symmetry

MDPI

Article

Efficient Edge-Cloud Publish/Subscribe Broker Overlay Networks to Support Latency-Sensitive Wide-Scale IoT Applications

Van-Nam Pham, VanDung Nguyen, Tri D. T. Nguyen and Eui-Nam Huh *

Department of Computer Science and Engineering, Kyung Hee University, Yogin-si 17104, Korea; nampv@khu.ac.kr (V.-N.P.); ngvandung85@khu.ac.kr (V.N.); tringuyendt@khu.ac.kr (T.D.T.N.)

* Correspondence: johnhuh@khu.ac.kr; Tel.: +82-031-201-3778

Received: 23 November 2019; Accepted: 13 December 2019; Published: 18 December 2019

Abstract: Computing services for the Internet-of-Things (IoT) play a vital role for widespread IoT deployment. A hierarchy of Edge-Cloud publish/subscribe (pub/sub) broker overlay networks that support latency-sensitive IoT applications in a scalable manner is introduced. In addition, we design algorithms to cluster edge pub/sub brokers based on topic similarities and geolocations to enhance data dissemination among end-to-end IoT devices. The proposed model is designed to provide low delay data dissemination and effectively save network traffic among brokers. In the proposed model, IoT devices running pub/sub client applications periodically send collected data, organized as a hierarchy of topics, to their closest edge pub/sub brokers. Then, the data are processed/analyzed at edge nodes to make controlling decisions promptly replying to the IoT devices and/or aggregated for further delivery to other interested edge brokers or to cloud brokers for long-term processing, analysis, and storage. Extensive simulation results demonstrate that our proposal achieves the best data delivery latency compared to two baseline schemes, a classical Cloud-based pub/sub scheme and an Edge-Cloud pub/sub scheme. Considering the similar Edge-Cloud technique, the proposed scheme outperforms PubSubCoord-alike in terms of relay traffic ratio among brokers. Therefore, our proposal can adapt well to support wide-scale latency-sensitive IoT applications.

Keywords: broker-based publish/subscribe; topic similarity; geolocation awareness; IoT applications; latency-sensitive; distributed pub/sub systems

1. Introduction

Recent advancements in Internet-of-Things (IoT) technologies have enabled various applications in many domains, such as agriculture, healthcare, transportation, industrial automation, and smart homes [1]. In these applications, IoT devices, for example, wearables, smart meters, and smart appliances, will create an unprecedented amount of data that needs to be transferred, stored, and analyzed at proper places. Cisco estimates that, with the widespread adoption of the IoT, by 2021 847 zettabytes (ZB) of IoT data will be generated per year [2]. A large part of that data will be ephemeral and approximately 7.2 ZB will be stored in the Cloud [2]. In addition, data are primarily generated near the edge of the network and will be processed by applications deployed at the edge to provide fast reactions. A portion of the data and information extracted will be uploaded to the Cloud. Designing a communication platform to provide low latency and high scalability for such IoT applications is a challenging.

Cloud computing platforms, typically characterized by on-demand self-service, resource pooling, and rapid elasticity, have proved to be a useful and economic alternatives for various applications, such as collaboration and computing-intensive applications [3,4]. However, latency-sensitive applications

restrict low transmission delay and require quality of service conditions; thus, these applications may not be suitable to be solely deployed in the Cloud. Moreover, deploying many wide-scale IoT projects requires the support of computing platforms with specialized characteristics, such as geo-distribution and location awareness. Conversely, the Fog Computing paradigm can meet these requirements and fit well for IoT applications by bringing computing, storage, and networking functions closer to the edge [4]. In addition, interplay between Cloud Computing and Fog/Edge Computing is worthwhile for long-term data management and IoT application analytics [5]. Thus, a communication paradigm that can take advantages of the Cloud and the Fog will be fruitful for IoT.

The Publish/Subscribe (pub/sub) paradigm has been used as a powerful communication protocol to develop a wide range of distributed applications. In this paradigm, communicating entities are loosely coupled in terms of time, space, and synchronization; thus, the paradigm can support useful and flexible characteristics, such as anonymity, many-to-many, and asynchronicity, that are critical for distributed systems [6]. This makes the pub/sub paradigm highly suited for complex, wide-scale information diffusion systems such as *Spotify*, *Twitter*, and *Facebook* [7]. For instance, *Spotify* [8] is an international music streaming service that uses pub/sub paradigm to provide various features for social interactions. In *Spotify* topic-based pub/sub system, users can subscribe topics under types of Friends, Playlists, and Artist pages. When users subscribe to an Artist page, they will receive future notifications related to the artist such as new album releases. Therefore, many IoT applications have relied on pub/sub systems for data dissemination [9,10]. In addition, topic-based pub/sub (TBPS) systems were mainly deployed for message delivery in IoT systems [11]. To support connectivity between IoT devices and applications, a lightweight messaging protocol, such as MQTT [12], is required. In addition, a message broker/server is deployed to facilitate data exchange among IoT devices. The applications access the IoT data and control the devices via this server using provided APIs. This method is highly suited to LAN environments but does not scale for application to wide-scale IoT projects where there are a very large number of geo-distributed IoT devices and a large number of brokers [13]. Furthermore, the centralized deployment of pub/sub brokers in the Cloud will increase latency for IoT applications. Consequently, there is an evident need to develop advanced coordination schemes to organize and scale these brokers effectively.

In this paper, we present an Edge–Cloud pub/sub broker model for data dissemination in wide-scale IoT projects. In the proposed model, in order to take advantages of the Cloud and the Fog, brokers are placed closer to the edge networks to reduce data delivery latency and are installed in the Cloud for long-term data analytics for the applications. The main contributions of this paper are summarized as follows.

- We propose a hierarchy of Edge-Cloud publish/subscribe (pub/sub) broker overlay networks that support latency-sensitive IoT applications in a scalable manner.
- We devise a collaborative scheme to enable pub/sub brokers to efficiently find, inform, and link a large number of topics in wide-scale IoT projects.
- We design algorithms to cluster edge pub/sub brokers based on topic similarities and geolocations to enhance data dissemination among end-to-end IoT devices.
- Extensive simulations results demonstrate that the proposed scheme can provide low delay data dissemination and effectively save network traffic among brokers that are critical for wide-scale IoT applications.

The remainder of this paper is organized as follows. Section 2 surveys works related to pub/sub systems for wide-scale applications. Section 3 outlines the proposed pub/sub model for message delivery in IoT projects. In Section 4, the coordination scheme and the details of algorithms to cluster pub/sub brokers are presented, and, in Section 5, we provide simulation results and a discussion. Conclusions and suggestions for future work are presented in Section 6.

2. Related Work

Firebase Cloud Messaging (FCM) [14] is a cross-platform messaging solution that can play as a development platform for IoT applications. FCM allows application developers to send notifications and messages across many IoT applications that need to collect data and deliver low latency content. An implementation of FCM comprises two main components to build, transport, and receive messages: a trusted server environment that supports the FCM server protocols and client applications. Another famous framework to address the emerging needs of IoT is IoTivity [15]. The IoTivity is an open source project with the goal to create a new standard that can connect billions of wired and wireless devices. In this framework, data are collected from IoT devices and ultimately stored in IoTivity Global Cloud using pub/sub brokers that might cause long delay for latency-sensitive IoT applications.

To support message delivery in wide-scale applications, many studies have proposed solutions by providing protocols and algorithms to build scalable peer-to-peer TBPS systems [7,16,17]. Rahimian et al. [7] proposed a hybrid TBPS overlay system that offers low relay messages among peers and scales well with a large number of peers and topics. In this system, peers with similar subscriptions are clustered by a gossip-based unstructured sampling method. Then, these clusters are linked by a structured rendezvous routing technique. Girdzijauskas et al. [17] presented a TBPS system that is efficient and scalable in message dissemination by exploiting topic subscription correlations among peers while dynamically clustering them on a skewed distributed hash table. In addition, a routing technique, based on the underlying routing structure, was proposed to build the multicast trees for locality preservation to reduce message delivery latency. In a different method, Chockler et al. [16] introduced a distributed protocol, SpiderCast, to dynamically organize correlated workload peers into an overlay network to support TBPS communication. SpiderCast works well with a large number of nodes and a limited number of correlated subscriptions per node. In short, these methods could be applied to manage broker-based TBPS systems where groups of distributed brokers are responsible for relaying messages among publishers and subscribers (IoT devices). However, it is infeasible to apply such methods directly on IoT devices in wide-scale IoT applications due to limited device resources. In addition, these methods lack broker/peer locality-awareness to further reduce latencies for delay-sensitive applications in a scalable manner.

