specifying a target range in the key space. Issued queries are delivered to nodes whose keys are included in the range.

Although Skip Graph enables a query to be delivered to one of the nodes inside the target range efficiently, it does not have definite methods of delivering the query to all nodes within the range from that node. Several simple ways, e.g., sequential forwarding,

can be considered as mentioned in [Beltran et al. 2008], but they are inefficient from the viewpoint of the latency or traffic volume.

Accordingly, we propose a novel algorithm named Split-Forward Broadcasting (SFB). It improves the average number of hops of Multi-Range Forwarding (MRF) used in Multi-key Skip Graph, which is described in Section 2.2.5.

4.1.1 Related Work

Beltran et al. [Beltran et al. 2008] have compared some methods of handling range queries with respect to the average number of messages and hops. The compared methods are as follows:

Sequential

Queries are forwarded along the doubly linked list at level 0, until the upper or lower bound of the range is found.

Broadcasting w/o memory

Each node within the range forwards the received queries to all its neighbors within the range.

Broadcasting w/ memory

It is an improvement in the number of messages of the broadcasting w/o memory method. Each message stores the list of nodes that have been visited so that it is avoided that nodes within the range receive the query several times.

Tree-based

Queries are forwarded by using links which are peculiar to Skip Tree Graph [Beltran et al. 2007].

The tree-based method assumes that there are additional links which are defined in the algorithm of Skip Tree Graph. We consider methods of handling range queries on normal Skip Graph for versatility, so this method is outside the scope of this dissertation.

The sequential method takes expected

$$O(N_R + \log N)$$

messages and hops, where N_R is the number of nodes within the target range R. On the other hand, both broadcasting methods require

$$O(N_R \log N_R + \log N)$$

messages and

$$O(\log N)$$

hops. Though the sequential method outperforms the broadcasting methods with respect to the number of messages, it requires a larger number of hops. That is, suppressing both the number of messages and hops by using these simple ways involves difficulties.

On the other hand, MRF used in Multi-key Skip Graph has both strong points of the sequential method and the broadcasting methods. It requires only

$$O(N_R + \log N)$$

messages and

$$O(\log N)$$

hops as described in Section 2.2.5.

4.1.2 Split-Forward Broadcasting

We propose a novel method named Split-Forward Broadcasting (SFB), which can reduce the average number of hops of MRF. The basic idea is to change the position of dividing the target range into subranges. The difference of forwarding processes of queries between SFB and MRF is depicted in Figure 4.1. SFB makes the target range divided into subranges by the key of neighbor nodes of the query receiver, whereas MRF makes it divided by the key of the receiver itself. Figure 4.2 and Figure 4.3 shows the pseudo-code of SFB and MRF respectively. The function **uponReceiving** is called when a node within the target range receives a query.

We explain the algorithm of SFB by using the example shown in the upper half of

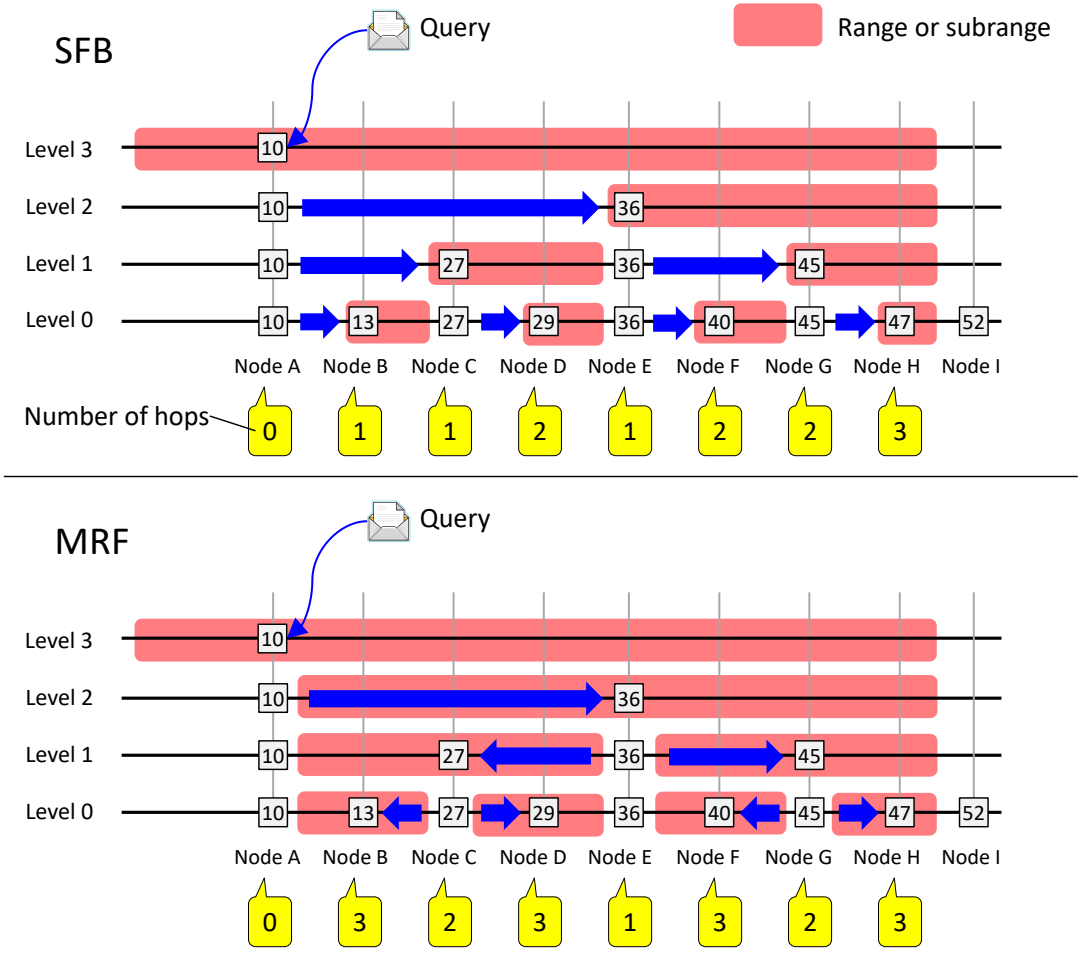

Figure 4.1: Difference of forwarding processes between SFB and MRF.

Figure 4.1. When node A receives a query with its target range

$$R = \{0 \leq key \leq 50\},$$

node A searches for neighbors which are connected to node A at the highest level among neighbors placed within R, one on each side. Regarding the right side, node E is the corresponding neighbor at level 2. Then node A divides R into subranges by the key of node E:

$$R_A = \{0 \leq key < 36\} \quad \text{and} \quad R_E = \{36 \leq key < 50\}.$$

```
1  uponReceiving(range, query) {
2      //Get delegate nodes of both the left side and the right side.
3      leftDelegateNode = getDelegateNode(range, LEFT)
4      rightDelegateNode = getDelegateNode(range, RIGHT)
5
6      if(leftDelegateNode != null OR rightDelegateNode != null) {
7          //If delegate nodes exist, transfer the query and the subrange to it.
8          if(leftDelegateNode != null) {
9              //Divide the target range into subranges by the neighbor's key.
10             subRange = getSubRanges(range, leftDelegateNode.key)[LEFT];
11             range = getSubRanges(range, leftDelegateNode.key)[RIGHT]
12             send(subRange, query) to leftDelegateNode;
13         }
14         if(rightDelegateNode != null) {
15             subRange = getSubRanges(range, rightDelegateNode.key)[RIGHT];
16             range = getSubRanges(range, rightDelegateNode.key)[LEFT]
17             send(subRange, query) to rightDelegateNode;
18         }
19
20         //Call uponReceiving() recursively for lower levels.
21         uponReceiving(range, query);
22     }
23 }
24
25 getDelegateNode(range, direction) {
26     delegateNode = null;
27
28     //Get a set of neighbor nodes of the specified direction from the routing table.
29     neighborNodes = routingTable.get(direction);
30
31     //Find a delegate node which has the highest level in nodes having keys within the range.
32     maxLevel = -1;
33     for each entry in neighborNodes {
34         if (entry.level > maxLevel && range.contain(entry.key)) {
35             maxLevel = entry.level;
36             delegateNode = entry;
37         }
38     }
39
40     return delegateNode;
41 }
```

Figure 4.2: Algorithm of SFB.

R_E is attached to the query and forwarded to node E from node A, while R_A is still possessed by node A.

Next, node A searches for another neighbor on its right side in the same way as mentioned above, but at levels lower than the level at which node A forwarded the query to node E. As a result, node A chooses node C at level 1. Then node A divides R_A into subranges by the key of node C:

$$R_{AA} = \{0 \leq key < 27\} \quad \text{and} \quad R_{AC} = \{27 \leq key < 36\}.$$

```
1  uponReceiving(range, query) {
2      //Get delegate nodes of both the left side and the right side.
3      leftDelegateNode = getDelegateNode(range, LEFT);
4      rightDelegateNode = getDelegateNode(range, RIGHT);
5
6      //If delegate nodes exist, transfer the query and the subrange to it.
7      if(leftDelegateNode != null) {
8          //Divide the target range into subranges by the local key.
9          leftSubRange = getSubRanges(range, localKey)[LEFT];
10         send(leftSubRange, query) to leftDelegateNode;
11     }
12     if(rightDelegateNode != null) {
13         rightSubRange = getSubRanges(range, localKey)[RIGHT];
14         send(rightSubRange, query) to rightDelegateNode;
15     }
16 }
```

Figure 4.3: Algorithm of MRF.

R_{AC} is attached to the query and forwarded to node C from node A, while R_{AA} is still possessed by node A.

Similarly, node A forwards the query to node B at level 0. Node A also executes such process on its left side. Every node within R except for node A will receive the query and execute the same process on the opposite side of the forwarder of the query. For example, node E receives the query from node A on its left side, and forwards the query to the corresponding nodes on its right side. (In Figure 4.1, node G and F.)

4.1.3 Analytical Comparison

In SFB, each node within the target range receives the same query only once. Hence, SFB requires expected

$$O(N_R + \log N)$$

messages. A query is forwarded from the issuer to one of the nodes within the range with expected

$$O(\log N)$$

hops, and subsequently it is forwarded from the node to every node within the range with expected

$$O(\log N_R)$$

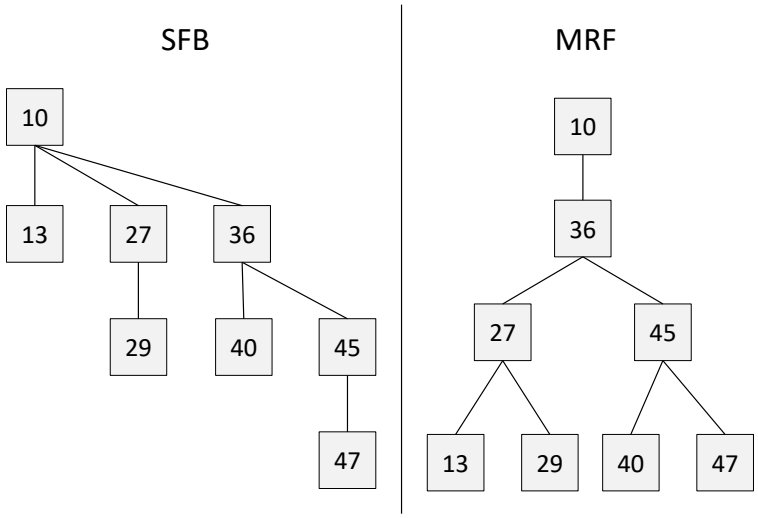

Figure 4.4: Difference of tree structures.

hops. Thus, SFB requires expected

$$O(\log N_R + \log N) = O(\log N_R \cdot N) = O(\log N)$$

hops, given that

$$N \geq N_R \geq 1$$

$$\therefore \log N \geq \log N_R \geq 0.$$

Difference of Tree Structures

The expected performances described in the preceding section are same as MRF, but SFB can reduce the actual average number of hops of MRF. Indeed, in Fig. 4.1, the number of hops from node A to each node within R which is depicted below the node name is relatively smaller than that of MRF. This is caused by the difference of structure of multicasting trees as shown in Figure 4.4.

In SFB, each node which receives a query forwards it to all of possible neighbors within the specified range or subrange. In contrast, in MRF, each node forwards it to at most two neighbors. As a result, SFB composes an unbalanced tree as shown in the left side of Figure 4.4, while MRF composes a balanced binary tree as shown in the right

side.

To compare analytically, we use following assumptions in this section:

- Skip Graph is composed with ideal membership vector, namely each list at each level consists of a row of evenly spaced nodes on the basis of the number of nodes.
- N_R is exponentiation of 2.
- The leftmost node is the first receiver of the query within the target range.

With these assumptions, the followings can be said: the number of nodes at each depth of the tree of SFB is the same as binomial coefficient, whereas that of MRF is exponentiation of 2. This tree structure of SFB is known as a binomial tree, which is used in a binomial heap [Vuillemin 1978]. For example, in Figure 4.4, the number regarding SFB is 1, 3, 3, 1, while the number regarding MRF is (1,) 1, 2, 4.

