sustainability

MDPI

Article

Fog BEMS: An Agent-Based Hierarchical Fog Layer Architecture for Improving Scalability in a Building Energy Management System

Uikyun Na [1,2] and Eun-Kyu Lee [1,*]

[1] College of Information Technology, Incheon National University, Incheon 22012, Korea; uikyun@inu.ac.kr
[2] Digital Solution Laboratory, KEPCO Research Institute, Daejeon 34056, Korea
* Correspondence: eklee@inu.ac.kr; Tel.: +82-32-835-8629

Received: 26 December 2019; Accepted: 27 March 2020; Published: 2 April 2020

Abstract: It has been found that a cloud building energy management system (BEMS) alone cannot support increasing numbers of end devices (e.g., energy equipment and IoT devices) and emerging energy services efficiently. To resolve these limitations, this paper proposes Fog BEMS, which applies an emerging fog computing concept to a BEMS. Fog computing places small computing resources (fog nodes) just next to end devices, and these nodes process data in real time and manage local contexts. In this way, the BEMS becomes distributed and scalable. However, existing fog computing models have barely considered scenarios where many end devices and fog nodes are deployed and interconnected. That is, they do not scale up and cannot be applied to scalable applications like BEMS. To solve the problem, this paper (i) designs a fog network where a list of functionally heterogeneous nodes is deployed in a hierarchy for collaboration and (ii) designs an agent-based, modular programming model that eases the development and management of computing services at a fog node. We develop a prototype of a fog node and build a real-world testbed on a campus to demonstrate the feasibility of the proposed system. We also conduct experiments, and results show that Fog BEMS is scalable enough for a node to connect up to 900 devices and that network traffic is reduced by 27.22–97.63%, with varying numbers of end devices.

Keywords: building energy management system; fog computing; scalability; multi-agent system; Internet of Things; hierarchical architecture; distributed system

1. Introduction

Existing building energy management systems (BEMS) have been developed on top of a cloud computing architecture, and such cloud BEMSs have enjoyed various benefits as follows. Cloud computing, sitting at a centralized data center, provides elastic computing resources; it supports both unlimited data storage and powerful data processing at a low cost. The cloud can handle a large volume of data and run advanced algorithms to extract insight from massive data. Recently, cloud computing has been integrated with the Internet of Things (IoT). More end devices (e.g., IoT devices, energy equipment, and mobile users) are being deployed and connected to the cloud, and advanced applications are being introduced every day. Suppose a BEMS for a large building complex consisting of a few buildings. In the future, many energy devices, sensors, and actuators will be installed in that complex; Gartner forecasted that 1.81 billion IoT devices will be in use in utilities and building automation sections in 2020 [1]. The BEMS of tomorrow will communicate with them and provide a variety of energy services. For instance, each building can have a number of tenants, each of which runs their own micro BEMS that provide tenant-customized services. However, recent research reports that cloud computing alone cannot meet all functional requirements of emerging IoT applications. Limitation primarily comes from the fact that the cloud is geographically far from

end devices. Distance introduces communication overhead, and long delay is inevitable, which is not suitable for emerging latency-sensitive IoT applications. Because of the distance, moreover, the cloud is often unable to access local context data such as precise location information, local situations, and user movement that recent interactive applications consume frequently.

1.1. Solution Approach

In order to resolve the limitation, this paper proposes a Fog BEMS that applies an emerging fog computing concept to a BEMS. Instead of using centralized cloud servers, fog computing places small computing resources (fog nodes) at the edge of a network where end devices are a distance of one hop away. By sitting next to end devices, nodes are able to process user data in real time and to use local contextual information. Computing resources are distributed, so a BEMS becomes scalable; it can cope with numerous end devices quickly and actively. Suppose that a mobile device runs an energy application using augmented reality that needs to process a large volume of video data in real-time. Due to limited computing resources, the device offloads a computationally intensive task to a nearby fog node. Because the node is very close to the device, it can quickly handle the video data on behalf of the device. Recent research has examined the feasibility of such offloading and its impact on the performance of applications that end devices are running.