Another promising approach to deliver data in wide-scale IoT systems is building scalable broker-based TBPS systems, for example, by deploying pub/sub brokers, such as Dynamoth [18]. Dynamoth is scalable pub/sub middleware that can be deployed in the Cloud to provide data delivery service for data producers and consumers. Dynamoth provides a scalable, load-balancing topic routing service among brokers inside the Cloud for outside clients. However, this solution may not suit delay-sensitive IoT applications due to long delays when transferring data from clients to Cloud brokers. Another way is to incorporate strong characteristics of both Fog Computing and Cloud Computing by deploying distributed brokers closer to the clients and coordinating them with brokers located in the Cloud. In PubSubCoord [19], edge brokers are deployed at isolated networks to reduce delays, and routing servers are deployed in the Cloud to route messages among the edge network brokers. In [19], however, the authors do not exploit topic correlations among proximate edge brokers to further decrease delays. In addition, it is very important to provide efficient mechanisms for edge/fog servers to communicate and collaborate with each other to improve system performance [20]. The integration of the Fog and the Cloud with the IoT will bring many benefits to different IoT applications. Therefore, in this paper, we attempt to provide an efficient coordination scheme for distributed pub/sub brokers and algorithms to build broker overlay networks for event notifications in wide-scale IoT systems.

3. Hierarchy of Proposed Edge–Cloud Pub/Sub Broker Overlay Networks

Our model under consideration comprises a three-tier hierarchy: user devices, edge cloud, and core cloud, as shown in Figure 1. First, IoT devices, such as smart sensors/actuators, smart gadgets, and smart appliances, are deployed at the edge of the network. In addition, pub/sub client applications

are installed in these devices to periodically send sensed data to their nearby pub/sub brokers located in the edge cloud tier. Note that the clients can be either producers or consumers in a pub/sub system. Once the clients subscribe to interested topic channels, they play a role as consumers in a pub/sub system used to receive event notifications from their counterparts via their corresponding brokers. In this scheme, we assume that pub/sub clients are pre-configured to connect to their corresponding edge brokers or dynamically arranged to their nearby edge brokers by applying previously proposed techniques [21–23]. Our proposed model differs from previously proposed models [21–23] in the following aspects. First, pub/sub brokers are strategically located in edge clouds and in the core cloud, and they will link all topic channels of the system with instructions provided by their coordinators. Coordination servers, which coordinate all brokers in the system, are located in the top tier. Second, pub/sub brokers will send pub/sub servers control information to help them know which brokers currently relay which topics in the system (Section 4.2); thus, they help determine whether the number of brokers should be scaled up or down relative to demand.

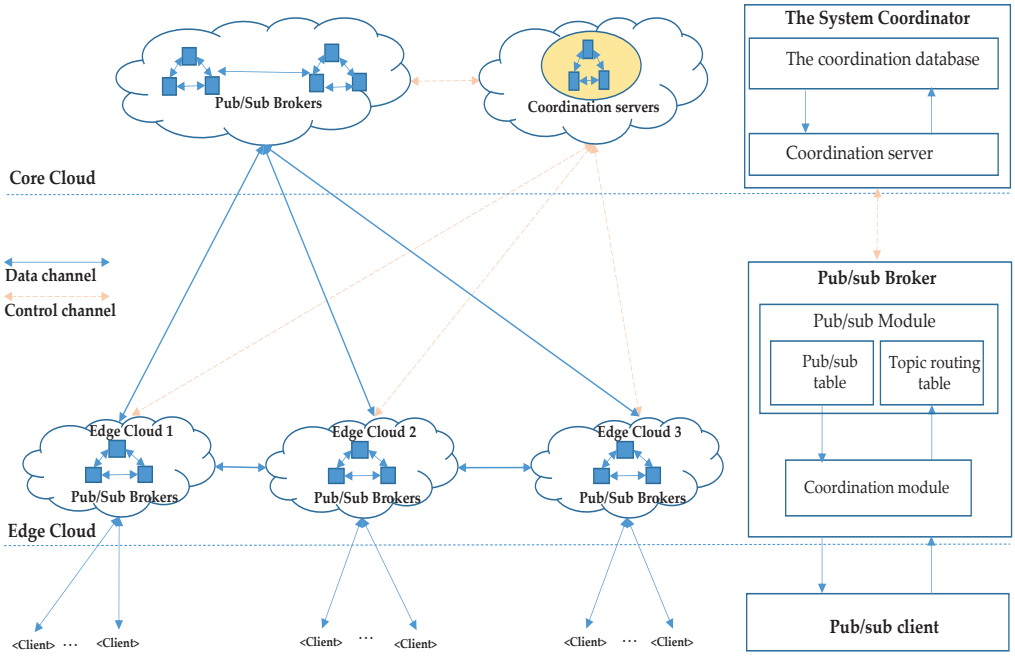

Figure 1. Pub/sub broker overlay networks with geolocation awareness and topic similarity clustering for wide-scale IoT applications.

Considering a wide-scale pub/sub system, it is necessary to design a distributed coordination service scheme, which is used to deal with coordination logic. One such approach is known as ZooKeeper. In this paper, we suggest that using ZooKeeper [24] is crucial for the following reasons. Due to the inherent nature of distributed systems, many problems can arise, such as unreliable transmissions, varied latencies, changing topologies, and heterogeneous networks. ZooKeeper can lessen these challenges for distributed application developers with its open source, distributed process coordination framework. In additional, ZooKeeper's set of APIs provides highly efficient coordination services, such as configuration management, distributed queues, locks, leader election, and group membership. As a result, a wide range of industrial applications has successfully used the ZooKeeper framework [25].

Next, to take advantages of the ZooKeeper service to build a specialized coordination service for a distributed application, an ensemble of ZooKeeper servers is installed and properly configured in our model. Depending on the coordination requirements of the application, a specialized hierarchical data node (*znodes*) structure is constructed. Consequently, application developers using the ZooKeeper client API manage these *znodes* through *read* and *write* requests to get and set synchronized data

on the *znodes* based upon the coordination logic. An application client can also register *listeners* on interested *znodes* to *watch* for changes to the nodes. The ZooKeeper server will notify the broker when events happen on the watched *znodes*. The operation and advantages of the ZooKeeper service will be applied in our model with reference to the above considerations. For more detail about the ZooKeeper coordination service, we refer interested readers to [24,25].

Figure 2 shows a specific *znodes* structure for the pub/sub system to illustrate the application of ZooKeeper in coordinating pub/sub brokers. A set of coordination servers takes responsibility for the topic discovery service of the pub/sub system. These servers elect a *leader* for task assignments and group management. Each server creates a *znode* for itself under the */CoordWorkers* namespace and solves tasks assigned in its corresponding */BrokerRequests znode*. Similarly, each pub/sub broker creates a *znode* under the */PublishSubscribeBrokers* namespace, and the leader will assign a coordination worker to provide the broker with the topic discovery service. The operational process shown in Figure 2 is explained as follows. Whenever a new pub/sub topic is detected, the broker sends a topic bridging request to its coordinator by writing the task on the coordinator's */BrokerRequests znode*. After processing the request to find either the best publish cluster or the most appropriate broker for the requester to join/subscribe the topic, the coordination worker writes the result under the requester's */CoordinatorRespones znode*. The worker also updates information about the new pub/sub broker of the topic under the */Topics znode*.