When $N_R = 131,072^*$, the difference of the number of nodes at each depth of the trees is shown in Figure 4.5. Regarding MRF, the number of nodes whose depth is deepest 17 is the largest. On the other hand, the SFB's tree has the largest number of nodes at the depth 8 and 9. This difference makes the superiority of SFB regarding the average number of hops.

Comparison of Average Number of Hops

From above assumptions, the average number of hops of SFB H_{SFB} can be calculated as below:

$$\begin{aligned} H_{SFB} &= \frac{1}{N_R} \sum_{k=0}^{\log N_R} \{ {}_{\log N_R}\mathrm{C}_k \cdot k \} \\ &= \frac{\log N_R}{2} \end{aligned}$$

* This is based on the rough estimation of large scale systems in Section 3.4.1.

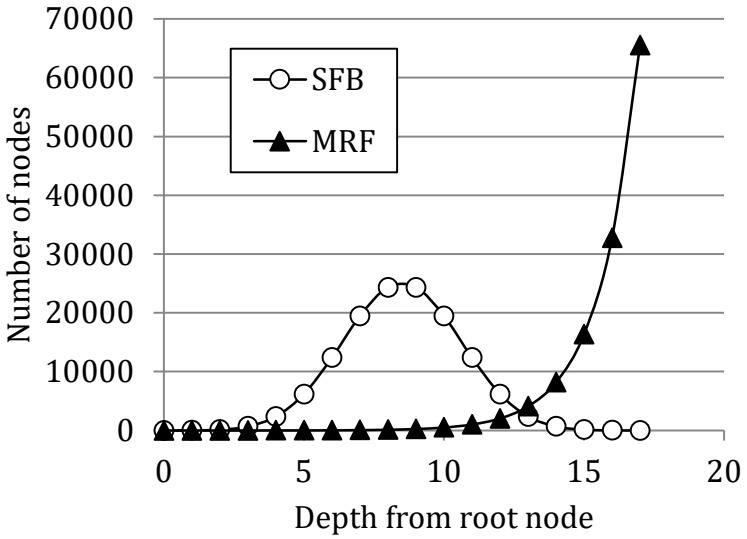

Figure 4.5: Comparison of number of nodes at each depth.

For MRF, the average number of hops H_{MRF} can be calculated as below:

$$\begin{aligned} H_{MRF} &= \frac{1}{N_R} \sum_{k=0}^{\log N_R - 1} \{(k+1) \cdot 2^k\} \\ &= \log N_R - 1 + \frac{1}{N_R} \end{aligned}$$

Figure 4.6 illustrates values of H_{SFB} and H_{MRF}, where the horizontal axis represents N_R[†] . From these, if N_R is enough large, it is clear that the average number of hops of SFB is approximately half of that of MRF.

As mentioned in Section 1, the distributed pub/sub middleware we are aiming at is expected to achieve low latency. Even though SFB increases the number of forwarding which each node handles, it less affects the latency compared to the average number of hops, since communication delays are generally more dominant than processing delays. Therefore, the fact that SFB can reduce the average number of hops is quite effective to achieve low latency.

[†] The maximum value of N_R is $131,072$, which is same as the value of N_R in Figure 4.5.

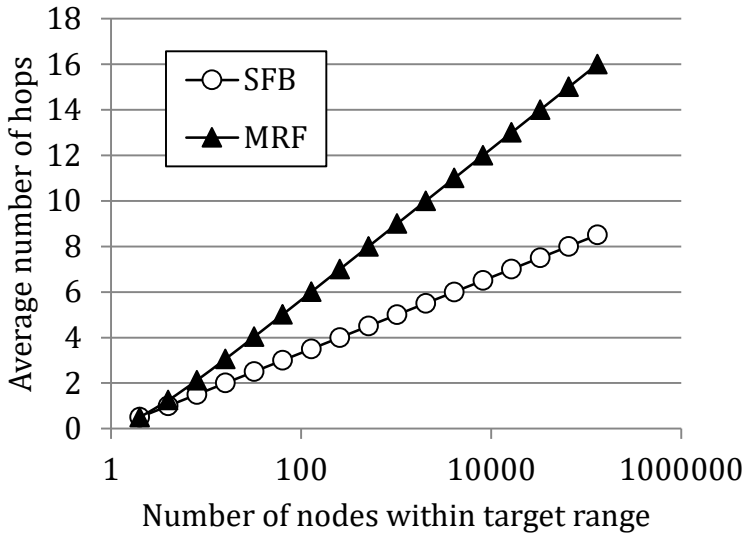

Figure 4.6: Average number of hops.

4.2 Client Assignment Strategies

In our architecture described in Chapter 1, multiple brokers are possible to be placed in a same place so that keeping enough computing resources. In addition, the proposed method in Chapter 3 is also applicable to the case that all brokers are placed in a specific local area, e.g., data centers. Discussions on such architecture have been brisk e.g., SDN-aware pub/sub systems [Akiyama et al. 2014, 2016].

In such cases, there is a range of alternatives about how to assign user end clients to each broker, as shown in Figure 4.7. In this section, we focus on the assignment of subscribers[‡] and compare two possible approaches: *intensive assignment* and *extensive assignment*. Since determining the subscriber assignment method affects the distribution delay time, we formulate the difference of it between the two. Subsequently, we derive the exchange of the superiority, and discuss about the optimization of subscriber assignment.

[‡]A subscriber means a user end client here, but not one of the brokers.

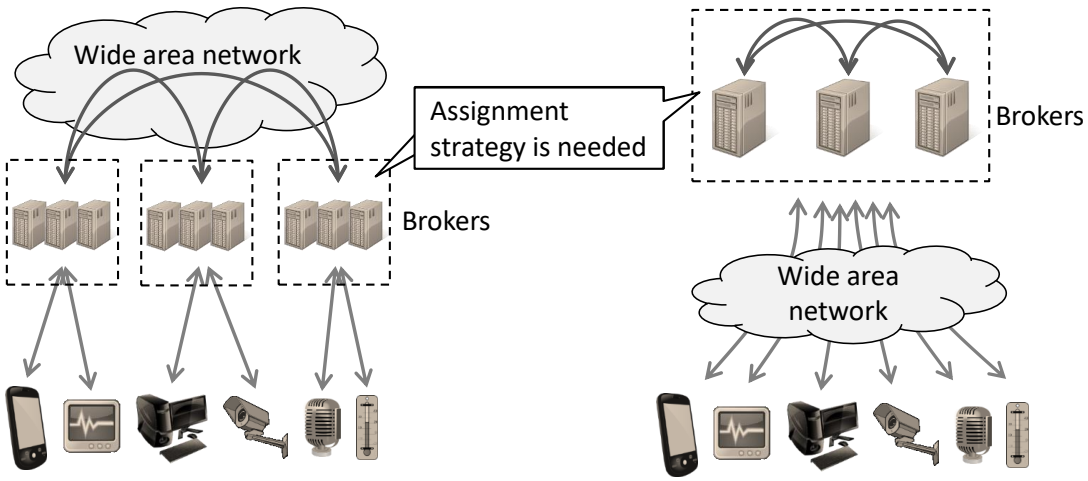

Figure 4.7: Cases of multiple brokers being in the same place.

4.2.1 Subscriber Assignment Methods

Throughout this section, we assume the followings:

- Each subscriber joins one topic.
- There is a necessary and sufficient number of brokers to accommodate subscribers.
- Each broker accommodates the same number of subscribers.
- Brokers compose an overlay network by our method described in Section 3.2.

In the method of Section 3.2, published messages are forwarded along the skip list composed by the publishers and subscribers of the corresponding topic. As described in Section 4.1, the forwarding paths configure a binary tree. To simplify, we assume the following in addition:

- For each topic, published messages are forwarded along the binary tree which is composed by a root node of a broker accommodating a publisher and the other nodes of brokers accommodating subscribers.

Based on the above preliminaries, there are two possible approaches; the intensive assignment and the extensive assignment.

Intensive Assignment

The intensive assignment is a method by which subscribers having a same topic are accommodated on a same broker as far as possible, as shown in the left side of Figure 4.8. As described in Section 4.2.1, a binary tree is composed for each topic, and published messages are forwarded along the tree. Each broker receiving the messages forwards them to subscribers, e.g., smartphones, actuators, etc., which are accommodated on the broker.

Extensive Assignment

The extensive assignment is a method by which subscribers having a same topic are accommodated on different broker as far as possible, as shown in the right side of Figure 4.8. As with the intensive assignment, published messages are forwarded by using a binary tree and finally delivered to the corresponding subscribers.

Qualitative Comparison

Because the tree size of the intensive assignment is smaller than that of the extensive assignment, network resources in the overlay network of brokers are less wasted. In addition, reduction of the size of the routing table of each broker leads restraining the consumption of computational resources of brokers.

If we assume that the traffic of all topics are uniform, the resource consumption of communication between brokers and devices is much the same regardless of the assignment methods. Accordingly, the intensive assignment has superiority from a viewpoint of resource consumption.

On the other hand, the superiority of the two methods is unclear from a viewpoint of the distribution delay time, which means the time required for delivering a published message to all corresponding subscribers. The intensive assignment brings about the reduction of the forwarding delay time because the path length on the tree is shorter. However, each broker has more subscribers so that the delay times of internal processing of brokers increase. Conversely, the extensive assignment reduces the delay time of internal processing of brokers and enlarges the forwarding delay time.

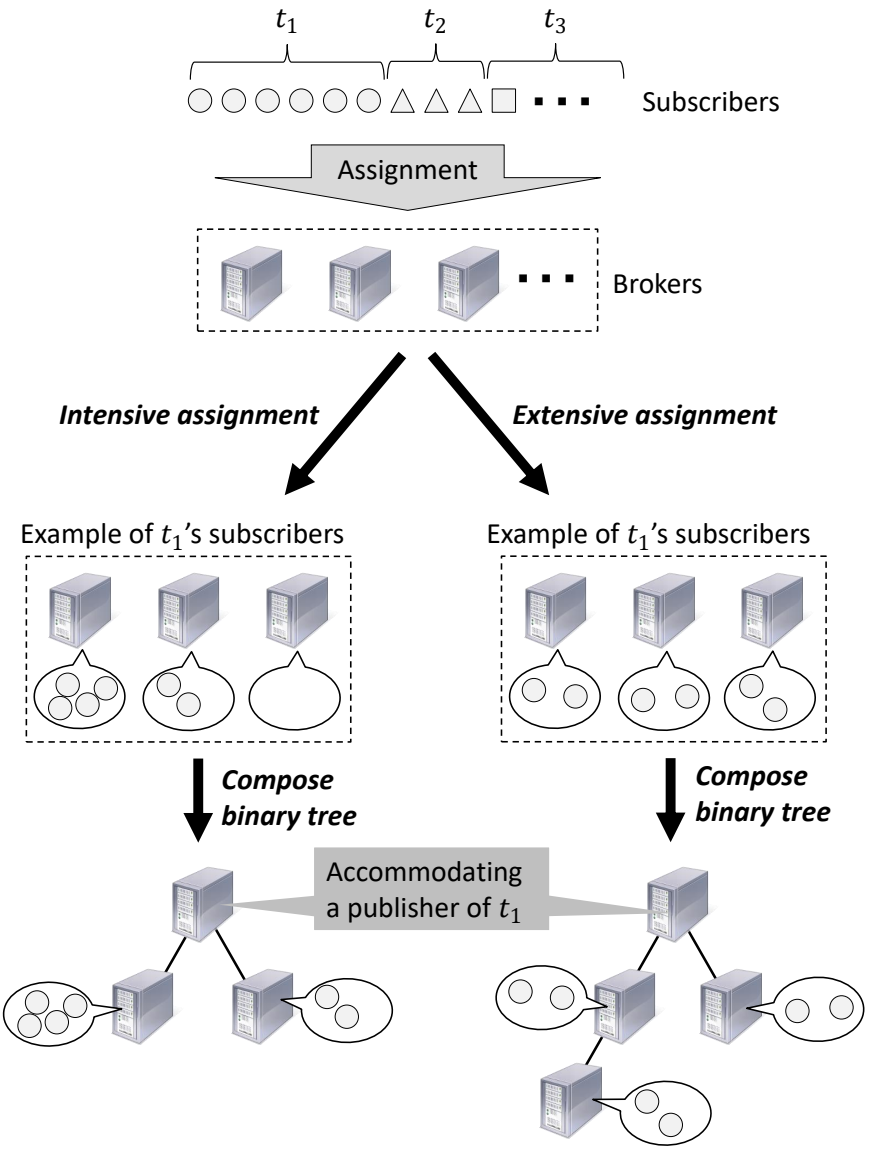

Figure 4.8: Subscriber assignment methods.

To consider the optimal subscriber assignment, it is required to clarify the superiority of the two methods regarding the distribution delay time. From next section, we formulate the difference of the distribution delay time between the two.