It is no doubt that fog computing technology benefits BEMSs in distributed environments. However, research is still at a very early stage; the computing architecture has yet not been fully designed. Its initial design only considers simple applications, such as the offloading scenario where a fog node provides a service to an end device at once. In Fog BEMS, we will see many fog nodes deployed, and a node will provide multiple computing services to a variety of end devices with different types of application requirements. A group of nodes may communicate with each other to provide computing services. Existing fog computing models, however, have hardly determined *how many fog nodes can collaborate together* and *how a fog node can operate multiple services efficiently*. That is, they do not scale up and cannot be applied to scalable applications such as BEMSs immediately.

1.2. Contribution

This paper aims to resolve these problems so that a fog computing model is enhanced enough to support scalable applications. Eventually, we build Fog BEMS that can process energy data in real time and interact with end devices more actively in a large building complex. The contributions of this paper are summarized as follows.

- We propose Fog BEMS, a fog-based computing architecture for a BEMS that addresses two technical issues. First, we designed a fog network where a list of independent and functionally heterogeneous nodes are deployed in a hierarchy for collaboration, and a data merging algorithm is proposed to minimize processing/communication overhead in the network. Second, we designed an agent-based, modular programming model that can ease the development, deployment, and management of computing services at a fog node. The model also abstracts communication details so that fog nodes can collaborate seamlessly (Section 3).
- We developed a prototype of Fog BEMS. Various types of fog nodes were developed using open source software. Using the prototype, we built a testbed on a campus that contains many sensors and energy devices, such as smart plugs, smart lights, etc. We also investigated message protocols used for communications between fog nodes. A simple scenario of an energy management operation was demonstrated on top of the testbed to show the feasibility of the Fog BEMS (Section 4).
- We conducted experiments to evaluate the performance of the Fog BEMS from two perspectives. First, scalability performance was assessed by measuring the maximum number of devices with which a fog node can communicate in a given time deadline. Next, experiments measured both the total numbers of messages and their sizes in bytes transmitted over the fog network. The results show that the proposed architecture reduces overall communication overhead (Section 5).

The following section begins with a brief review about computing architectures and the smart energy applications used with them.

2. Preliminary

2.1. Technology Review: Computing Architecture for an Energy Management System

2.1.1. Cloud Computing

In cloud computing, a large cloud service provider such as Amazon provides an on-demand availability of computing resources, especially data storage and computation power to end users. Instead of building a server room, a company purchases the computing resources with which it provides its own services faster and is able to adjust resources quickly. This capability reduces up-front IT infrastructure cost and leads to improved manageability and less maintenance [2,3]. In order to meet complex customer demands, several independent service providers often collaborate by composing cloud services across themselves. In such a multi-cloud environment, it is a challenging task to locate best-fit services that satisfy both user needs and cloud providers' interests. Various multi-agent systems (MASs) have been proposed to resolve the service composition problem. Gutierrez-Garcia et al. [4] developed an MAS where a horizontal composition integrated heterogeneous services scattered across several clouds, and a vertical composition put homogeneous resources together to expand the capacity of the cloud. Kendrick et al. [5] proposed an energy efficient service composition method consisting of agent-matchmakers that worked on behalf of the user to process requests and agent representatives that mediated communications between the matchmaker and cloud data center. Kim [6] proposed a multiagent-based negotiation mechanism that established a service level agreement among cloud stakeholders for service composition.

The integration of cloud computing and IoT enables centralized data management and powerful data processing at a low cost [7]. A cloud computing architecture for IoT applications consists of two layers: a cloud layer and an IoT layer. The cloud layer provides a centralized interface for data storage, processing, and access for large-scale data. The IoT layer represents billions of devices deployed pervasively around human beings—user devices, such as mobile phones, and complex equipment, such as vehicles. The two layers communicate with each other via intermediate equipment, such as gateways and routers, and exchange data using standard communication protocols [8]. The cloud-based IoT architecture has also been applied to smart grid applications and BEMSs where a variety of energy devices comprise the IoT layer [9,10], building a cloud BEMS architecture. Researchers have proposed cloud solutions for demand response and microgrid management [11–14]. Several works have leveraged cloud computing for collecting, processing, and sharing massive amounts of energy data [15–17].

2.1.2. The Paradigm Shift from Cloud To Fog

Despite the benefits of the cloud-based IoT architecture, cloud computing does not meet all of the functional requirements of emerging IoT applications. A fundamental limitation is the physical distance between any two layers. All of the computing resources are centralized in the cloud, while IoT devices are deployed over wide, geographically distributed areas. A large separation between them causes an increase in average network latency and jitter, which is not suitable for latency-sensitive applications such as connected vehicles, augmented reality, and smart grids [18,19]. Moreover, the distance prevents the cloud from directly accessing local contextual information, such as precise location information, local network conditions, and mobility patterns, and this disables support-related applications.