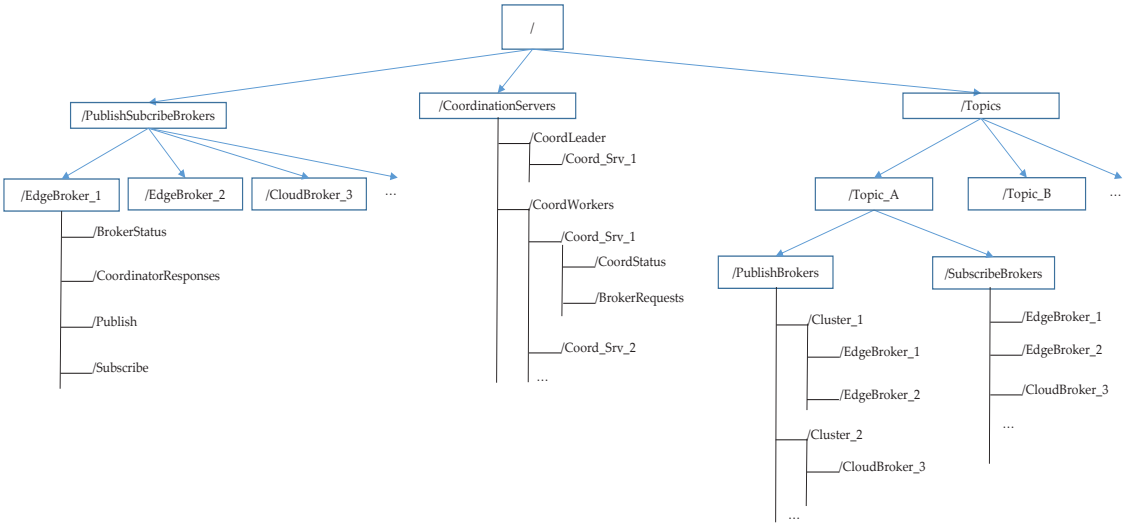

Figure 2. Illustration of the specific data tree in ZooKeeper.

4. Coordination Scheme and Geolocation Awareness Topic Similarity Clustering Algorithms

To support an increasing number of topics for a large number of clients where data are periodically exchanged in geographically distributed IoT applications, how the pub/sub system implements an advanced coordination scheme for topic discovery among brokers is critical. Furthermore, algorithms for dynamic clustering pub/sub brokers must guarantee data delivery latency requirements. Therefore, we devise an effective coordination scheme and clustering algorithms to tackle these issues by efficiently clustering all brokers based on both geolocation awareness and topic similarities to be represented as a single data delivery domain.

4.1. Coordination Scheme for Pub/Sub Brokers

In the proposed pub/sub system, whenever a broker receives a new topic subscription from one of its clients, the broker, with the help of the coordinator, has to find the most suitable broker of the requested topic to register the interesting channel. Similarly, when receiving a new topic publication, the responsible broker needs to be linked to the closest possible cluster of brokers publishing events

on the topic. The core interactions among pub/sub brokers and their coordinators are described as follows.

- Brokers promptly inform their coordination servers about new topics published or subscribed by their clients.
- Information about which brokers pub/sub what topics is stored in the coordination database. Coordination servers will make clustering decisions based on topic similarities and proximity coordinates among relevant brokers.
- Coordination servers maintain information about a set of publish broker overlay networks. The set comprises one overlay network for each topic that can include many publish broker clusters that forwards the topic's published events. Furthermore, broker leaders of these clusters will connect to each other to bridge the topic in the entire system.
- When notified of a new publish broker of a topic, the coordinator assigns the new publish broker to the "nearest" publish broker cluster and sends the updated cluster membership information to all cluster members. In each cluster, brokers exchange topic event notifications directly. The broker leader, however, is the only broker that forwards topic event notifications to other clusters.
- A subscribe broker that has client(s) interested in a topic needs to consult the coordinator to determine which publish broker it should connect with to receive all related notifications. The coordinator, in turn, first determines the publish broker cluster nearest to the subscribe broker based on the brokers' topic similarities and coordinates. Then, it selects the publish broker in that cluster with the highest utility score based on topic similarities and topic loads, and returns the selected broker's address to the subscribe broker.

Figure 3 provides an example of how publish/subscribe messages are delivered among brokers in the system. For topic A, in the example, there are three publishers ($P1$, $P2$, and $P3$) that will periodically send related event notifications and three subscribers ($S1$, $S2$, and $S3$) interested in the topic. Similarly, $P4$, $P5$, and $P6$ are publishers of topic B, and $S4$ and $S5$ are subscribers of topic B. Two publish broker clusters are established for topics A and B ($\{EB1, EB2, EB3\}$ and $\{EB4, EB5, EB6\}$). The broker leaders for topic A and B are $EB2$ and $EB4$, respectively. Within a cluster, brokers directly exchange data with each other. An interested broker from outside has to contact one of the publish brokers in the best fit cluster to receive all related events. For example, using control information received from the coordinator, the edge broker $EB5$ subscribes to topic A from $EB2$ to receive all published messages about the topic.

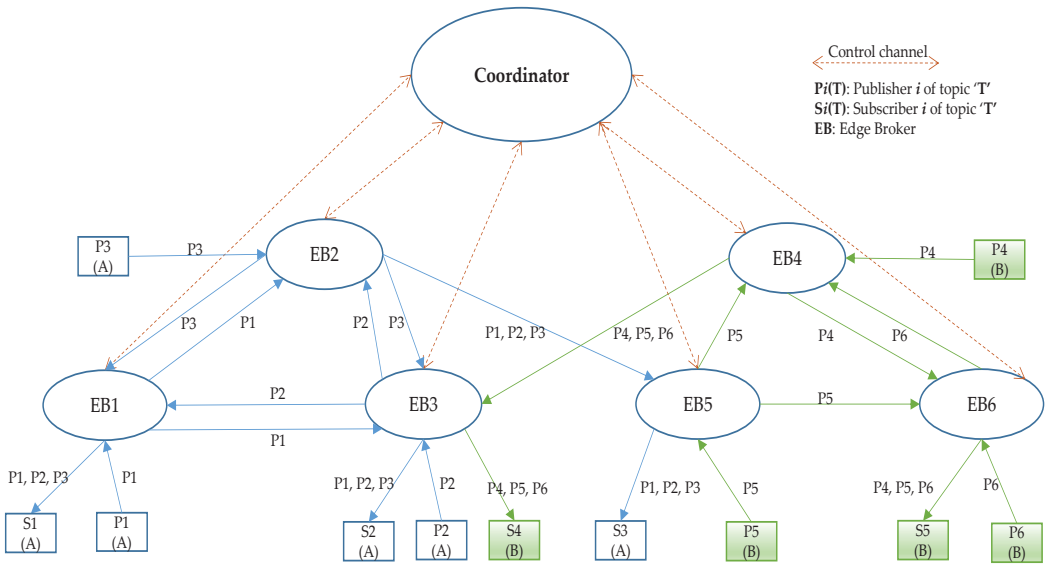

Figure 3. Illustration of message delivery in the system.

4.2. Operation of Proposed System

In the proposed pub/sub model, we deploy pub/sub brokers near IoT devices to support low delay data delivery. We also place pub/sub brokers in the Cloud to collect data for long-term data management and analytics. This subsection describes the main schemes and algorithms executed by coordination servers and pub/sub brokers to interact with each other and forward event notifications among geographically distributed clients.

In the proposed system, pub/sub clients collect and exchange data with the help of their responsible pub/sub brokers as follows. (1) Data are structured as a set of topics. (2) Publishers of each topic collect related data and send them to their brokers. (3) Subscribers express their interests to their brokers by subscribing to corresponding topic channels to receive relevant notifications. A coordination module in each pub/sub broker connects to a coordinator to communicate control information. Coordinators must be informed of broker topic lists to efficiently cluster brokers based on topic similarities and proximity coordinates. More specifically, whenever a broker receives a new topic publication or a new topic subscription from one of its pub/sub clients, it promptly notifies the coordinator. Then, the coordinator uses the following procedures and algorithms to process the notifications.

4.2.1. Handling a New Topic *Publish* Message

According to the model shown in Figure 1, when an IoT device has a new topic, all brokers in the system are required to be informed about the topic. To do that, the new topic information is processed in the corresponding pub/sub broker, the coordinator, and updated in the database. The sequence diagram for handling a new topic publication message from a client at a pub/sub broker is shown in Figure 4. When a publisher sends a Publish message on a new topic, A, the responsible broker has to check its publish topic list to determine if the message constitutes a new topic. If the publish topic list includes topic A and the sender is a new publisher, the pub/sub broker adds the new sender to its publisher. Otherwise, the broker adds the new sender to its publisher list and sends a message to inform the coordinator that it is now a new publish broker of topic A. In turn, the coordinator queries the coordination database, selects the best fit publish cluster for the new broker according to Algorithm 1, and returns the selected cluster list of brokers back to the new broker. It also notifies all cluster members about the new publish member. Thus, all current cluster members know about the new publish broker and, consequently, they subscribe to topic A on the new broker to receive incoming event notifications. Similarly, the new broker subscribes topic A to other members of the selected cluster. Note that, in both cases, the pub/sub broker forwards the publish message to any interested subscribers.