Table 4.1: Definitions of notations.

m	number of brokers
S	total number of subscribers
s_t	number of subscribers of topic t
t_c	delay time of communication between brokers
t_p	delay time of internal processing of brokers for each subscriber

4.2.2 Formulation of Distribution Delay Time

By using notations as defined in Table 4.1, the ideal number of brokers for each topic regarding the intensive assignment can be calculated as

$$\lceil s_t/(S/m) \rceil.$$

Hence the distribution delay time T_i of the intensive assignment is at most as follows:

$$T_i = \begin{cases} t_c \cdot \lfloor \log_2(\lceil s_t/(S/m) \rceil + 1) \rfloor + t_p \cdot (S/m) & (s_t \geq S/m) \\ \\ t_c + t_p \cdot s_t & (otherwise) \end{cases}$$

Regarding the extensive assignment, the distribution delay time T_e can be calculated as follows:

$$T_e = \begin{cases} t_c \cdot \lfloor \log_2(m) \rfloor + t_p \cdot (s_t/m) & (s_t \geq m) \\ \\ t_c \cdot \lfloor \log_2(s_t) \rfloor + t_p & (otherwise) \end{cases}$$

Therefore, the difference $T_d = T_i - T_e$ can be calculated as follows, based on approximation by ignoring the floor and ceiling functions.

$$
T_d = \begin{cases}
t_c \cdot (1 - \log_2(s_t)) + t_p \cdot (s_t - 1) & \\
\qquad \text{if} \quad s_t < S/m \quad \wedge \quad s_t < m & (4.1) \\
t_c \cdot (1 - \log_2(m)) + t_p \cdot s_t \cdot ((m-1)/m) & \\
\qquad \text{if} \quad s_t < S/m \quad \wedge \quad s_t \geq m & (4.2) \\
t_c \cdot \log_2((m \cdot s_t + S)/(s_t \cdot S)) + t_p \cdot (S/m - 1) & \\
\qquad \text{if} \quad s_t \geq S/m \quad \wedge \quad s_t < m & (4.3) \\
t_c \cdot \log_2(s_t/S + 1/m) + t_p \cdot ((S - s_t)/m) & \\
\qquad \text{if} \quad s_t \geq S/m \quad \wedge \quad s_t \geq m & (4.4)
\end{cases}
$$

Discussions

In the equations (4.1) to (4.4), there are three constants which are determined by the environment of vendors providing the brokers: m, t_c and t_p. We assume the values of these constants as follows[§] :

$$
\begin{aligned}
m &= 10,000 \\
t_c &= 1 \,[\text{msec}] \\
t_p &= 0.01 \,[\text{msec}]
\end{aligned}
$$

Figure 4.9 shows the change of T_d depending on s_t with above constants, for four patterns of S.

It can be seen that T_d becomes larger as S becomes large. Note that the maximum value of s_t is S. If $T_d > 0$, it can be said that the extensive assignment is superior to the intensive assignment regarding the distribution delay time.

As described in Section 4.2.1, the intensive assignment has superiority from a viewpoint of resource consumption. Therefore, if $T_d < 0$, the intensive assignment is basically preferred over the extensive assignment. On the other hand, if $T_d > 0$, subscribers

[§] We used $100,000$ nodes in Section 3.4.1, by considering the rough estimation of large scale systems. Since the client assignment strategies focus on the situation that each edge environment has multiple brokers, we set the value of m as $10,000$.

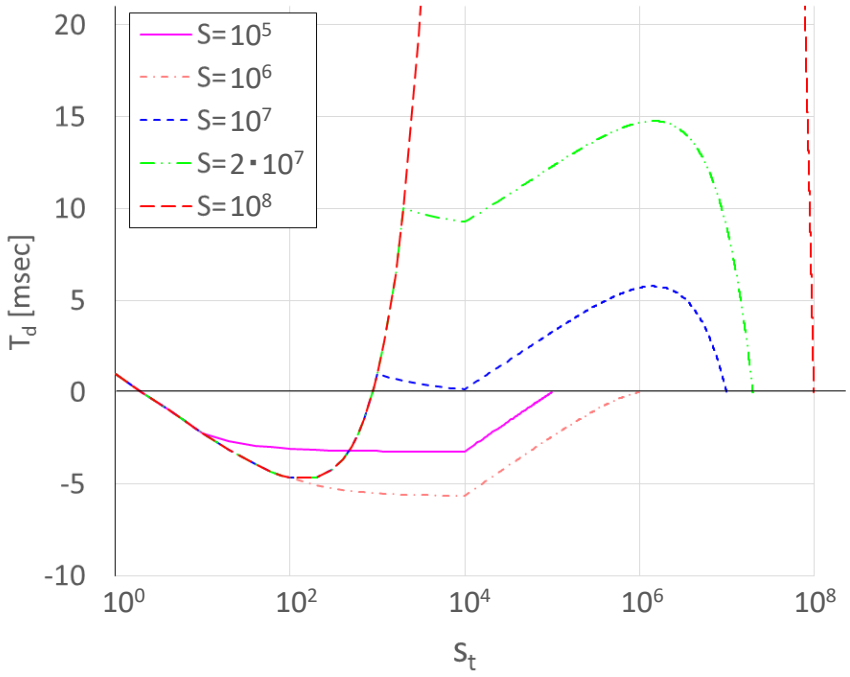

Figure 4.9: Difference of the distribution delay time between two methods.

should be assigned considering the trade-off between distribution delay time and resource consumption.

Focusing on the area of $T_d > 0$ in Figure 4.9, there are striking curves. For instance, the case of $S = 2 \cdot 10^7$, $T_d(s_t)$ is upward-convex in the range [10,000, 20,000,000], while it is downward-convex in the range [1, 2,000).

The values of s_t corresponding to the maximum value of the upward-convex curve can be calculated as follows by differentiating T_d shown in the equation (4.4) with respect to s_t:

$$
\begin{aligned}
T_d(s_t)' &= \frac{m \cdot t_c}{(m \cdot s_t + S) \cdot \log_e(2)} - \frac{t_p}{m} \\
&= 0 \\
\therefore s_t &= \frac{m \cdot t_c}{t_p \cdot \log_e(2)} - \frac{S}{m}
\end{aligned}
$$

In our assumption, $s_t \approx 1{,}440{,}695$ for $S = 2 \cdot 10^7$.

It can be said that subscribers joining to topics of which the numbers of subscribers are close to the above value are desired to be assigned by the extensive assignment, from the viewpoint of the distribution delay time. Note that the graph may change by the values of m, t_c and t_p.

4.3 Conclusion of This Chapter

In this chapter, we first proposed SFB which is a routing algorithm for range queries on Skip Graphs. SFB requires only the smaller number of hops and messages compared to the sequential method and the broadcasting methods. In addition, SFB can reduce the average number of hops of MRF. This is not only effective for reducing the average latency, but also for improving the churn tolerance because the reduction of hops lowers the probability of message loss by nodes' disappearing.

Subsequently, we focused that there is a range of alternatives about how to assign user end clients to each broker. We presented two assignment methods: the intensive assignment and the extensive assignment. By formulating the distribution delay time for each method, we derived the difference of the delay time between them, and discussed about the optimization of the subscriber assignment.

By the techniques in this chapter, we can reduce the latency of the proposed method described in Section 3 with keeping its high scalability and throughput. This contributes to meeting the requirements previously mentioned in Section 1; the distributed pub/sub middleware we are aiming at should achieve high throughput and low latency.

5

Development of Distributed Pub/Sub Systems

To develop the proposed method as a middleware, considering the actual environment brokers are placed at and supporting a practical protocol are inevitable. We focus on MQTT, a standardized protocol of topic-based pub/sub messaging. It has attracted much academic and industrial interest in recent years as one of the key technologies of IoT services [Al-Fuqaha et al. 2015].

Assuming the edge-based architecture, there is an issue of heterogeneity of brokers. Namely, an appropriate product of a broker is different according to an environment of each network edge. There are many choices: open source or proprietary, software or embedded appliance, difference in supported OSs or functional features, and so on. Even though some of existing products have functions of cooperation between multiple brokers, e.g., "bridge" of Mosquitto [Light 2017] and "cluster" of HiveMQ [HiveMQ], there is no interoperability between different products because any cooperation protocols are not standardized in the MQTT specification [Banks and Gupta 2015].

In this chapter, we propose Interworking Layer of Distributed MQTT brokers (ILDM), which enables arbitrary kinds of brokers to cooperate with each other. ILDM provides

APIs which facilitate rapid development of variety of cooperation algorithms. By using the APIs, we can easily implement the method proposed in Chapter 3, so that the distributed pub/sub architecture we aiming at can support the practical protocol. To clarify the feasibility, we show two basic algorithms using the above APIs in this Chapter.

In addition to the heterogeneity, there is another issue of benchmark. To evaluate and determine an appropriate architecture, benchmark method which can be used for both a single broker and multiple brokers is needed. For this matter, we formulate a benchmark method which ensures that error ratios of resulted performance are not more than 5 percent.

In this chapter, we explain the following three contributions:

- First, we give a fundamental idea of ILDM with two basic cooperation algorithms.
- Second, we provide a practical method for benchmark of MQTT broker/brokers.
- Third, we show the feasibility of ILDM-based cooperation through experiments.

5.1 Related Work

Dynomite [Dynomite] makes existing non distributed data stores, e.g., Redis and Memcached, into a distributed data store. The aim is to provide high availability and resiliency on storage engines which do not have those functionalities. BondFlow [Balasooriya et al. 2005] proposes a system enables encapsulated web services to interconnect. These are similar to ILDM from the viewpoint of modularizing functionality of interwork, while the target is different from ILDM.

As far as we know, there are no existing proposals of connecting heterogeneous MQTT brokers.

5.2 Interworking Layer of Distributed MQTT Brokers

In this section, we propose Interworking Layer of Distributed MQTT brokers (ILDM). ILDM-based cooperation is composed by multiple brokers and ILDM nodes. An ILDM

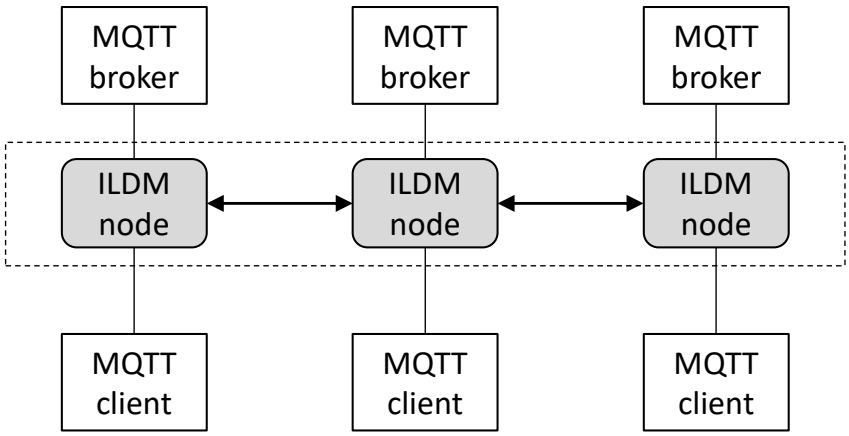

Figure 5.1: Interworking Layer of Distributed MQTT brokers.

node is arranged between a broker and clients as shown in Figure 5.1. As well as relaying MQTT clients and a broker as if it were a proxy, an ILDM node can connect with other ILDM nodes so that multiple and arbitrary kinds of brokers can communicate with each other via ILDM nodes.

Regarding an ILDM node, we assume the following notations: **local client** denotes a client which directly connects with the ILDM node, **local broker** denotes a broker which directly connects with the ILDM node, **remote ILDM node** denotes one of the other ILDM nodes included in the whole cluster, **neighbor ILDM node** denotes one of the remote ILDM nodes which directly connects with the ILDM node, **remote client** denotes a client which connects with a remote ILDM node, **remote broker** denotes a broker which connects with a remote ILDM node.

As there can be a variety of cooperation algorithms, an ILDM node provides APIs which facilitate rapid implementation. Figure 5.2 illustrates the components of an ILDM node.

We abstracted commonly used functions as five components: session manager, message listener, event listener, status listener, and message generator. These components have programming interfaces so that algorithm developers can implement their algorithms easily. By using these APIs, we can easily implement the method proposed in Chapter 3 with improving by techniques of Chapter 4. In this Chapter, we use two basic

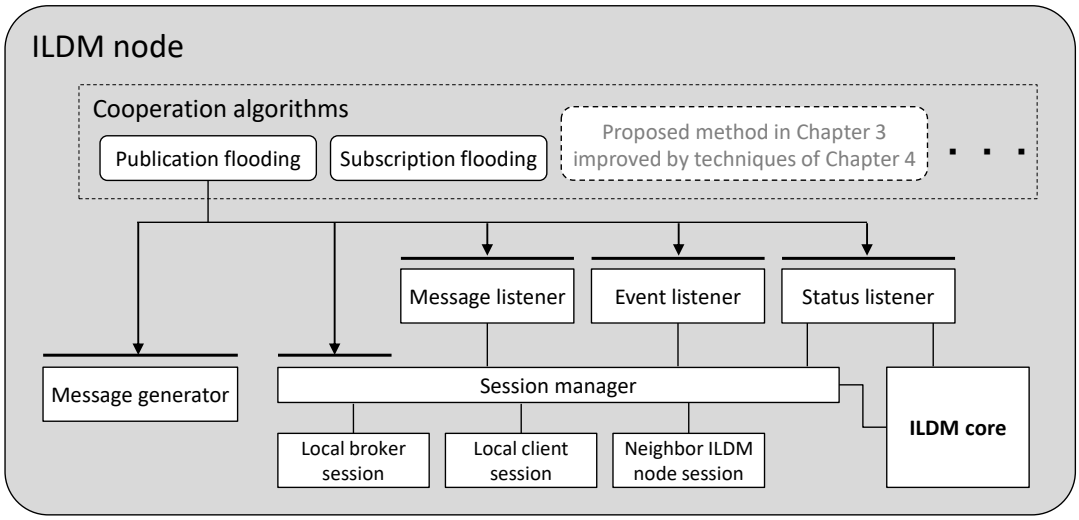

Figure 5.2: Components of an ILDM node.

algorithms "Publication flooding" and "Subscription flooding" to clarify the feasibility of ILDM. Details of them are discussed later in Section 5.3.