A new concept of fog computing (or edge computing) has been introduced to tackle these limitations [20]. The fog defines an architecture that provides computing, storage, and networking services, which have been supported fully by the cloud, at the edge of the network. This is achieved through shared computing resources in routers, proxies, and small servers positioned near the end

devices. Fog computing has become a novel paradigm of computing, with key properties such as decentralization, ubiquity, and support for collaboration among fog nodes, not as an extension of cloud computing. Table 1 summarizes the distinguishing features of fog computing [21,22], and Table 2 provides a comparison between cloud computing and fog computing. Note that fog computing is not intended to replace the cloud; the fog performs local data analysis and processes certain application tasks, while the cloud executes permanent storage and global analysis and runs computational intensive tasks [18].

With the fog, an entire computing architecture is composed of three layers as shown in Figure 1: a *cloud layer*, a *fog layer*, and an *IoT device layer*. The fog layer consists of network equipment (e.g., routers, access points, and gateways) and local servers (e.g., an embedded server, video surveillance cameras, and industrial devices). These fog nodes are deployed anywhere with network connections and run applications. Due to the proximity of fog nodes to the end devices, they easily acquire local knowledge, quickly respond to latency-sensitive applications, and significantly reduce network traffic by removing the need for massive data transmission toward a cloud server. Fog nodes, as intermediate components, can forward data summaries to the cloud and provide services of transient storage and real-time data analysis. In addition, they can collaborate to offer services together without connection to the cloud: smart traffic lights and local content distribution.

Figure 1. An entire computing architecture consists of three layers: cloud, fog, and IoT devices.

Table 1. Features of fog computing.

Features	Definition
Location Awareness and Low Latency	The fog is able to specify the location of fog nodes physically and logically and to ensure low latency by placing fog nodes close to end devices.
Geographical Distribution	Services and applications provided from the fog are extensively distributed.
Extensibility	The fog provides storage that can work with distributed computing and large end devices.
Mobility	The fog provides mobility via direct connection to mobile devices.
Collaboration in Real-Time	It supports interaction and collaboration among fog nodes.
Heterogeneity	Fog nodes can run on different, heterogeneous platforms.
Interoperability	A fog component can seamlessly interoperate with counterparts in other domains or with different service providers.
Interaction with Cloud	The fog is placed between the cloud and end devices and performs data processing in the middle.

Table 2. Comparison of cloud and fog computing.

Feature	Fog Computing	Cloud Computing
Latency	Low	High
Distance between Client and Server	One Hop	Multiple Hops
Architecture	Dencentralization	Centralization
The Number of Computing Nodes	Many	Few
Location Awareness	Yes	N/A
Mobility	Yes	N/A
Perspective of Hardware	Limited Computing Power and Storage	Extensive Computing Power and Storage

2.2. Related Work

2.2.1. Existing Research in Fog Computing

Research on fog computing has been conducted in three fields: architecture, algorithm, and applications. Researchers have studied architectures to enable federation between the cloud and the fog [23] and to perform optimal resource management and load balancing [24]. Such distributed architectures inevitably require efficient communications, which have been studied in the literature [25]. The authors in [26] proposed a high-level programming model for IoT applications provided over the cloud and the fog. Studies in the algorithm field have focused on enabling distributed fog nodes to run portions of computing tasks. Tasks are performed on top of resources shared by many fog nodes. In this sense, resource sharing algorithms and task scheduling algorithms have been active research topics [27,28]. The authors in [29,30] proposed strategies of computation offloading and load distribution. Fog computing has been applied to a variety of applications, including healthcare and medical applications, connected vehicles, smart cities, surveillance systems, and video streaming [31–35]. Previous research has shown that (i) fog computing reduces latency compared to traditional cloud computing [33] and (ii) reduces the volume of network traffic towards the cloud [25]. Local processing at fog nodes saves bandwidth and decreases processing delays by avoiding the turnaround overhead with the cloud.