In the following, we describe how the coordinator determines the best fit cluster for the new publish broker of the $topicID$. The coordinator calculates the average distances from the new publish broker to each publish cluster of the topic. A new publish broker is included in a cluster if the average distance between the new publish broker and existing publish clusters is less than or equal to $MAX\_DISTANCE$, i.e. 10 km. To calculate the distance between two points in the sphere, according to [26], we have: $d(x,y) = r * (haversine(\theta_2 - \theta_1)) + \cos(\theta_1) * \cos(\theta_2) * haversine(\lambda_2 - \lambda_1))$, where r is the radius of the earth (6371 km), and $x = (\lambda_1, \theta_1)$ and $y = (\lambda_2, \theta_2)$ are coordinates of two points in the sphere in decimal degrees (longitude, latitude). We convert decimal degrees to radians to be used in trigonometric functions. With $haversine(\theta) = \sin^2(\theta), \Delta\theta = \theta_2 - \theta_1, \Delta\lambda = \lambda_2 - \lambda_1$ (in radians), we can further derive:

$$\begin{aligned} a(x,y) &= \sin^2(\Delta\theta/2) + \cos(\theta_1) * \cos(\theta_2) * \sin^2(\Delta\lambda/2), \\ d(x,y) &= 2 * r * atan2(\sqrt{a(x,y)}, \sqrt{1 - a(x,y)}). \end{aligned} \tag{1}$$

Given that we know the brokers' coordinates in advance, the average distance between broker b_i and cluster C_k can be calculated as follows:

$$avg\_Distance(b_i, C_k) = \frac{1}{m}\sum_{j=1}^{m} d(b_i, b_j), C_k = \{b_1, ..., b_m\}. \quad (2)$$

For the selected clusters, the coordinator then calculates publish utility scores between the new broker and each cluster using a utility function that is similar to a previously proposed utility function [27].

$$pub\_Utility(b_i, C_k) = \frac{1}{avg\_Distance(b_i, C_k)} * \frac{\mid subs(b_i) \cap subs(C_k) \mid}{\mid subs(b_i) \cup subs(C_k) \mid}. \quad (3)$$

Here, $subs(b_i)$ denotes the broker b_i's topic subscription list, and $subs(C_k)$ denotes all subscribe topics in cluster C_k. After calculating publish utility scores, the coordinator selects the optimal cluster with the best publish utility score to return to the new member. In a sporadic density scenario, if the average distance between the new broker and all clusters is greater than $MAX\_DISTANCE$, a new cluster is created with the new broker as the leader.

Algorithm 1: Finding the best fit publish cluster for a new publish broker.

```
Function findPublishCluster(topicID, brokerID, publishTable, topicClusters):
    add (topicID, brokerID) to publishTable;
    if topicID is in topicClusters then
        initialize potentialClusters;
        for each cluster in topicClusters of topicID do
            avg_Distance(brokerID, cluster) → Equation (2);
            if avg_Distance(brokerID, cluster) ≤ MAX_DISTANCE then
                add cluster to potentialClusters;
            end
        end
        if length(potentialClusters) > 0 then
            initialize clusterUtilityScores;
            for each cluster in potentialClusters do
                pub_Utility (brokerID, cluster) → Equation (3);
                save to clusterUtilityScores;
            end
            sort clusterUtilityScores in descending order;
            add brokerID to the best cluster with the highest utility score in topicClusters;
            return the best fit cluster;
        end
    end
    create a new cluster with (topicID, brokerID) and add it to topicClusters;
    return the new cluster as the best cluster;
```

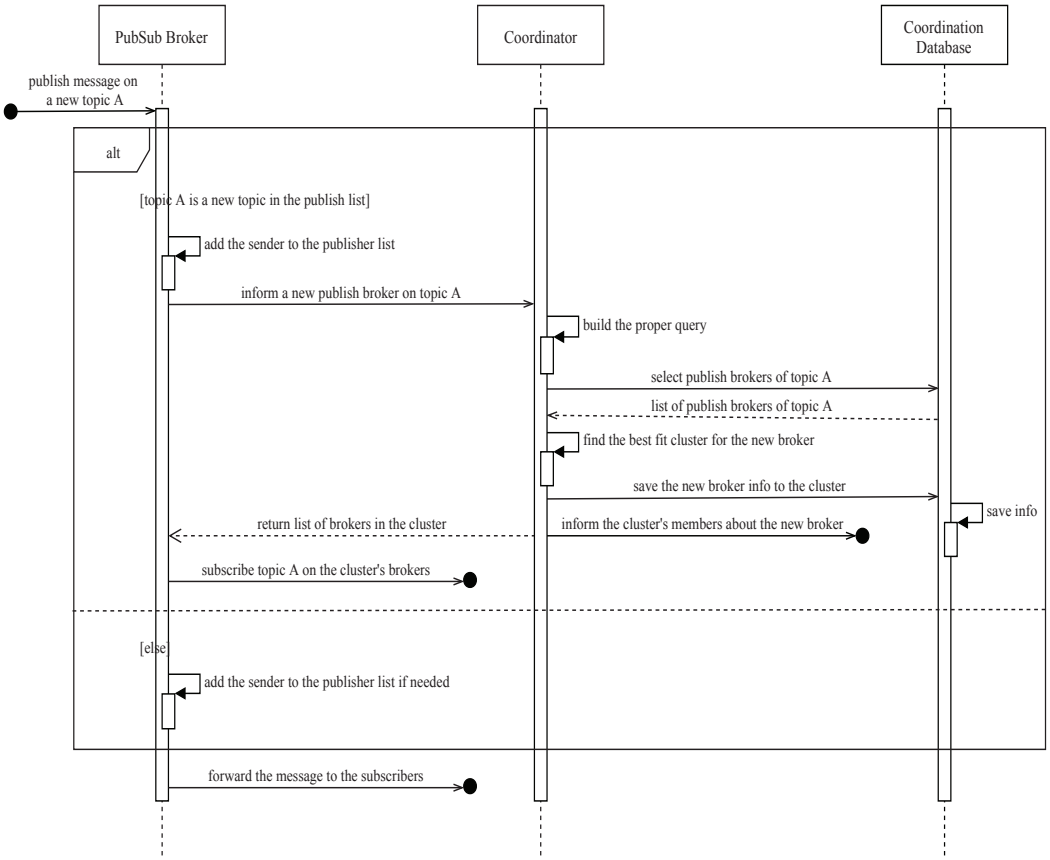

Figure 4. New topic *Publish* message handling sequence diagram.

4.2.2. Handling a New Topic *Subscribe* Message

In this section, we discuss how to handle a new topic subscription message from a client at a pub/sub broker, as shown in Figure 5. When a client subscribes to a new topic, the corresponding broker consults its coordinator on finding the best publish broker to bridge the topic. In turn, the coordinator, first, finds the best fit cluster publishing the topic, and then selects the best broker to return to the requester. The detail procedure is described as follows. When a subscriber sends a *Subscribe* message on a new topic, B, the corresponding broker must check whether topic B is a new topic in its subscribe topic list. If the subscribe topic list already includes topic B and the sender is a new subscriber, the pub/sub broker adds the new sender to its subscriber. If the subscribe topic list does not include topic B, the broker adds the client to its subscriber list and sends a message to inform the coordinator that the broker is now a new subscribe broker of topic B. Upon receiving the announcement, initially, the coordinator searches the coordination database to find the best available publish broker cluster for the new subscribe broker. Then, the coordinator selects the best publish broker in the selected cluster and returns the result to the new subscribe broker (Algorithm 2). Then, the new broker subscribes topic B on the selected publish broker to receive event notifications for the topic.

Algorithm 2 provides the logic for the coordinator to find the best fit publish broker for the new subscribe broker on an interested topic, denoted as $topicID$. If there are brokers publishing events on $topicID$, the coordinator uses formula (2) to calculate average distances from the new subscribe broker to the existing clusters to select the most suitable cluster. Then, the coordinator computes subscribe utility scores between the new broker, b_i, and each broker, b_j, in the cluster, C_k, to select the best publish broker as shown in formula (4), which is based on a previously published formula [28].

$$sub\_Utility(b_i, b_j, C_k) = \frac{\mid subs(b_i) \cap subs(b_j) \mid}{\mid subs(b_i) \mid} - \beta * \frac{\mid subs(b_j) \mid}{\mid subs(C_k) \mid}. \quad (4)$$

Here, $| subs(b_j) | / | subs(C_k) |$ represents broker b_j's topic workload in correlation with other brokers in the cluster, and β is the relative weight of workloads. In formula (4), we consider both topic subscription similarity between two brokers (b_i, b_j) and the examined broker's workload when selecting the most appropriate publish broker for the new subscribe broker. This is because the system's topic routing efficiency increases relative to the extent that two brokers share similar topics. In addition, considering the examined broker's workload helps maintain the workload balance among brokers in the cluster.