This architecture is useful not only for improving the performance such as throughput, but also for comparing cooperation algorithms. Unlike implementing a broker with capability of cooperation from scratch, ILDM-based implementations do not have differences in quality or design of implementations of functions irrelevant to cooperation. By utilizing this characteristic, we can fairly compare cooperation algorithms.

Major APIs which are provided by the components are listed in Table 5.1. Details of these APIs are described in the following sections.

5.2.1 Session Manager

Session manager manages communication sessions connecting with a local broker, local clients and neighbor ILDM nodes.

createSession is used to create a new session. The type **SessionInfo** indicates meta-information regarding a session, including session identifier, session status, and input/output streams.

closeSession is used to close an existing session.

Table 5.1: Major APIs of ILDM.

Component	API
Session manager	***SessionInfo createSession(Address ip, int port)*** ***void closeSession(SessionInfo session)*** ***boolean sendMessage(SessionInfo session, byte[] message)*** ***SessionInfo getPairedSession(SessionInfo session)***
Message listener	***void mqttMessageArrived(SessionInfo session, MsgType type, byte[] message)***
Event listener	***void onTcpDown(SessionInfo session)*** ***void onRecvFailure(SessionInfo session, Exception exception)***
Status listener	***void onSessionEstablished(SessionInfo session)*** ***void onSessionClosed(SessionInfo session)*** ***void onIldmStop()*** ***void onIldmAdd(Address ip, int port)*** ***void onIldmRemove(Address ip, int port)***
Message generator	***byte[] createMqttMessage(MsgType type, MqttParam parameters)***

sendMessage sends out a message to a specified session. It returns a boolean value indicating that the transmission is succeeded or not. The **message** argument is a byte array, e.g., an MQTT message.

getPairedSession is for obtaining a session making a pair with a specified session. When an ILDM node receives a TCP connection request on the listening port for local clients, it automatically creates a session with the local broker. These two sessions make a pair.

5.2.2 Message Listener

Message listener has an asynchronous callback API: **mqttMessageArrived**. It is called when an ILDM node receives an MQTT message. The type **MsgType** indicates the type of MQTT messages, e.g., CONNECT, CONNACK, and PUBLISH. This callback API is typically a start point of the algorithm-specific processes. If *session* is a local client's session, the ILDM node can relay the MQTT message to the local broker by using **getPairedSession** and **sendMessage**, and can behave according to a cooperation algorithm by using the copy of the message.

5.2.3 Event Listener

Event listener has asynchronous callback APIs.

onTcpDown is called when an ILDM node detects unexpected termination of a TCP connection, while **onRecvFailure** is called when an ILDM node failed to receive a message on a TCP connection.

5.2.4 Status Listener

Status listener enables developers to insert arbitrary processes in the middle of status transitions.

We define three statuses regarding sessions as shown in Figure 5.3. When an ILDM node established a TCP connection, the status of the session changes from **CLOSED** to **ESTABLISHED**. When terminating the TCP connection begins, the status changes to **TERMINATING**. After finishing the termination process, it changes to **CLOSED**.

We also define four statuses regarding the process of an ILDM node, as shown in Figure 5.4. When an ILDM node is started, the status changes from **STOPPED** to **RUNNING**. When the ILDM node begins to add or remove a neighbor ILDM node, it changes to **ILDM_ADDING** or **ILDM_REMOVING**. After finishing the process to add or remove, it changes back to **RUNNING**.

Status listener has following synchronous callback APIs.

onSessionEstablished is called after the status of the session specified as the argument changes to **ESTABLISHED**. Similarly, **onSessionClosed** is called after the status changes to **CLOSED**.

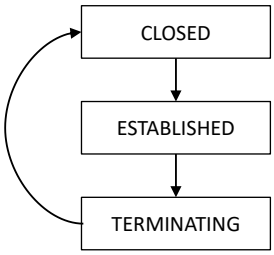

Figure 5.3: Status transitions of sessions.

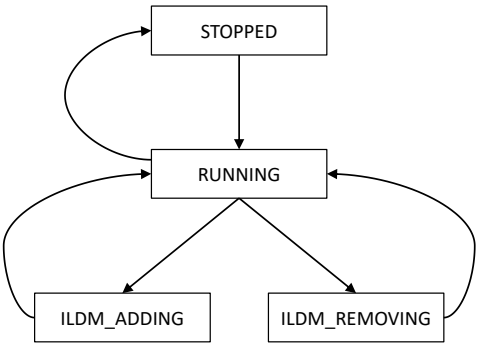

Figure 5.4: Status transitions of ILDM nodes.

onIldmStop is called when the status of an ILDM node changes to **CLOSED**, i.e., just before the ILDM node is terminated.

onIldmAdd is called when the status of an ILDM node changes to **ILDM_ADDING**. The process of adding a neighbor ILDM node will be coded in this callback. **onIldmRemove** is used in the same manner for removing.

5.2.5 Message Generator

Message generator is a utility component.

createMqttMessage returns an MQTT message as a byte array. The type **MqttParam** indicates the parameters of MQTT messages, e.g., client identifier and keep-alive interval.

5.3 Cooperation Algorithms

In this section, we propose two basic cooperation algorithms: Publication Flooding (PF) and Subscription Flooding (SF). These algorithms suppose ILDM nodes are connected in a tree structure which does not include closed paths.

5.3.1 PF-based Cooperation

PF is a method to share each PUBLISH message among all brokers via ILDM nodes.

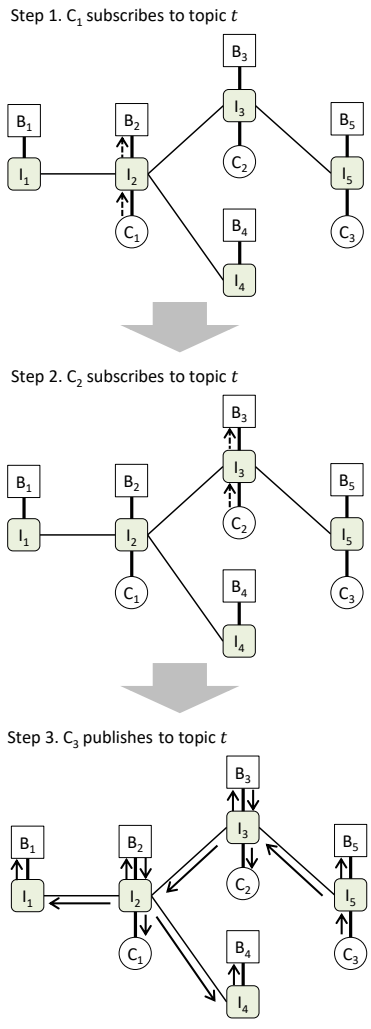

Figure 5.5: Example of PF-based cooperation.

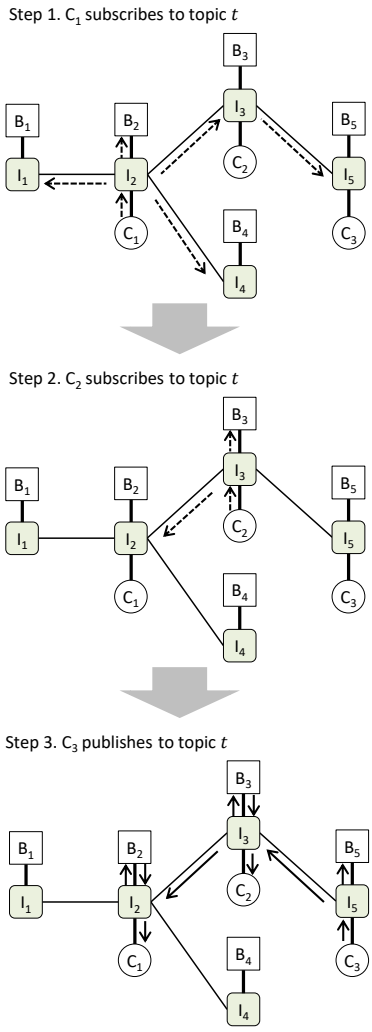

Figure 5.6: Example of SF-based cooperation.

Each ILDM node relays a SUBSCRIBE message received from a local client to its local broker. Regarding a PUBLISH message, an ILDM node does not only relay, but also transfers to its neighbor ILDM nodes. ILDM nodes, which receive the transferred PUBLISH message, send it to their own local broker. They further transfer the message to their neighbor ILDM nodes, if exist. Eventually, all connected brokers receive the PUBLISH message and forward it to their local clients subscribing to the corresponding topic.

Figure 5.5 shows an example. There are five sets of a broker and an ILDM node: B_1 and I_1 to B_5 and I_5. There are also three clients: C_1 to C_3. We consider the following three steps.

1. Step 1: C_1 subscribes to a topic t.
2. Step 2: C_2 subscribes to the same topic t.
3. Step 3: C_3 publishes to the same topic t.

Dotted arrows represent the flow of SUBSCRIBE messages, while solid arrows are the PUBLISH messages'.

When I_2 and I_3 receive a SUBSCRIBE message from C_1 and C_2, they just relay it to their local broker. As well as being relayed alike, a PUBLISH message from C_3 is transferred by I_5 to I_3, and spread to all ILDM nodes in a chain reaction.

5.3.2 SF-based Cooperation

Unlike PF, the basic idea of SF is to share subscription information among ILDM nodes.

When an ILDM node receives a SUBSCRIBE message, it informs the subscription information, e.g., topic name and QoS level to its neighbor ILDM nodes, as well as relays the message to its local broker. We call this operation "inter-subscribe", because it is as if it were the subscribe operation of the MQTT protocol between two ILDM nodes. For example, when an ILDM node X informs the information to another ILDM node Y, it means that "ILDM node X inter-subscribes against ILDM node Y".

When an ILDM node is about to inter-subscribe against a neighbor ILDM node, it checks overlapping with existing subscriptions. If an overlap is judged to be present, the ILDM node will not inter-subscribe redundantly. That is, inter-subscribe operations between ILDM nodes are to share only the difference from existing subscriptions.

Regarding a PUBLISH message, as well as relaying to the local broker, an ILDM node transfers it to neighbor ILDM nodes which have inter-subscribed to the topic of the message. ILDM nodes which receive the transferred PUBLISH message send it to their own local broker. They further transfer the message to their neighbor ILDM nodes which have inter-subscribed to the topic. Eventually, all brokers which have local clients

subscribing to the topic receive the PUBLISH message and forward it to corresponding subscribers.

Figure 5.6 shows an example. The topology and scenario is same as Figure 5.5. When I_2 receives a SUBSCRIBE message from C_1, it does not only relay the message to its local broker, but also inter-subscribes against I_1, I_3 and I_4. I_3 further inter-subscribes against I_5. In the next step, I_3 receives a SUBSCRIBE message from C_2 and subsequently inter-subscribes against I_2. I_3 does not inter-subscribe against I_5, because I_3 has already inter-subscribed in the first step. Similarly, I_2 does not inter-subscribe against I_1 and I_4.

A PUBLISH message from C_3 is transferred by I_5 to I_3, because I_3 has inter-subscribed to the topic t against I_5. I_3 also transfers the message to I_2, and finally C_1 and C_2 receive the message.

5.3.3 Furnishing MQTT-specific Functions

As we described in Section 2.1.2, MQTT has some specific functions. We show the way to enable clients to use these functions transparently over multiple brokers with PF and SF method.

QoS

Both in PF and SF, An ILDM node relays QoS-related messages such as PUBACK so that QoS level configuration is available between a local broker and local clients. Further, we can apply the idea of QoS control to transferring a PUBLISH message between adjacent ILDM nodes. This enables distributed brokers to adjust a tradeoff of reliability and performance.

Retain

PF method can provide Retain function without adding special processes, because each broker receives all PUBLISH messages and stores them if they have retain-flag being set to *true*. In case of SF method, when an ILDM node receives a PUBLISH message with retain-flag set to *true*, it needs to transfer the message to adjacent ILDM nodes even

though the topic is not inter-subscribed. This makes sure each broker can send out an appropriate retained message when a new subscriber of the corresponding topic comes.

Will

As described above, an ILDM node transfers a PUBLISH message when it receives it from local clients, not from a local broker. Hence, a will-message which comes from a local broker is not transferred to the neighbor ILDM nodes within the basic procedures of PF and SF methods.

According to the MQTT specification of version 3.1.1, a PUBLISH message from a broker does not have any information to know whether it is a will-message or not. Consequently, an ILDM node itself needs to store a will-message and a will-topic internally when it receives a CONNECT message.

When an ILDM node detects the unexpected closing of a network connection with a local client or the local broker, and if the will-flag of the connection is set to *true*, it sends out the corresponding will-message to its neighbor ILDM nodes. The ILDM node needs not to send the will-message to local clients, because the local broker sends it. In case of SF method, sending will-messages to neighbor ILDM nodes is executed only if the will-topics are inter-subscribed.