Algorithm 2: Finding the best fit publish broker for a new subscribe broker.

Function
`findPublishBroker`(*topicID*, *brokerID*, *publishTable*, *subscribeTable*, *topicClusters*, *pendingTable*)**:**
  if *topicID is in publishTable* **then**
    add (*topicID*, *brokerID*) to *subscribeTable*;
    for *each cluster in topicClusters of topicID* **do**
      avg_Distance(*brokerID*, *cluster*) → Equation (2);
    end
    select the *cluster* with the lowest average distance;
    for *each broker in the selected cluster* **do**
      sub_Utility (*brokerID*, *broker*) → Equation (4);
    end
    return the publish *broker* with the highest subscribe utility score;
  else
    add (*topicID*, *brokerID*) to the *pendingTable*;
  end

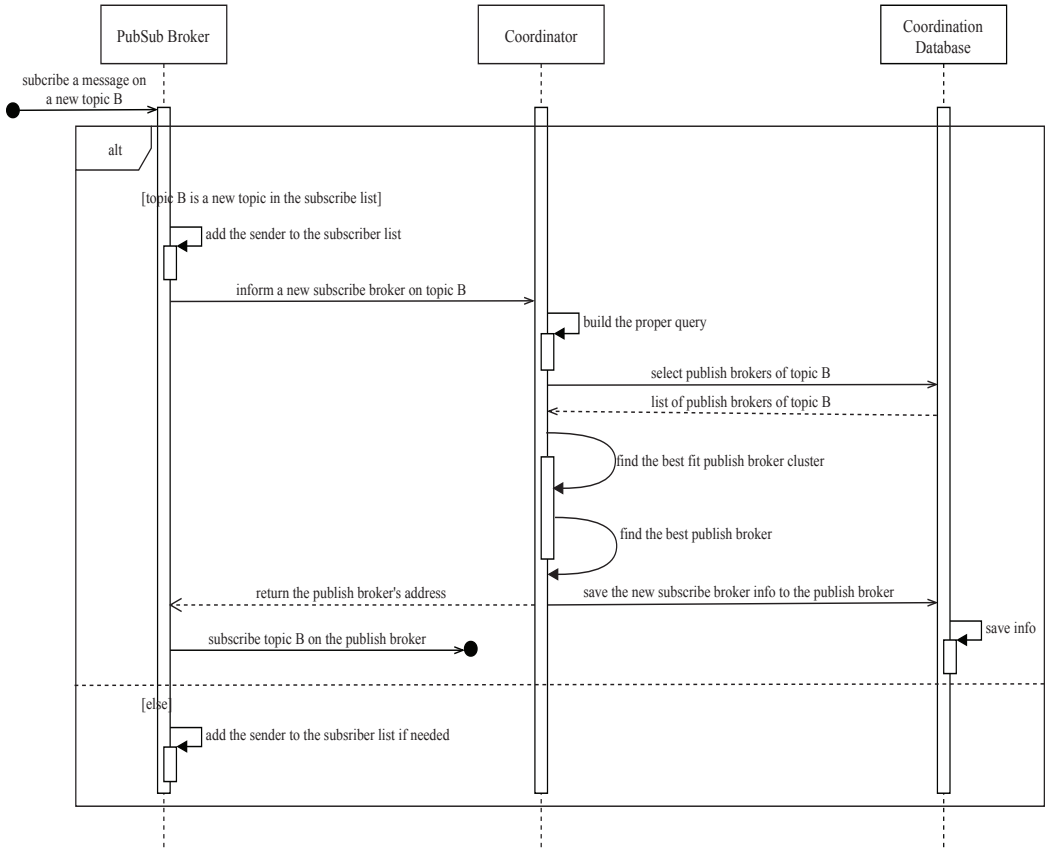

Figure 5. New topic *Subscribe* message handling sequence diagram.

5. Simulation Results and Discussion

This section describes our simulation experiments and results to validate the proposed model and algorithms. We also compare the proposed method to some analogous approaches. We simulate our pub/sub system based on Python and SimPy [29], a process-based discrete event simulation framework. In our scheme, the coordinates of pub/sub brokers used to apply clustering algorithms with locality-awareness and topic similarity are given in advance. Therefore, we used approximately 3300 Starbuck store coordinates for broker locations downloaded from [30] as a coordinate dataset. Assuming that the round-trip time (RTT) between any two brokers is a random variable where the mean is proportional to their geographical distance, and RTTs between IoT devices and their corresponding brokers are uniformly distributed with an average delay of 4 ms.

Here, we define three key performance indicators to evaluate the relevant approaches.

1. **End-to-end message delivery latency**. This value is measured from publishers to subscribers on all tested topic channels with various scenarios.
2. **Percentage of forwarded traffic among brokers**. This value is the ratio between the number of packets that are forwarded among brokers to link topics and all sent packets. It is used to measure the efficiency of bridging topics in the system.
3. **Scalability**: This property is achieved by considering average delivery latencies and percentage of forwarded traffic among brokers while applying different pub/sub schemes for wide-scale IoT scenarios.

5.1. End-to-End Message Delivery Latency with Varying Data Localities

To evaluate the effect of data locality on data delivery latency in the system, our proposal was implemented in the following system: 300 brokers were deployed with their coordinates, randomly selected from the dataset. We varied the number of topics per broker in each simulation run time: 50, 150, and 250 topics with one publisher and 10 subscribers per local topic. We also changed the local subscription rate (LSR) for each broker from 0.1 to 0.9 in steps of 0.2. For example, for a broker with 100 topics and an LSR of 0.9, 90 topics were local for its subscribers and its consumers also subscribed to 10 topics from other brokers. Each publisher sends a 64-byte packet every tick time. For each setting, we ran the simulation in 15 time steps and collected up to 7.5 million subscriber/consumer messages. In the experiment, we calculated the average delivery latency (ADL) of event notifications from producers to consumers with varying local subscription rates at brokers.

Figure 6 shows the simulation results with varying data localities. It is clear that, the more subscribers belonging to the same broker share local topics, the better (lower) the event notification delay for the entire pub/sub system improves. Specifically, in the highest case of 250 topics per broker, when the LSR increases from 0.1 to 0.9, the average delay decreases from approximately 35 ms to approximately 11.5 ms, respectively. This occurs because, when the LSR is high, a large percentage of messages are transferred from producers to consumers within the responsible brokers, which do not incur long delay dissemination among brokers. This proves that our use of edge brokers to manage pub/sub clients helps reduce end-to-end delay and is consistent with the current known trend that data are processed near where they are created. This is a very useful feature for IoT applications.

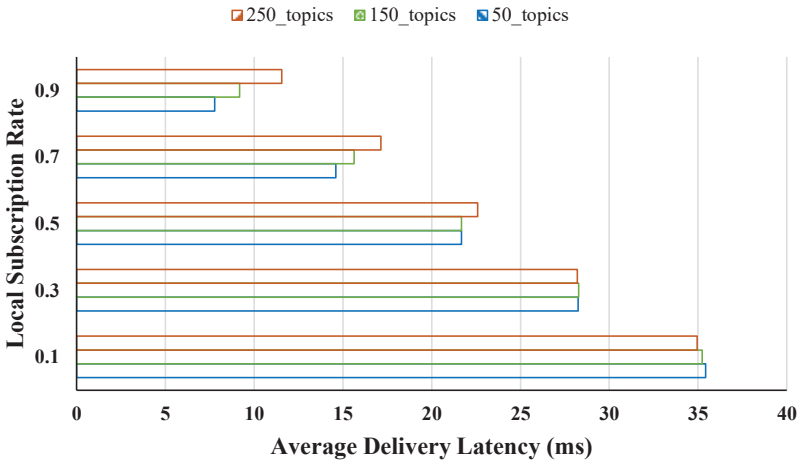

Figure 6. Average delivery latency with varying local subscription rate.

5.2. *Percentages of Forwarding Traffic among Brokers with Varying Data Localities*

The experimental setup to evaluate Average Forwarding Traffic (AFT) among brokers was similar to that of the experiment to evaluate the effect of data locality on data delivery latency, i.e., 300 brokers, 200 topics per broker, LSR ranges from 0.1 to 0.9. The simulation was repeated under different subscription distributions (uniform and Zipf) when subscribers selected topics from outside pools. AFT reflects the average number of messages exchanged among brokers during event notifications between publishers and subscribers across the system. AFT can be used as an indicator to evaluate the effectiveness of a broker overlay network. Therefore, we conducted an experiment to examine the correlation between topic subscription distributions and AFT.

In Figure 7, when LSR increases from 0.1 to 0.9, AFT rapidly decreases in both uniform and Zipf subscription distributions. Even with the lowest LSR value (0.1) when subscribers randomly select topics from remote brokers, the percentage of forwarding traffic between brokers to deliver messages from senders to receivers is approximately 8.15%. Furthermore, when client topic subscriptions follow a Zipf distribution (α = 1.2) and LSR is 0.1 and 0.9, the AFT is approximately 3.94% and 0.74%, respectively. This is because the more the consumers register to local topic channels (high LSR), the less traffic the corresponding brokers relay to each other (low AFT). In addition, note that, in the Zipf case, brokers share more common topics than the uniform distribution case; thus, our clustering algorithm provides lower relay traffic among brokers. The experiment proves that the proposed broker overlay networks are efficient in terms of the amount of relay traffic required to deliver event notifications in the pub/sub system.

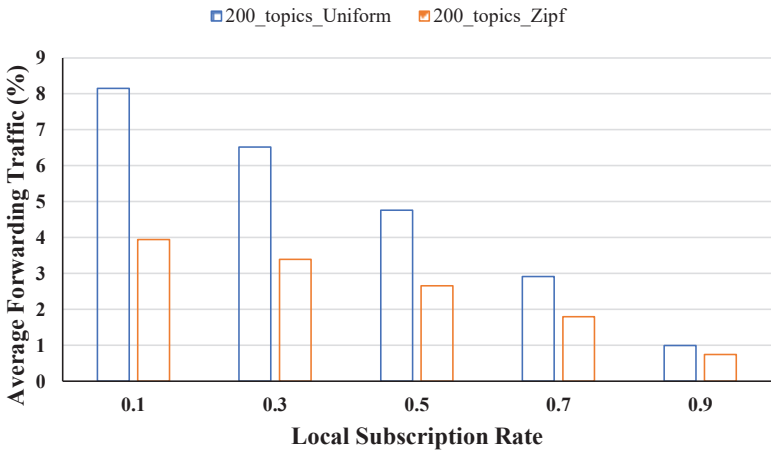

Figure 7. Average forwarding traffic with varying local subscription rate.

5.3. Scalability Evaluation

In this section, we evaluate the scalability of our pub/sub scheme and compare it to other schemes having similar approaches. Here, the experiment consisted of several simulations run with the parameters shown in Table 1. We highlight the benchmark between our proposal and two other pub/sub schemes as follows:

- A Cloud-based pub/sub scheme, a.k.a. PSCloud, in which brokers are located in the core cloud, similar to [18]. In this scheme, where pub/sub clients connect directly to pub/sub brokers in the Cloud considering topic similarity and load balance, we implemented a classic round-robin load-balancing scheme. RTTs between clients and cloud brokers are uniformly distributed with an average delay is the average delay between edge brokers calculated from our proposed scheme simulation results. Moreover, RTTs among cloud brokers are uniformly distributed between U[1, 2] ms.
- We implement another Edge-Cloud based pub/sub scheme that is inspired by the work in [19], a.k.a. PubSubCoord-alike. In this approach, edge brokers manage nearby pub/sub clients, and a group of cloud brokers provides the routing service for edge brokers to bridge topic channels in the whole system. RTTs among cloud brokers are set similar to the corresponding RTTs in the PSCloud scheme. In addition, RTTs between edge brokers and cloud servers are set similar to RTTs between clients and cloud brokers in the PSCloud case. For fair comparison, the two schemes are implemented with parameters similar to the ones in Table 1.

Table 1. Parameters for the scalability experiment.

Parameter	Value
Number of brokers	50, 100, 200, … , 600
Number of topics per broker	100
Local subscription rate (LSR)	0.8
RTT between brokers	Proportional to their distance
RTT between clients and brokers	Uniformly distributed between U[1, 4] ms
Number of clients per topic	1 publisher, 10 subscribers
Broker's port rate	10^8 bit per second
Packet size	64 bytes
Subscription distribution	Uniform, Zipf (α = 1.2)
Simulation duration	15 time steps per setting

In addition, we tested the capability of the pub/sub systems using many-to-many communication by assigning multiple publishers of different brokers to several selected topics during each simulation. Specifically, with 100 topics per broker and LSR = 0.8, 80 local topics were created for each broker, and 20 topics were randomly selected from other brokers' topic pools for the broker's clients to publish and subscribe. As a result, multi-publisher and multi-subscriber scenarios were created for our experiment. While increasing the number of brokers in the system, we calculated ADLs for messages sent from publishers to subscribers on all topic channels in the simulations. In addition, we measured the AFT of data packets exchanged among brokers to assess the effectiveness of the three schemes. The results are shown in Figures 8–11.

Figures 8 and 9 show the ADL values for the three pub/sub schemes where consumer subscription modes follow uniform and Zipf distributions, respectively. In all tested cases, our proposed pub/sub scheme provides the lowest ADL values compared to other schemes. When the clients subscribe topics according to uniform distribution, Figure 8, our proposal's ADL is approximately 57% and 20% of PubSubCoord-alike and PSCloud's ADL in all testing cases, respectively. Take the case of 300 brokers, for example—the ADL values of our proposal, PubSubCoord-alike, and PSCloud schemes are 14.98 ms, 26.36 ms, and 74.65 ms, respectively. Interestingly, ADLs of each pub/sub scheme are roughly similar while increasing the number of brokers from 50 to 600. The reason is that while the

number of clients increases together with the increase of brokers, the topic similarity rates fluctuate narrowly because subscribers follow uniform subscription mode in this experiment. That causes similar effects to ADLs in the test cases. However, it is evident from the two figures that PSCloud experiences the highest average delay followed by PubSubCoord-alike. This is due to the long delay between clients and cloud brokers in the PSCloud scheme. PubSubCoord-alike demonstrates lower delay because it takes advantage of edge brokers to manage nearby clients. The proposed scheme further reduces end-to-end delay of event notifications by clustering edge brokers based on their topic similarity and proximity geolocation. The proximate edge brokers in the same cluster can directly exchange relevant messages rather than forwarding to cloud brokers for topic bridging, as in the case of PubSubCoord-alike. Therefore, our scheme outperforms the other schemes in terms of average data delivery delay.

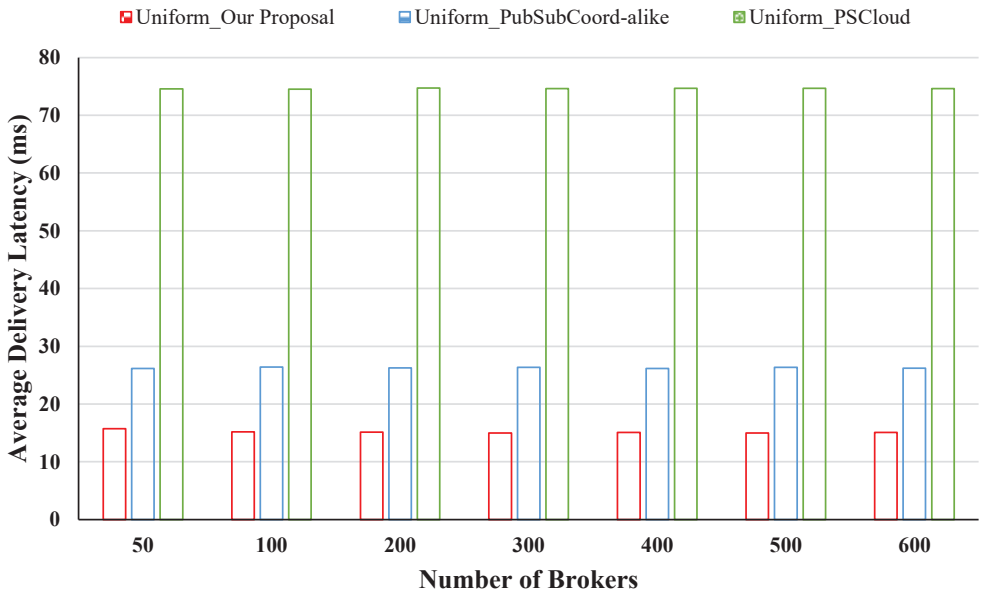

Figure 8. ADL comparison among schemes with uniform subscription distribution.