5.3.4 Qualitative Comparison

In PF method, each broker receives all PUBLISH messages regardless of the presence of corresponding subscribers. This means that the total number of ingress messages on each broker is basically same as the case of a single broker. Therefore, the effect of load distribution mainly depends on a dispersion condition of subscribers. The more scattered the subscribers are, the more effective this method is.

In SF method, a PUBLISH message is delivered to brokers which have subscribers of the same topic as the PUBLISH message. Brokers, which do not have such subscribers and are not in the paths of delivering the message, do not receive it. Hence this method is effective when publishers and subscribers of a same topic are convergently placed on a small sub-tree.

5.4 Implementation of ILDM

We implemented an ILDM node in Java, based on the MQTT version 3.1.1 specification.

Our implementation has a configuration file and some shell commands which can be executed at runtime. By these, we can specify neighbor ILDM nodes to add or remove not only statically but also dynamically. The process of adding or removing can be implemented by using *onIldmAdd* or *onIldmRemove* callbacks described in Section 5.2.4.

5.4.1 Implementation of PF and SF

We implemented PF and SF by using the APIs. Both implementations enable MQTT clients to use MQTT's functions such as QoS, Will, Retain, and Clean-session transparently over multiple brokers.

Unless otherwise specified, the descriptions in this section are common to PF and SF.

Communication between ILDM Nodes

We diverted the message format of MQTT to the communication between ILDM nodes, because of its lightness. That is, adjacent ILDM nodes establish TCP connections and use PUBLISH message for transferring PUBLISH messages, while using SUBSCRIBE message for inter-subscribing. They also use PINGREQ/PINGRESP messages to confirm connections of communication.

As a feature, we implemented a multi-session mechanism. An ILDM node can have multiple TCP connections for each neighbor ILDM node. Before the ILDM node transfers a PUBLISH message, it selects one connection to be used in a round-robin fashion. Since an ILDM node has to handle all of PUBLISH messages sent from its local clients, this mechanism is quite effective for improving entire throughput.

To distinguish connections with one neighbor ILDM node from with others, adjacent ILDM nodes exchange their identifier by using client-ID field of CONNECT message. The number of connections per one neighbor ILDM node can be set by a configuration file.

An ILDM node creates the same number of sessions for the local broker as the number of sessions it created for each neighbor ILDM node. These sessions are used for forwarding the transferred PUBLISH messages to the broker.

Relaying between Clients and Broker

An ILDM node relays MQTT messages from its local clients to its local broker, including messages of retransmission caused by QoS control. PINGREQ/PINGRESP messages are also relayed so that clients and the broker can confirm connections.

An ILDM node processes relaying in parallel to improve performance, except for SUBSCRIBE/UNSUBSCRIBE messages and their acknowledgement messages. Regarding those messages, relaying is processed serially because switching the order can cause unexpected status, e.g., a topic which was supposed to be unsubscribed is still subscribed.

Transferring between Adjacent ILDM Nodes

When using PF method, an ILDM node transfers a PUBLISH message received from a local client to adjacent ILDM nodes, asynchronously with relaying to the local broker. Like MQTT protocol, QoS control for transferring a PUBLISH message between adjacent ILDM nodes is available. The QoS level is set in the configuration file statically.

In case of SF method, an ILDM node receiving a SUBSCRIBE message inter-subscribes when it has not inter-subscribed to corresponding topics yet, after it has successfully finished relaying a SUBACK message to the local client. When the ILDM node receives a PUBLISH message, it transfers the message in the same manner as in PF method if the topic of the message is inter-subscribed by adjacent ILDM nodes. If it receives a PUBLISH message with the retain flag on, it transfers the message to adjacent ILDM nodes even though the topic is not inter-subscribed. This makes sure each broker can send out an appropriate retained message when a new subscriber of the corresponding topic comes.

Will-messages

According to the MQTT specification, PUBLISH messages from a broker do not have any information to know whether they are will-messages or not. However, an ILDM node is desired to be able to transfer will-messages to its neighbor ILDM nodes as necessary. To do this, we implemented the ILDM node so as to store a will-message and a will-topic internally when it receives a CONNECT message.

When an ILDM node detects the unexpected closing of a network connection with a local client or the local broker, and if the will flag of the connection is set to be *true*, it sends out the corresponding will-message to its neighbor ILDM nodes. The ILDM node needs not to send the will-message to local clients, because the local broker sends it. In case of SF method, sending will-messages to neighbor ILDM nodes is executed only if the will-topics are inter-subscribed.

Clean-session

When using PF method, there is no specific process regarding the clean-session.

In case of SF method, an ILDM node keeps the status of inter-subscribing related to a local client whose connection has the clean-session flag set to be $false$, even if the connection is unexpectedly closed.

Adding/Removing Neighbor ILDM Nodes

In case of PF method, the only thing an ILDM node has to do is creating/closing sessions when adding/removing neighbor ILDM nodes.

When using SF method, an ILDM node additionally needs to synchronize the status of inter-subscribing. For adding, the ILDM node calculates the set of topics to which it has already inter-subscribed. Subsequently, it inter-subscribes to all the topics of the set against the new neighbor ILDM node. If the ILDM node does not have existing neighbor ILDM nodes, the set of topics to which its local clients have subscribed is used instead. The new neighbor ILDM node also inter-subscribes against the ILDM node in the same manner. Afterward, both ILDM nodes newly paired proceed the normal process of being inter-subscribed. For removing, ILDM nodes, which are about to disconnect, inter-unsubscribe against each other.

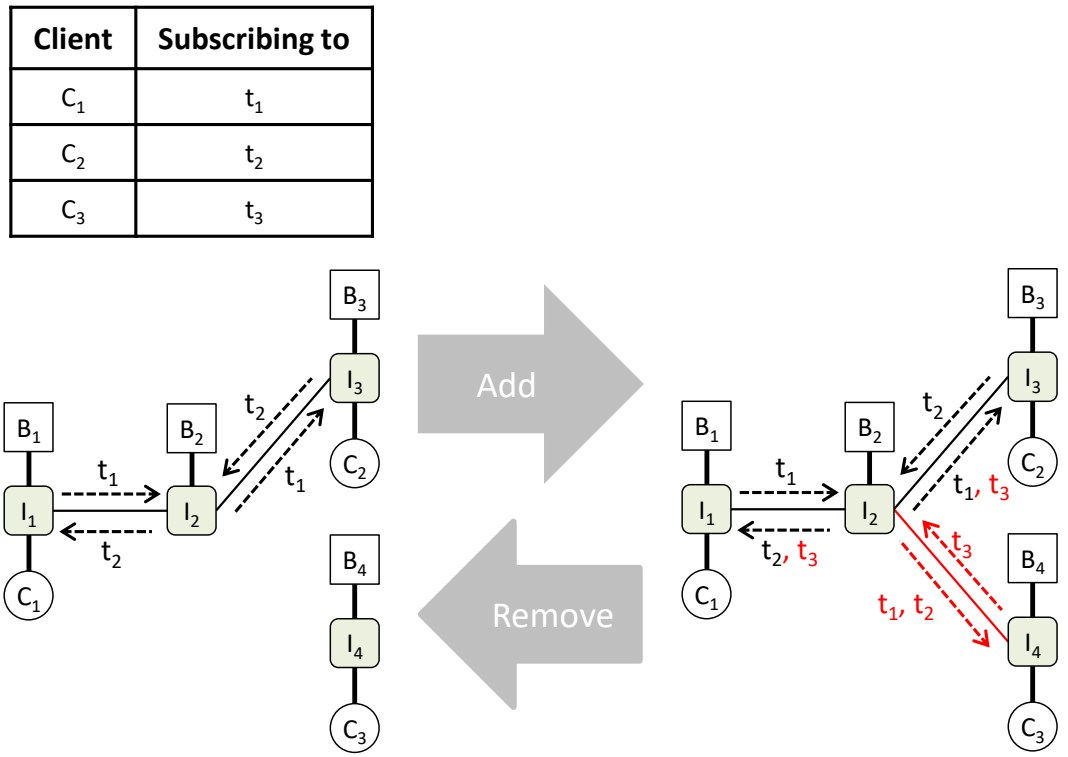

Figure 5.7: Adding/removing neighbor ILDM nodes.

Figure 5.7 shows an example. I_2 and I_4 are initially not connected. When I_2 and I_4 add each other as a new neighbor, I_2 inter-subscribes to the topic t_1 and t_2 against I_4, because I_1 and I_3 have inter-subscribed to these topics. I_4 inter-subscribes to the topic t_3 to which C_3 has subscribed. I_2 subsequently inter-subscribes against I_1 and I_3 by adding the topic t_3.

5.5 Benchmark System for MQTT Brokers

To verify the effects of ILDM, we formulate a benchmark method which can be applied for both a single broker and multiple brokers.

5.5.1 Performance Indexes

We consider the following four viewpoints as performance indexes of MQTT brokers.

Table 5.2: Spec of servers.

	Type S_1	Type S_2
Processor	Atom C2750 (8 core, 2.4 GHz)	Xeon E5-2690V3 (12 core, 2.6 GHz)×2
Memory	16 GB	256 GB
OS	Ubuntu 14.04	Ubuntu 14.04
NW	1 GbE	10 GbE

ingress throughput

Number of messages brokers receive from publishers per unit time.

egress throughput

Number of messages brokers send out to subscribers per unit time.

latency

Required time since a publisher sends a message until a subscriber receives it.

loss rate

Ratio of the number of missed messages to that which subscribers should receive.

Figure 5.8 shows the components of the benchmark system. Multiple publishers and subscribers are run for measuring throughput and loss rate. We denote these clients by **t-client**. Another pair of a publisher and a subscriber is also placed on a server different from those for t-clients. We denote these clients by **l-client**. They specify same topic so that latency can be acquired by calculating the turn around time. In parallel, resource usage on each server is recorded, e.g., CPU usage.

5.5.2 Hardware Environments

In the benchmark system, we connected servers described in Table 5.2 by using a non-blocking L2 switch. Ten type S_1 servers and one type S_2 server were prepared, and we used an appropriate number of servers for each evaluation pattern.

Power saving functions such as EIST (Enhanced Intel SpeedStep Technology) are disabled to avoid unexpected performance control and to clarify the relation between the spec of the servers and the results of measurement.

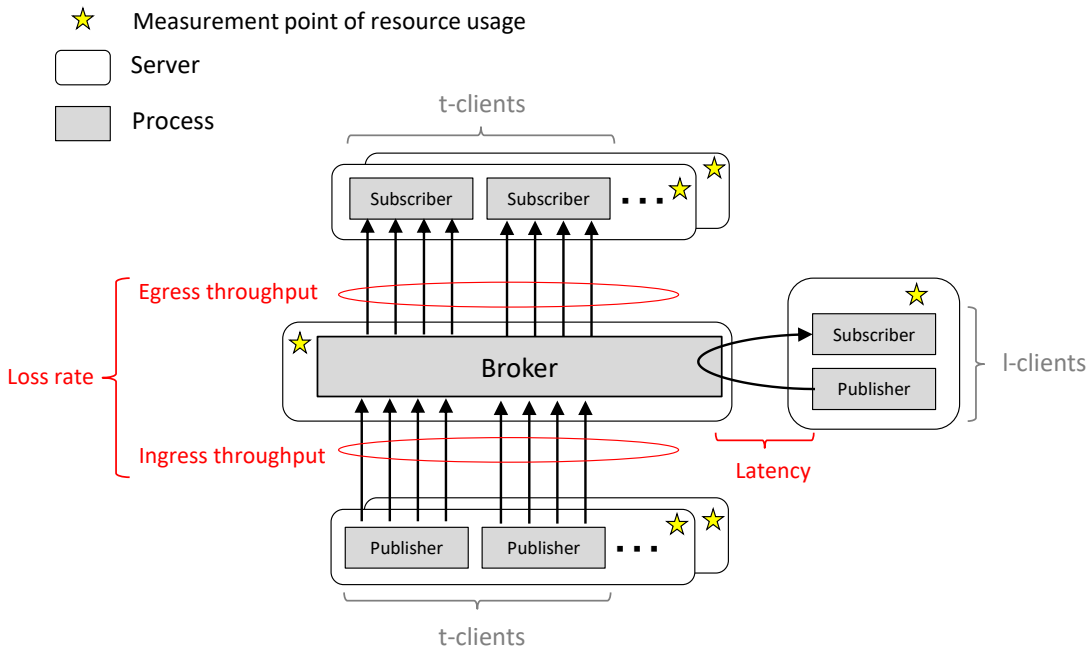

Figure 5.8: Overview of the benchmark system.

The time of servers on the benchmark system are synchronized by using an NTP server placed in the same network segment.

5.5.3 Load Testing Tool

We implemented a load testing tool, which is operated as t-clients and l-clients. We used the client library of SurgeMQ [SurgeMQ] known for its high performance so that the tool can send/receive PUBLISH messages with high frequency.

Functions which this tool has are as follows:

- Subscribe to topics according to a pre-defined scenario.
- Send PUBLISH messages to topics at a certain interval during certain period of time, according to a pre-defined scenario. t-clients and l-clients can be configured with different intervals.
- Record logs of sending and receiving PUBLISH messages.
- Gather logs recorded on multiple servers after the duration.

- Calculate performance indexes from the gathered logs.

During the measurement period of time, this tool records a timestamp when: (i) a publisher sends a PUBLISH message, and (ii) a subscriber receives a PUBLISH message.

By using (i) of t-clients, the tool calculates the ingress throughput for each second. Similarly, the egress throughput for each second is also calculated by using (ii).

For acquiring latency, the tool uses the difference between (i) and (ii) of l-clients. Since the tool sets identifiers of a client and a message to a payload, it can determine the correspondence of timestamps.

Regarding loss rate, the tool firstly calculates the sum of the number of PUBLISH messages to be received by each t-client as a subscriber, against the number of produced messages calculated by (i). This can be derived analytically by considering the scenario, i.e., the number of subscribers belonging to each topic. Secondly the tool finds the difference between the above sum and the number of arrived messages calculated by (ii). This is the number of missed messages. Finally, the tool calculates the loss rate as the ratio of the number of missed messages to the sum.

5.5.4 Definition of Performance Limit

The aim of this benchmark system is to obtain the limit of performance of broker/brokers. Supposing every t-client acting a publisher sends PUBLISH messages at a same interval, we define the limit of performance as follows.

Definition 1. *If measured throughput satisfies the following restriction, the performance is under the limit.*

$$\frac{\{\text{egress throughput}\}}{\{\text{ingress throughput}\} \cdot \{\text{sp-ratio}\}} \geq 0.99$$

$$\begin{aligned} &\text{where} \\ \text{sp-ratio} &= \frac{\sum_i P(t_i) \cdot S(t_i)}{\sum_i P(t_i)}, \\ P(t_i) &= \{\text{number of publishers of } i\text{th topic}\}, \\ S(t_i) &= \{\text{number of subscribers of } i\text{th topic}\}. \end{aligned}$$

This is based on the idea that if egress throughput is less than ingress throughput

multiplied by sp-ratio, the number of pending messages in the brokers is monotonically increasing. In other words, this definition represents the limit of allowable continuous load.

5.5.5 Benchmark Procedure

To find the maximum performance satisfying the restriction defined in Definition 1, we introduce a new benchmark method. This method tries to find the point of the very limit by varying the interval of PUBLISH messages. It is conducted along with the following steps.

Step 1: Conduct measurement repeatedly with doubling the interval of sending a PUBLISH message. For example: 1 ms, 2 ms, 4 ms, 8 ms, 16 ms, ….

Step 2: From the results of Step 1, find the minimum interval satisfying the restriction stated in Definition 1.

Step 3: Divide the segment between the minimum interval and the smaller interval next to the minimum interval into 20. For example, if the minimum interval is 4ms, we divide the segment between 4ms and 2ms like as : 2 ms, 2.1 ms, 2.2 ms, …, 3.9 ms, 4 ms.

Step 4: Conduct measurement for each interval calculated in Step 3.

Step 5: From the results of Step 4, find the minimum interval satisfying the restriction in Definition 1.

Finally, the result by using the minimum interval clarified in Step 5 indicates the limit of performance.

This result is ensured that the error ratio is not more than five percent. In other words, the throughput resulted by using the smaller interval next to the minimum interval is at most 1.05 times larger than using the minimum interval. We can prove it as follows:

Proof. We assume that x is the minimum interval in Step 2. Therefore $x/2$ is the smaller interval next to the minimum interval. Here the stepping width calculated in Step 3 is

$$(x - x/2)/20 = x/40.$$

We denote the width as y. The error ratio is at least

$$x/(x-y)-1$$

and at most

$$(x/2+y)/(x/2)-1.$$

Consequently, the highest error ratio is 0.05. □

To evaluate the performance appropriately, it is also important whether there is a bottleneck caused by things other than the performance of broker/brokers. We considered the following viewpoints:

- TCP flow control caused by overloaded receiving on subscribers.
- Ethernet flow control caused by lacking bandwidth of subscribers' side.
- TCP retransmission caused by packet loss on the network.

In our benchmark system, we monitored above matters by checking the window size in TCP ACK frames, occurrence of PAUSE frames, and retransmission logs.

5.6 Evaluation

We conducted some experiments by using the benchmark method described in Section 5.5. As mentioned in Section 1, we are aiming to provide techniques required for realizing the distributed pub/sub middleware which should achieves high throughput and low latency. Therefore, the experiments are designed to confirm the feasibility of ILDM mainly from the viewpoint of improvement of throughput, which is carried from reducing consumption of resources with the edge-based architecture. Additionally, we also evaluate the latency compared to a single broker.

In each experiment, we ran the load testing tool for 80 seconds. The performance indexes stated previously were calculated by excluding the first and last 10 seconds, i.e., substantial measurement time was 60 seconds. QoS level was set to 0, and the size of payload of each PUBLISH message was 32 bytes. This is because actual IoT services generally handle small data with high frequency. Since the configuration of QoS has a

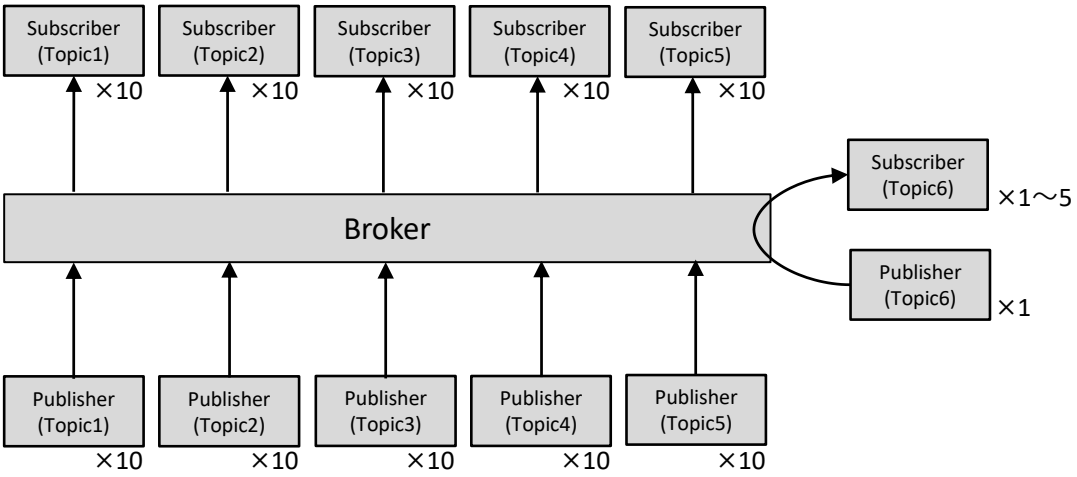

Figure 5.9: Configuration of topics and clients.

large influence on the throughput, large scale systems tend to set QoS level to 0 and have external recovery mechanisms in the upper layer, i.e., "Data consumer" in Figure 1.3, against loosing messages.

Figure 5.9 shows the configuration of topics and clients. There are five topics for measuring throughput and loss rate: from topic1 to topic5. These topics have 10 publishers and 10 subscribers respectively, thus the sp-ratio is 10. There also be topic6 with a publisher and subscribers for measuring latency. This publisher sends a PUBLISH message for each one second.

As previously stated, we denote the clients of topic1 to topic5 by **t-clients**, and the clients of topic6 by **l-clients**. In case of multiple brokers, we used additional l-clients. We describe about this later.

We calculated the average of ingress/egress throughput and latency in the measurement time of 60 seconds, and found the limit of performance by using the benchmark method.

5.6.1 Evaluation of Single Brokers

As preliminary experiments, we evaluated the performance of open-source MQTT brokers alone. The aim is to get a reference of choosing a broker in evaluation of ILDM,

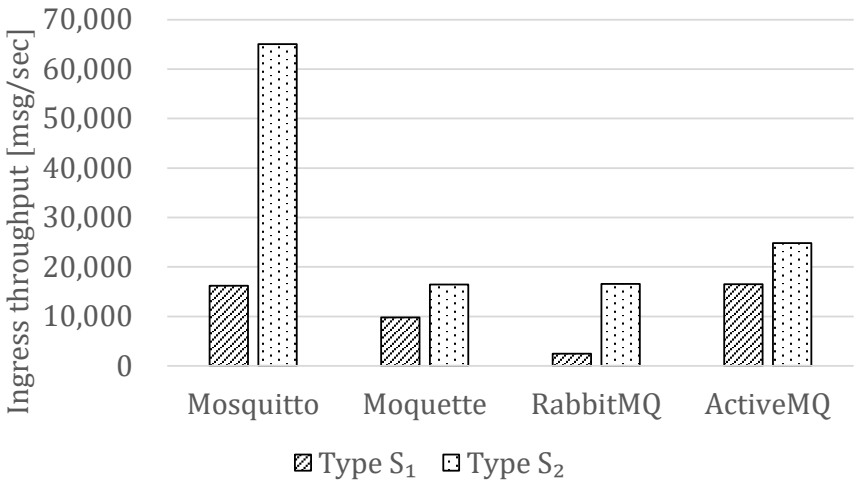

Figure 5.10: Evaluation of single brokers: ingress throughput.

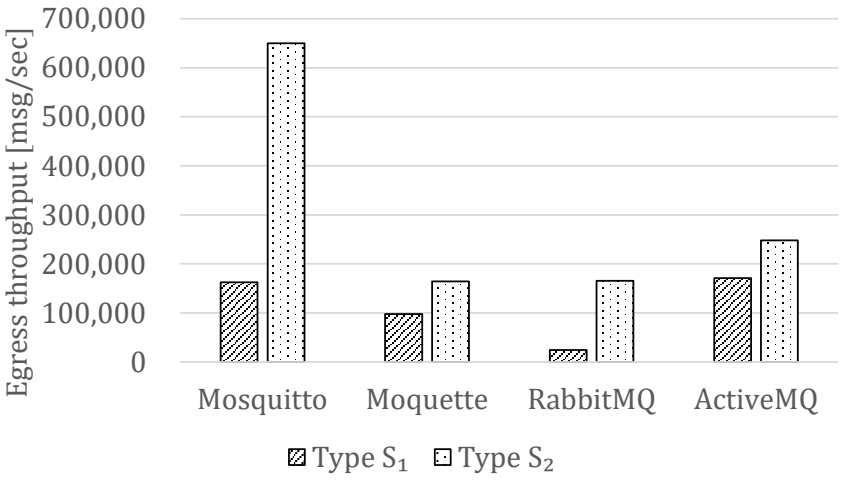

Figure 5.11: Evaluation of single brokers: egress throughput.

as well as providing knowledge of performance characteristics of well-known MQTT brokers.

We used the following four brokers: Mosquitto 1.4.5, Moquette 0.8 [Moquette], RabbitMQ 3.6.0 [Videla and Williams 2012], and ActiveMQ 5.13.3 [Snyder et al. 2008]. We measured the performance by changing the types of servers, S_1 and S_2, on which we

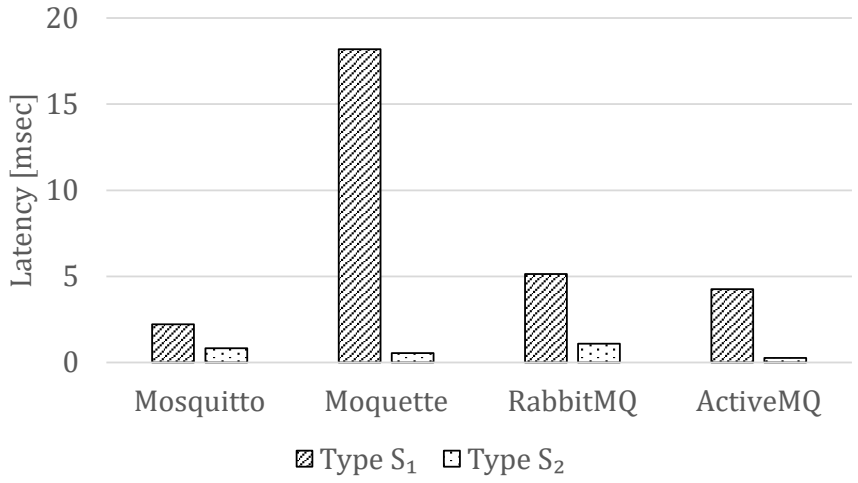

Figure 5.12: Evaluation of single brokers: latency.

ran the brokers.

Figure 5.10 and 5.11 shows the results of throughput. As the benchmark method indicates, egress throughputs are almost equal to ingress throughputs multiplied by the sp-ratio 10.

When using type S_1 server, the performance of ActiveMQ and Mosquitto are the tops. ActiveMQ is slightly larger, but almost even. Regarding type S_2 server, Mosquitto is the largest and its egress throughput reaches over $600,000$.

Figure 5.12 shows the result of latency. As for latency, the shorter the better. In case of type S_1 server, Mosquitto has the shortest latency. On the other hand, using type S_2 server, every broker has approximately less than 1 millisecond latency. ActiveMQ is the best, but the difference is quite small.

In these measurements, the loss rate was zero for all patterns.