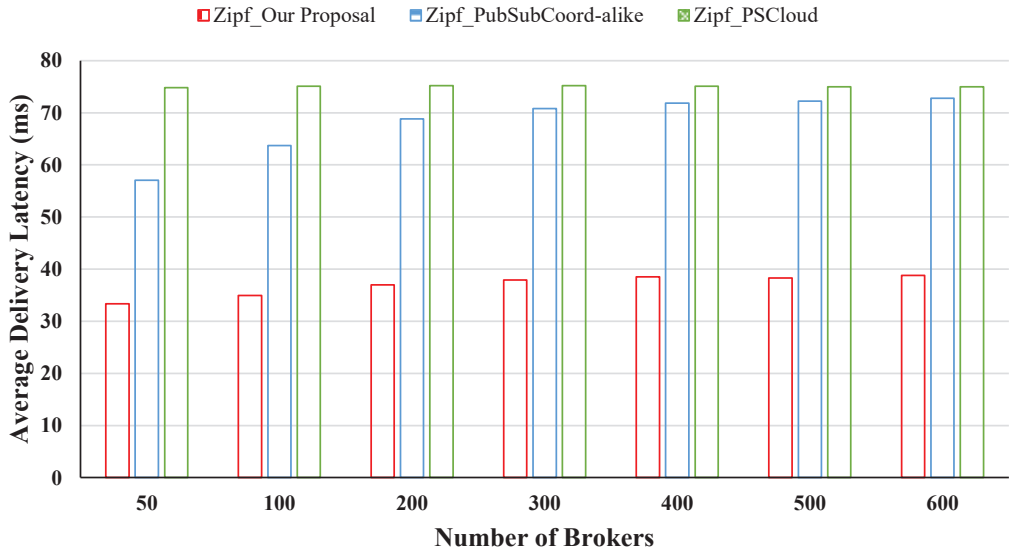

Figure 9. ADL comparison among schemes with Zipf subscription distribution.

Figures 10 and 11 plot the average forwarding ratio of pub/sub brokers of the three schemes relative to uniform and Zipf topic subscription distributions. While our proposal achieves the best

AFT in the uniform case (approximately 3%, Figure 10), PSCloud accomplishes the lowest AFT in the Zipf case (ranging from approximately 2.62% to 3.67%, Figure 11). The proposed scheme has the best AFT in the uniform distribution case because we support both topic similarity and exploit geolocation awareness while clustering brokers. Only the cluster head forwards relevant messages to other leaders. That is the reason why our scheme is better than PubSubCoord-alike in terms of average forwarding rate in both subscription modes. Interestingly, PSCloud has the best AFT in the Zipf case because there is a large number of common topics in this subscription mode; thus, clustering publishers and subscribers can be done more effectively based on topic similarity when all brokers are located in the core Cloud. However, this has no great effect in terms of saving internetworking traffic because the brokers are in the same Cloud infrastructure.

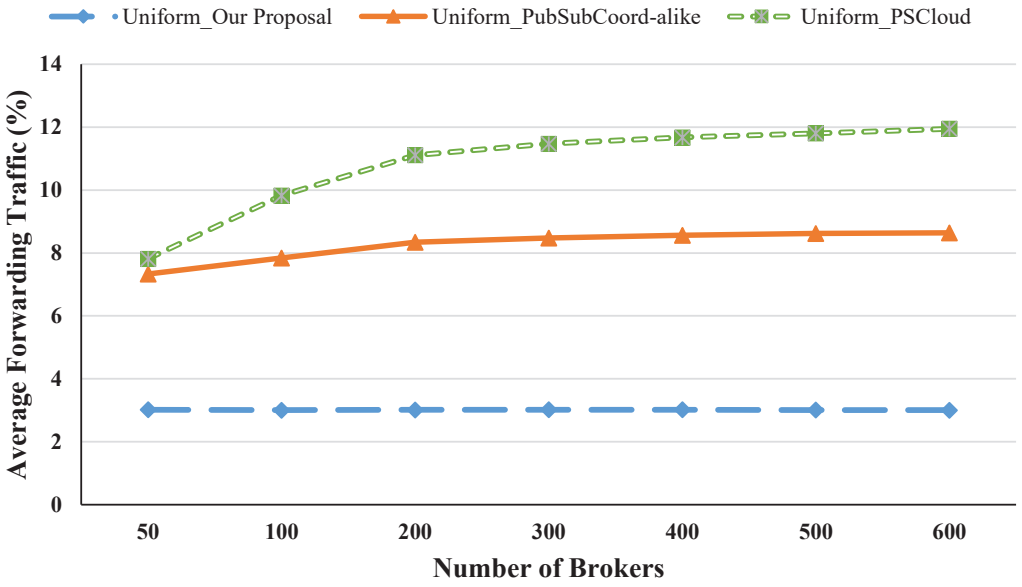

Figure 10. AFT comparison among schemes with Uniform subscription distribution.

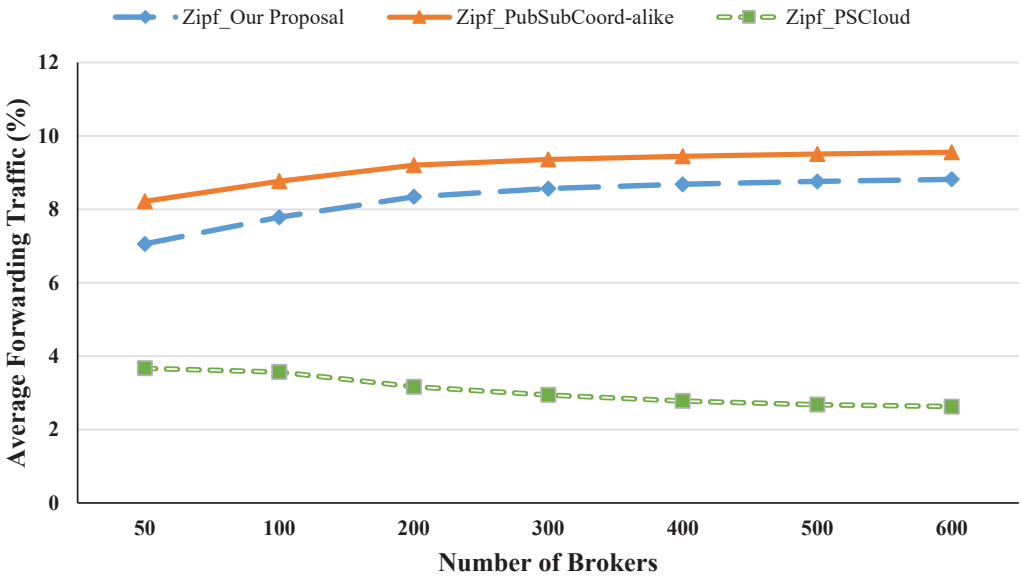

Figure 11. AFT comparison among schemes with Zipf subscription distribution.

The comparison results in Figures 8–11 prove that the proposed scheme can scale well and provide low delay message delivery for very wide-scale IoT applications with efficient forwarding traffic ratio among brokers.

6. Conclusions

We have presented a hierarchy of Edge-Cloud publish/subscribe broker overlay networks that acts as an efficient, low delay, and scalable data delivery service for wide-scale IoT applications. In the proposed model, IoT devices running pub/sub clients periodically send collected data to their responsible brokers. Then, the edge nodes process the data to make prompt controlling decisions or forward the data to other interested edge brokers or to cloud brokers for long-term processing, analysis, and storage. To facilitate these processes, we designed an effective coordination scheme to help pub/sub brokers quickly find and notify interested topics. This scheme links all brokers in the system as a single domain to deliver data for wide-scale IoT applications. Furthermore, we clustered pub/sub brokers based on similar interests and proximate geolocations to reduce delivery time and forwarding traffic among brokers.

The simulation results show that our proposal achieves the best data dissemination latency compared to PSCloud and PubSubCoord-alike, a classical cloud pub/sub scheme and a similar Edge-Cloud pub/sub scheme, respectively. In addition, the proposed scheme outperforms PubSubCoord-alike in terms of relay traffic ratio among brokers while considering the integration of the Cloud and the Fog in supporting a broad range of IoT applications. Thus, our proposal can well-support data delivery in wide-scale latency-sensitive IoT applications.