5.6.2 Evaluation of ILDM-based Cooperation

We evaluated the performance of ILDM-based cooperation. Although the principal feature of ILDM is the capability of connecting heterogeneous brokers, we used one kind of broker to clarify the performance characteristics of ILDM itself. We chose Mosquitto because it indicated relatively better performance among the four brokers in

Table 5.3: Patterns of measurments.

Pattern	Description
A	Using one broker with one ILDM node.
B	Using 5 brokers with ILDM. t-clients are placed with no locality.
C	Using 5 brokers with ILDM. t-clients are placed with low locality.
D	Using 5 brokers with ILDM. t-clients are placed with high locality.

Section 5.6.1.

Table 5.3 states the patterns of measurements. In these patterns, pairs of a broker and an ILDM node are placed on one or five S_1 servers.

Pattern B, C, and D use 5 pairs of a broker and an ILDM node connected in a row. Each ILDM node has the same number of local t-clients, i.e., 20 t-clients. These patterns have a difference of locality of those 100 t-clients. The number of t-clients placed on each ILDM node is as follows:

Pattern B: two publishers and two subscribers for every five topics.

Pattern C: Eight publishers and eight subscribers of a topic, one publisher and one subscriber of a different topic, one

Pattern D: 10 publishers and 10 subscribers of a topic. publisher and one subscriber of another different topic.

In pattern B, C, and D, each of five ILDM nodes has a l-client as a subscriber of topic6. Only one ILDM node placed at the end of the list of the five ILDM nodes has one more l-client as a publisher of topic6. Hence, five data of latency are obtained every second in the measurement time of 60 seconds.

Figure 5.13 and 5.14 shows the results of throughput. “PF” and “SF” in the legend denote the cooperation algorithms. The results of single Mosquitto broker are depicted again for comparison, on the right side of the graphs. Same as results in section 5.6.1, egress throughputs are almost equal to ingress throughputs multiplied by the sp-ratio 10.

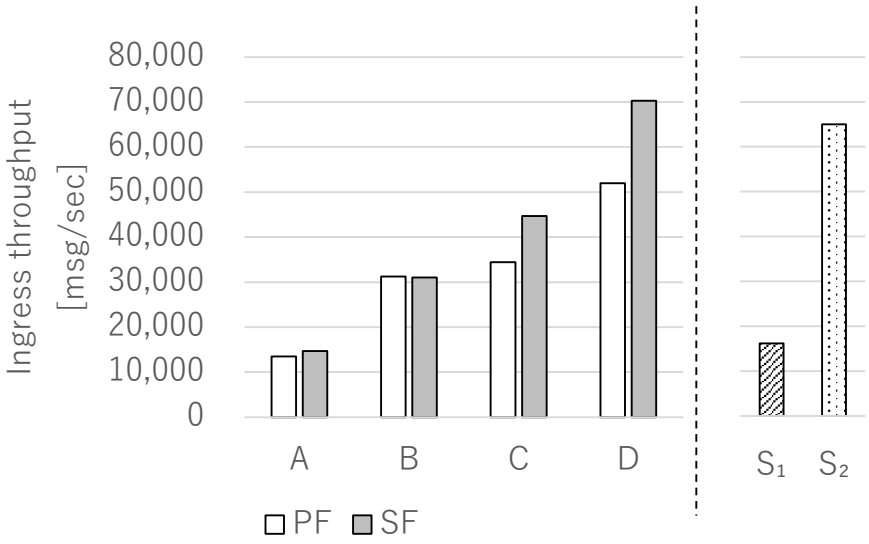

Figure 5.13: Evaluation of ILDM-based cooperation: ingress throughput.

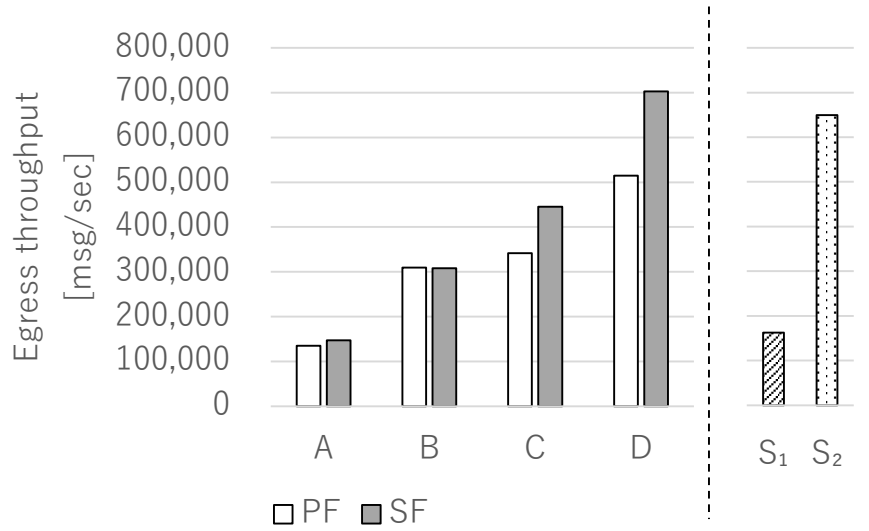

Figure 5.14: Evaluation of ILDM-based cooperation: egress throughput.

By comparing patterns A and S_1, we can see that the overhead of using an ILDM node was suppressed to approximately 10 percent. Results of pattern B, C, and D indicate that ILDM-based cooperation can provide better throughput compared with

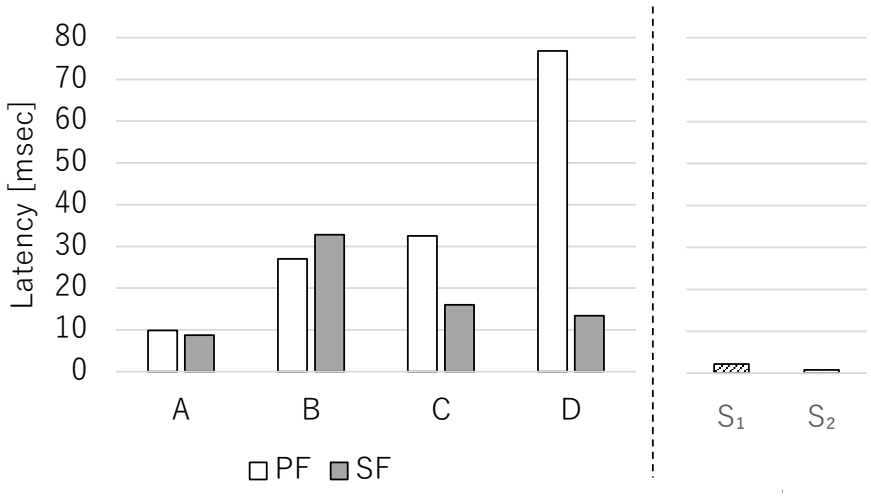

Figure 5.15: Evaluation of ILDM-based cooperation: latency.

using a single broker. Especially in pattern D with SF method, the throughput overtook the case of using a single broker on type S_2 server. Since the spec of type S_2 is quite higher than type S_1, this is an impressive result.

It can be said that locality of placing clients affects the performance, by comparing patterns B, C and D. High locality made throughput larger, especially with SF method. This is due to the characteristic of SF method described in Section 5.3.4. Considering edge-heavy data, having high locality of utilization, such tendency could be effective.

The fact that SF method achieves higher throughput than PF method also indicates that the less messages required for processing a one publish message, the more throughput we can get, as mentioned in Section 3.4.

Figure 5.15 shows the result of latency. Here also the results of single Mosquitto broker are depicted again for comparison. Basically the patterns using multiple brokers are inferior, because a PUBLISH message is forwarded with multi-hop until it arrives at corresponding subscribers.

Patterns A shows approximately 10 msec. Although this is larger than S_1, the result is considered not to impair the effect of reducing latency in the edge-based architecture, since RTT between IoT devices and data centers could be over 100 msec if it across different countries.

In pattern B, both cases of PF and SF seem to have the same load of throughput.

Therefore, the latency of SF method is a little longer due to its complicated processing compared to PF method probably. On the other hand, pattern C and D show that latency of PF method is longer than that of SF method. The reason for this is considered that more redundant messages are propagated in PF method compared to SF method. Pattern D is the case with the highest throughput, so that brokers and ILDM nodes running with PF method tend to be busy and take much time for handling PUBLISH messages.

In these measurements, the loss rate was zero for all patterns.

5.7 Conclusion of This Chapter

In this chapter, we proposed a novel mechanism called ILDM which enables heterogeneous MQTT brokers to cooperate with each other. The APIs provided by ILDM enable to develop a variety of cooperation algorithms easily. Two basic algorithms, PF and SF, and a practical benchmark method for MQTT brokers were also presented. The benchmark method ensures that the error ratios of performance are no more than 5 percent.

We evaluated the feasibility of ILDM with the benchmark method. By connecting five brokers via ILDM, the throughput increases approximately two to four times than using a Mosquitto broker alone. This result indicates that the architecture based on edge brokers is useful for reducing consumption of cloud resources.

By using the APIs of ILDM, we can easily implement the method proposed in Section 3 as a cooperation algorithm. The study in this chapter shows that the distributed pub/sub middleware we are aiming at can be developed for actual environments, i.e., it can support a practical protocol and have the flexibility for the heterogeneity of edge environments.

6

Conclusion

Considering the coming of large scale smart services composed of cooperative things, scalable and adaptive techniques for exchanging messages between devices are indispensable. In this dissertation, we explained our studies aiming to provide techniques required for realizing the distributed pub/sub middleware in the edge-based architecture.

At first, we presented a novel method of topic-based pub/sub messaging using Skip Graph. The proposed method is effective for getting high throughput, since it can suspend publishing by detecting the absence of subscribers and prevent the excessively long forwarding path. From the results of simulation experiments, we confirmed the superiority in comparison with Scribe.

Subsequently, we discussed about the latency. We first proposed SFB which is a routing algorithm for range queries in Skip Graph. SFB can reduce the number of hops and messages compared to existing methods. Second, we focused on how to assign user end clients to each broker. We proposed two assignment methods: the intensive assignment and the extensive assignment. By formulating the distribution delay time for each method, we derived the difference of the delay time between the two methods.

Finally, we gave consideration for developing the proposed method as a middleware

with the capability of a practical protocol. We proposed a mechanism called ILDM which enables heterogeneous MQTT brokers to cooperate with each other. The APIs provided by ILDM enable to develop a variety of cooperation algorithms easily. In this dissertation, we introduced two basic algorithms; PF and SF. A practical benchmark method for MQTT brokers were also presented, and we evaluated the feasibility of ILDM. By connecting five brokers via ILDM, the throughput increases approximately two to four times than using a Mosquitto broker alone.

By these studies, we clarified the feasibility of an architecture using edge brokers. As mentioned in Section 1, large-scale future services are considered to require high throughput and low latency. Different from the centralized architecture, our techniques enable to compose an autonomous distributed system without a single point of failure and a performance bottleneck, so that they have quite high scalability. Even in comparison with existing decentralized techniques, our techniques are superior regarding the above requirements since they have unique features such as the suspendability and the smaller average number of hops.

We believe that the studies in this dissertation accelerate the development of a wide variety of applications of information technologies which make our society smarter and more affluent.

References

[**Aguilera et al. 1999**] Marcos K. Aguilera, Robert E. Strom, Daniel C. Sturman, Mark Astley, and Tushar D. Chandra. Matching Events in a Content-based Subscription System. In *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing*, pp. 53–61, 1999.

[**Akiyama et al. 2014**] Toyokazu Akiyama, Yukiko Kawai, Katsuyoshi Iida, Jianwei Zhang, and Yuhki Shiraishi. Proposal for a New Generation SDN-aware Pub/Sub Environment. In *Proceedings of the IARIA International Conference on Networks*, pp. 210–214, 2014.

[**Akiyama et al. 2016**] Toyokazu Akiyama, Yuuichi Teranishi, Ryohei Banno, Katsuyoshi Iida, and Yukiko Kawai. Scalable Pub/Sub System Using OpenFlow Control. *Journal of Information Processing*, vol. 24, no. 4, pp. 635–646, 2016.

[**Al-Fuqaha et al. 2015**] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, vol. 17, no. 4, pp. 2347–2376, 2015.

[**Amazon Simple Notification Service**] Amazon Simple Notification Service. ***aws.amazon.com/sns*** (accessed: Apr. 26, 2018).

[**Aspnes and Shah 2007**] James Aspnes and Gauri Shah. Skip Graphs. *ACM Transactions on Algorithms*, vol. 3, no. 4, pp. 37:1–37:25, 2007.

[**Balasooriya et al. 2005**] Janaka Balasooriya, Mohini Padhye, Sushil K. Prasad, and Shamkant B. Navathe. BondFlow: A System for Distributed Coordination of Workflows over Web Services. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium*, pp. 121a–121a, 2005.

[**Banks and Gupta 2015**] Andrew Banks and Rahul Gupta. MQTT Version 3.1.1 Plus Errata 01, December 2015.

[**Banno et al. 2014**] Ryohei Banno, Susumu Takeuchi, Michiharu Takemoto, Tetsuo Kawano, Takashi Kambayashi, and Masato Matsuo. A Distributed Topic-Based Pub-/Sub Method for Exhaust Data Streams towards Scalable Event-Driven Systems. In *Proceedings of the IEEE Annual Computer Software and Applications Conference*, pp. 311–320, 2014.

[**Banno et al. 2015a**] Ryohei Banno, Tomoyuki Fujino, Susumu Takeuchi, and Michiharu Takemoto. SFB: A Scalable Method for Handling Range Queries on Skip Graphs. *IEICE Communications Express*, vol. 4, no. 1, pp. 14–19, 2015.

[**Banno et al. 2015b**] Ryohei Banno, Tetsuo Kawano, Susumu Takeuchi, Michiharu Takemoto, and Masato Matsuo. Comparison of Subscriber Assignment Methods on Scalable Distributed Pub/Sub Systems. In *Proceedings of the Asia-Pacific Conference on Communications*, pp. 632–632, 2015.

[**Banno et al. 2015c**] Ryohei Banno, Susumu Takeuchi, Michiharu Takemoto, Tetsuo Kawano, Takashi Kambayashi, and Masato Matsuo. Designing Overlay Networks for Handling Exhaust Data in a Distributed Topic-based Pub/Sub Architecture. *Journal of Information Processing*, vol. 23, no. 2, pp. 105–116, 2015.

[**Banno et al. 2017**] Ryohei Banno, Jingyu Sun, Masahiro Fujita, Susumu Takeuchi, and Kazuyuki Shudo. Dissemination of Edge-Heavy Data on Heterogeneous MQTT Brokers. In *Proceedings of the IEEE International Conference on Cloud Networking*, pp. 1–7, 2017.

[**Beltran et al. 2007**] Alejandra Gonzalez Beltran, Paul Sage, and Peter Milligan. Skip Tree Graph: a Distributed and Balanced Search Tree for Peer-to-Peer Networks. In *Proceedings of the International Conference on Communications*, pp. 1881–1886, 2007.

[**Beltran et al. 2008**] Alejandra Gonzalez Beltran, Peter Milligan, and Paul Sage. Range Queries Over Skip Tree Graphs. *Computer Communications*, vol. 31, no. 2, pp. 358–374, 2008.

[**Bharambe et al. 2004**] Ashwin R. Bharambe, Mukesh Agrawal, and Srinivasan Seshan. Mercury: Supporting Scalable Multi-attribute Range Queries. In *Proceedings of the Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 353–366, 2004.

[**Castro et al. 2002**] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. SCRIBE: A Large-scale and Decentralized Application-level Multicast Infrastructure. *IEEE Journal on Selected Areas in communications*, vol. 20, no. 8, pp. 1489–1499, 2002.

[**Chen et al. 2014**] Whei-Jen Chen, Rahul Gupta, Valerie Lampkin, Dale M. Robertson, and Nagesh Subrahmanyam. Responsive Mobile User Experience Using MQTT and IBM MessageSight, March 2014.

[**Chockler et al. 2007**] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Constructing Scalable Overlays for Pub-Sub with Many Topics. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, pp. 109–118, 2007.

[**de Bruijn and Erdős 1948**] Nicolaas Govert de Bruijn and Paul Erdős. On a Combinatorial Problem. *Proceedings of the Section of Sciences of the Koninklijke Nederlandse Akademie van Wetenschappen te Amsterdam*, vol. 51, no. 10, pp. 1277–1279, 1948.

[**Deakin 2015**] Nigel Deakin. Java Message Service: The JMS API is an API for accessing enterprise messaging systems from Java programs (Version 2.0 revision a), March 2015.

[**Dynomite**] Dynomite. ***github.com/Netflix/dynomite*** (accessed: Apr. 26, 2018).

[**Eugster et al. 2003**] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, vol. 35, no. 2, pp. 114–131, 2003.

[**FIPS PUB180-1 1995**] FIPS PUB180-1. *Secure Hash Standard*. National Institute of Science and Technology, April 1995.

[**Google Cloud pub/sub**] Google Cloud pub/sub. ***cloud.google.com/pubsub*** (accessed: Apr. 26, 2018).

[**Harvey et al. 2003**] Nicholas J. A. Harvey, John Dunagan, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. SkipNet: A Scalable Overlay Network with Practical Locality Properties. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, pp. 9–23, 2003.

[**HiveMQ**] HiveMQ. ***www.hivemq.com*** (accessed: Apr. 26, 2018).

[**Hodges et al. 2013**] Steve Hodges, Stuart Taylor, Nicolas Villar, and James Scott. Prototyping Connected Devices for the Internet of Things. *IEEE Computer*, vol. , pp. 26–34, 2013.

[**Hojo et al. 2016**] Masashi Hojo, Ryohei Banno, and Kazuyuki Shudo. FRT-Skip Graph: A Skip Graph-style Structured Overlay based on Flexible Routing Tables. In *Proceedings of the IEEE Symposium on Computers and Communication*, pp. 657–662, 2016.

[**Huebsch 2008**] Ryan Jay Huebsch. *PIER: Internet Scale P2P Query Processing with Distributed Hash Tables*. PhD thesis, EECS Department, University of California, Berkeley, May 2008.

[**Hunkeler et al. 2008**] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. MQTT-S — A Publish/Subscribe Protocol for Wireless Sensor Networks. In *International Conference on Communication Systems Software and Middleware*, pp. 791–798, January 2008.

[**Jokela et al. 2009**] Petri Jokela, András Zahemszky, Christian Esteve Rothenberg, Somaya Arianfar, and Pekka Nikander. LIPSIN: Line Speed Publish/Subscribe Internetworking. In *Proceedings of the ACM SIGCOMM Conference on Data Communication*, pp. 195–206, 2009.

[**Kaachoek and Karger 2003**] M. Frans Kaachoek and Dvid R. Karger. Koorde: A Simple Degreeoptimal Distributed Hash Table. *Peer-to-Peer Systems II*, vol. , pp. 98–107, 2003.

[**Kawaguchi et al. 2016**] Takafumi Kawaguchi, Ryohei Banno, Masashi Hojo, and Kazuyuki Shudo. Self-Refining Skip Graph: A Structured Overlay Approaching to Ideal Skip Graph. In *Proceedings of the IEEE Annual Computer Software and Applications Conference*, pp. 377–378, 2016.

[**Konishi et al. 2008**] Yuji Konishi, Mikio Yoshida, Susumu Takeuchi, Yuuichi Teranishi, Kaname Harumoto, and Shinji Shimojo. An Extension of Skip Graph to Store Multiple Keys on Single Node. *Journal of Information Processing Society of Japan* (in Japanese), vol. 49, no. 9, pp. 3223–3233, 2008.

[**Kreps et al. 2012**] Jay Kreps, Neha Narkhede, and Jun Rao. Kafka: A Distributed Messaging System for Log Processing. In *Proceedings of the ACM SIGMOD Workshop on Networking Meets Databases*, 2012.

[**Light 2017**] Roger A. Light. Mosquitto: Server and Client Implementation of the MQTT Protocol. *Journal of Open Source Software*, vol. 2(13), no. 265, pp. 1–2, 2017.

[**Manku et al. 2003**] Gurmeet Singh Manku, Mayank Bawa, and Prabhakar Raghavan. Symphony: Distributed Hashing in a Small World. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*. USENIX Association, 2003.

[**Manyika et al. 2011**] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, and Angela Hung Byers. *Big Data: The Next Frontier for Innovation, Competition, and Productivity*. McKinsey Global Institute, 2011.

[**Maymounkov and Mazieres 2002**] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, pp. 53–65, 2002.

[**Mizutani et al. 2012**] Kimihiro Mizutani, Toru Mano, Osamu Akashi, and Kensuke Fukuda. Efficient Query Bundling Mechanism in a DHT Network. In *Proceedings of the IEEE Global Communications Conference*, pp. 2695–2700, 2012.

[**Moquette**] Moquette. ***github.com/andsel/moquette*** (accessed: Apr. 26, 2018).

[**Node-RED**] Node-RED. ***nodered.org*** (accessed: Apr. 26, 2018).

[**Okanohara et al. 2013**] Daisuke Okanohara, Shohei Hido, Nobuyuki Kubota, Yuya Unno, and Hiroshi Maruyama. Krill: An Architecture for Edge Heavy Data. In *Proceedings of the Third Workshop on Architectures and Systems for Big Data*, 2013.

[**Plaxton et al. 1999**] C. Greg Plaxton, Rajmohan Rajaraman, and Andrea Werneck Richa. Accessing Nearby Copies of Replicated Objects in a Distributed Environment. *Theory of Computing Systems*, vol. 32, no. 3, pp. 241–280, 1999.

[**Pugh 1990**] William Pugh. Skip Lists : A Probabilistic Alternative to Balanced Trees. *Communications of the ACM*, vol. 33, no. 6, pp. 668–676, 1990.

[**Ramabhadran et al. 2004**] Sriram Ramabhadran, Sylvia Ratnasamy, Joseph M. Hellerstein, and Scott Shenker. Prefix Hash Tree: An Indexing Data Structure over Distributed Hash Tables. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, p. 368, 2004.

[**Ratnasamy et al. 2001a**] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-addressable Network. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communications*, pp. 161–172, 2001.

[**Ratnasamy et al. 2001b**] Sylvia Ratnasamy, Mark Handley, Richard Karp, and Scott Shenker. Application-Level Multicast using Content-Addressable Networks. In *Proceedings of the International COST264 Workshop on Networked Group Communication*, pp. 14–29, 2001.

[**Rivest 1992**] Ronald Linn Rivest. The MD5 Message-Digest Algorithm (RFC 1321), April 1992.

[**Rowstron and Druschel 2001**] Antony Rowstron and Peter Druschel. Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pp. 329–350, 2001.

[**Setty et al. 2012**] Vinay Setty, Maarten Van Steen, Roman Vitenberg, and Spyros Voulgaris. PolderCast: Fast, Robust, and Scalable Architecture for P2P Topic-based Pub/Sub. In *Proceedings of the International Middleware Conference*, pp. 271–291, 2012.

[**Shi et al. 2016**] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, vol. 3, no. 5, pp. 637–646, October 2016.

[**Shudo 2017**] Kazuyuki Shudo. Message Bundling on Structured Overlays. In *Proceedings of the IEEE Symposium on Computers and Communications*, pp. 424–431, 2017.

[**Snyder et al. 2008**] Bruce Snyder, Dejan Bosanac, and Rob Davies. Introduction to Apache ActiveMQ, August 2008.

[**Stoica et al. 2001**] Ion Stoica, Rovert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications. *ACM SIGCOMM Computer Communication Review*, vol. 31, no. 4, pp. 149–160, 2001.

[**SurgeMQ**] SurgeMQ. ***github.com/influxdata/surgemq*** (accessed: Apr. 26, 2018).

[**Teranishi 2009**] Yuichi Teranishi. PIAX: Toward a Framework for Sensor Overlay Network. In *Proceedings of the IEEE Consumer Communications and Networking Conference*, pp. 1–5, 2009.

[**Teranishi et al. 2017**] Yuuichi Teranishi, Takashi Kimata, Hiroaki Yamanaka, Eiji Kawai, and Hiroaki Harai. Dynamic Data Flow Processing in Edge Computing Environments. In *Proceedings of the IEEE Annual Computer Software and Applications Conference*, pp. 935–944, July 2017.

[**Videla and Williams 2012**] Alvaro Videla and Jason J. W. Williams. *RabbitMQ in Action: Distributed Messaging for Everyone*. Manning Publications, April 2012.

[**Vinoski 2006**] Steve Vinoski. Advanced Message Queuing Protocol. *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, November 2006.

[**Vuillemin 1978**] Jean Vuillemin. A Data Structure for Manipulating Priority Queues. vol. 21, pp. 309–315, April 1978.

[**Watts and Strogatz 1998**] Duncan James Watts and Steven Henry Strogatz. Collective Dynamics of 'Small-world' Networks. *Nature*, vol. 393, pp. 440–442, 1998.

[**Welhuis**] Annouck Welhuis. Twitter and the pareto principle. ***www.annouckwelhuis.nl/twitter-and-the-pareto-principle-2*** (Accessed: Apr. 26, 2018).

[**Zhang and Hu 2003**] Rongmei Zhang and Y. Charlie Hu. Borg: a Hybrid Protocol for Scalable Application-level Multicast in Peer-to-Peer Networks. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 172–179, 2003.

[**Zhao et al. 2004**] Ben Y. Zhao, Ling Huang, Jeremy Stribling, Sean C. Rhea, Anthony D. Joseph, and John D. Kubiatowicz. Tapestry: A Resilient Global-scale Overlay for Service Deployment. *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, 2004.

[**Zhao et al. 2013**] Ye Zhao, Kyungbaek Kim, and Nalini Venkatasubramanian. DYNATOPS: A Dynamic Topic-based Publish/Subscribe Architecture. In *Proceedings of the International Conference on Distributed Event Based Systems*, pp. 75–86, 2013.

[**Zhuang et al. 2001**] Shelley Q. Zhuang, Ben Y. Zhao, Anthony D. Joseph, Randy H. Katz, and John D. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-area Data Dissemination. In *Proceedings of the International Workshop on Network and Operating Systems Support for Digital Audio and Video*, pp. 11–20, 2001.

[**Zhuge and Feng 2008**] Hai Zhuge and Liang Feng. Distributed Suffix Tree Overlay for Peer-to-Peer Search. *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 2, pp. 276–285, 2008.