Along with advantages of our proposal, our designed brokers may consume high resources to maintain topic overlay networks in case of very large number of topics and low topic similarities among pub/sub brokers. Therefore, in the future, we will enhance our model by balancing between delivery latency, relay traffic ratio, and a number of connections between brokers when constructing the broker overlay network. In addition, we will consider applying artificial intelligence algorithms to cluster brokers and route messages in pub/sub systems more efficiently.

Author Contributions: Methodology, V.-N.P.; Software, V.-N.P.; Supervision, E.-N.H.; Writing—original draft, V.-N.P.; Writing—review & editing, V.-N.P., V.N., T.D.T.N. and E.-N.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the service mobility support distributed cloud technology (No.2017-0-00294) and under the ICT Consilience Creative program (IITP-2019-2015-0-00742) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

IoT	Internet of Things
ZB	zettabytes
TBPS	topic-based publish/subscribe
MQTT	Message Queue Telemetry Transport
API	Application Programming Interface
LAN	Local Area Network
FCB	Firebase Cloud Messaging
EB	Edge Broker
RTT	Round-Trip Time
LSR	Local Subscription Rate
ADL	Average Delivery Latency
AFT	Average Forwarding Traffic

References

1. Al-Fuqaha, A.; Guizani, M.; Mohammadi, M.; Aledhari, M.; Ayyash, M. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2347–2376. [CrossRef]
2. Cisco Global Cloud Index: Forecast and Methodology, 2016–2021 White Paper—Cisco. Available online: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/global-cloud-index-gci/white-paper-c11-738085.html (accessed on 8 November 2019).
3. Mell, P.; Grance, T. *The NIST Definition of Cloud Computing. NIST Special Publications SP 800-145*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2011.
4. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog computing and its role in the internet of things. In Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012; ACM: New York, NY, USA, 2012.
5. Bonomi, F.; Milito, R.; Natarajan, P.; Zhu, J. Fog computing: A platform for internet of things and analytics. In *Big Data and Internet of Things: A Roadmap for Smart Environments*; Springer: Cham, Switzerland, 2014; pp. 169–186.
6. Eugster, P.T.; Felber, P.A.; Guerraoui, R.; Kermarrec, A.M. The many faces of publish/subscribe. *ACM Comput. Surv. CSUR* **2003**, *35*, 114–131. [CrossRef]
7. Rahimian, F.; Girdzijauskas, S.; Payberah, A.H.; Haridi, S. Vitis: A Gossip-based Hybrid Overlay for Internet-scale Publish/Subscribe Enabling Rendezvous Routing in Unstructured Overlay Networks. In Proceedings of the 2011 IEEE International Parallel & Distributed Processing Symposium, Anchorage, AK, USA, 16–20 May 2011; pp. 746–757.
8. Setty, V.; Kreitz, G.; Vitenberg, R.; Van Steen, M.; Urdaneta, G.; Gimåker, S. The hidden pub/sub of spotify: (Industry article). In Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, Arlington, TX, USA, 29 June–3 July 2013; ACM: New York, NY, USA, 2013; pp. 231–240.
9. Antonic, A.; Roankovic, K.; Marjanovic, M.; Pripuic, K.; Arko, I.P. A Mobile Crowdsensing Ecosystem Enabled by a Cloud-Based Publish/Subscribe Middleware. In Proceedings of the 2014 International Conference on Future Internet of Things and Cloud, Barcelona, Spain, 27–29 August 2014; pp. 107–114.
10. Rowe, A.; Berges, M.E.; Bhatia, G.; Goldman, E.; Rajkumar, R.; Garrett, J.H.; Moura, J.M.; Soibelman, L. Sensor Andrew: Large-scale campus-wide sensing and actuation. *IBM J. Res. Dev.* **2011**, *55*, 6. [CrossRef]
11. Menzel, T.; Karowski, N.; Happ, D.; Handziski, V.; Wolisz, A. Social sensor cloud: An architecture meeting cloud-centric iot platform requirements. In Proceedings of the 9th KuVS NGSDP Expert Talk on Next Generation Service Delivery Platforms, Berlin, Germany, 9 April 2014.
12. Message Queue Telemetry Transport. Available online: http://mqtt.org/ (accessed on 8 November 2019).
13. Chen, C.; Tock, Y.; Jacobsen, H.; Vitenberg, R. Weighted Overlay Design for Topic-Based Publish/Subscribe Systems on Geo-Distributed Data Centers. In Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, USA, 29 June–2 July 2015; pp. 474–485.
14. Firebase Cloud Messaging. Available online: https://firebase.google.com/docs/cloud-messaging/ (accessed on 10 Deccember 2019).
15. IoTivity Software Framework. Available online: https://www.iotivity.org (accessed on 10 December 2019).
16. Chockler, G.; Melamed, R.; Tock ,Y.; Vitenberg, R. Spidercast: A scalable interest-aware overlay for topic-based pub/sub communication. In Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, Toronto, ON, Canada, 20–22 June 2007; ACM: New York, NY, USA, 2007; pp. 14–25.
17. Girdzijauskas, S.; Chockler, G.; Vigfusson, Y.; Tock, Y.; Melamed, R. Magnet: Practical subscription clustering for internet-scale publish/subscribe. In Proceedings of the 4th ACM International Conference on Distributed Event-Based Systems (DEBS), Cambridge, UK, 12–15 July 2010.
18. Gascon-Samson, J.; Garcia, F.; Kemme, B.; Kienzle, J. Dynamoth: A Scalable Pub/Sub Middleware for Latency-Constrained Applications in the Cloud. In Proceedings of the 2015 IEEE 35th International Conference on Distributed Computing Systems, Columbus, OH, USA, 29 June–2 July 2015; pp. 486–496.
19. An, K.; Khare, S.; Gokhale, A.; Hakiri, A. An autonomous and dynamic coordination and discovery service for wide-area peer-to-peer publish/subscribe: Experience paper. In Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems, Barcelona, Spain, 19–23 June 2017; pp. 239–248.

20. Atlam, H.F.; Walters, R.J.; Wills, G.B. Fog computing and the internet of things: A review. *Big Data Cognit. Comput.* **2018**, *2*, 10. [CrossRef]
21. Jafari, S.J.; Naji, H. GeoIP clustering: Solving replica server placement problem in content delivery networks by clustering users according to their physical locations. In Proceedings of the 5th Conference on Information and Knowledge Technology, Shiraz, Iran, 28–30 May 2013; pp. 502–507.
22. Katz-Bassett, E.; John, J.P.; Krishnamurthy, A.; Wetherall, D.; Anderson, T.; Chawathe, Y. Towards IP geolocation using delay and topology measurements. In Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement, Rio de Janeriro, Brazil, 25–27 October 2006; ACM: New York, NY, USA, 2016; pp. 71–84.
23. Eriksson, B.; Barford, P.; Maggs, B.; Nowak, R. *Posit: An Adaptive Framework for Lightweight IP Geolocation*; Computer Science Department, Boston University: Boston, MA, USA, 2011.
24. Hunt, P.; Konar, M.; Junqueira, F.P.; Reed, B. ZooKeeper: Wait-free coordination for internet-scale systems. In Proceedings of the USENIX Annual Technical Conference, Boston, MA, USA, 23–25 June 2010; Volume 8.
25. ZooKeeper Overview. Available online: https://zookeeper.apache.org/doc/r3.5.5/zookeeperOver.html (accessed on 8 November 2019).
26. Wikipedia. Haversine Formula. Available online: https://en.wikipedia.org/wiki/Haversine_formula (accessed on 8 November 2019).
27. Rahimian, F.; Huu, T.L.N.; Girdzijauskas, S. Locality-awareness in a peer-to-peer publish/subscribe network. In *IFIP International Conference on Distributed Applications and Interoperable Systems*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 45–58.
28. Zhao, Y.; Kim, K.; Venkatasubramanian, N. Dynatops: A dynamic topic-based publish/subscribe architecture. In Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems, Arlington, TX, USA, 29 June–3 July 2013; ACM: New York, NY, USA, 2013; pp. 75–86.
29. SimPy. Available online: https://simpy.readthedocs.io/en/latest/ (accessed on 8 November 2019).
30. Periscope. Starbucks-Locations Data. Available online: https://community.periscopedata.com/t/80fyna/starbucks-locations (accessed on 8 June 2019).