2.2.2. Fog Computing in Smart Energy Applications

Many tutorial papers address that fog computing benefits smart grid applications well [36,37]. Jalali et al. [38] examined that interconnecting fog computing and a microgrid cloud localize IoT loads and distributed energy. This reduces the distance between energy use and production and thus decreases the energy consumption of IoT applications. Naranjo et al. [39] addressed the benefits of using big data on the fog when designing and supporting smart grid applications. Li et al. [40] proposed a secure demand response (DR) scheme to protect energy systems against collusion attacks. They utilize a fog node as a sanitizer that randomly transfers encrypted energy states and DR strategies in a homomorphic manner. Khalid et al. [41] investigated a fog-based approach to share energy among consumers, where a fog server enables the consumers to communicate each other. Javed et al. [42] addressed the case of communication failure between a cloud BEMS and endpoints (e.g., electric vehicles). They proposed installing a forecasting model at the edge of the network so that endpoints could operate as expected, even in the case of failure. Some research has investigated network architectures. The authors in [43] proposed a hierarchical structure of cloud-fog computing for efficient resource management in a smart grid. They evaluated five heuristic algorithms for load balancing. Wang et al. [44] proposed a fog-based architecture for the purpose of latency-sensitive monitoring and for location-aware and intelligent control for IoT applications in a smart grid. Yue et al. [45] proposed a hierarchical energy management structure with artificial intelligence embedded in a fog layer. Based on the architecture, they established a utility and revenue model for various stakeholders, which optimized their decision-making. The authors in [46] noted that an existing, centralized information processing architecture could not scale up with the increasing numbers of smart meters. They proposed a new fog computing architecture where smart meters act as

fog nodes with advanced capabilities of data storage and processing, improving data management efficiency. Moghaddam and Leon-Garcia [47] proposed a multitier communication architecture for transactive energy management systems. With a fog node acting as a retail energy market server, the architecture implements a real-time interface between distributed energy resources and customer devices via home gateways, which enables customers to trade energy with neighbors. Al-Faruque and Vatanparvar [48] implemented a service-oriented architecture (SOA) for a home energy management system over the fog in order to reduce traffic toward the cloud and network delay. Control panels act as fog nodes and are responsible for gathering, storing, processing, and analyzing energy data at home without any interaction with the cloud.

2.3. Problem Statement

Despite the research advancement in fog computing, we still face challenges that occur when implementing the fog computing concept in a real-world environment. The state-of-the-art research in fog computing has focused on the feasibility of two types of individual functions at a fog node. First, a node acts as an interacting node that obtains and processes data from end devices and responds to them for real-time services and location-aware services. Second, a node acts as an intermediate node between end devices and the cloud; it forwards data collected from end devices to the cloud and/or distributes cloud messages to the devices. Existing research, however, has barely investigated a "fog layer", where *many fog nodes* are deployed and inter-connected, each of which runs *multiple functions*. That is, existing fog computing models do not scale up and cannot be applied to scalable applications, such as a BEMS. Suppose a building environment where a BEMS is deployed and operated: (i) a commercial building that consists of private rooms, public spaces, hallways, parking lots, a rooftop, etc. and (ii) a campus with buildings, roads, and a sports field. It is expected that end devices are everywhere in the environment (say, on a campus); countless numbers of sensors and energy devices are deployed both inside and outside buildings, and mobile devices walk around. Accordingly, many fog nodes will be deployed and construct a fog layer that provides fog computing functionalities for various smart energy services. Such a pervasive deployment of end devices and fog nodes introduces new research problems to be resolved.

First, a network for the fog layer must be designed. Fog nodes sit just next to end devices to provide real-time services. As more end devices are continuously installed and users actively use services, fog nodes will work more, eventually suffer from processing overhead, and create unfavorable latency; we call it a *device scalability* issue in the fog layer. Our initial experiment (shown in Section 5) discovers that it takes 2.86 s for a fog node to collect data from more than 2000 energy data points. If the node is required to perform more functions (say, control devices), additional delay is unavoidable and real-time services are impossible. One might say that deploying more fog nodes on the campus and distributing functions among them can appropriately solve the issue. However, this approach can cause another problem. Note that fog nodes communicate directly with the cloud to exchange a number of data. From a network perspective, network traffic concentrates on a gateway of the fog layer. This star topology, thus, can cause network traffic problems inside the layer as the number of end devices and fog nodes in the fog layer increases. In addition, an interaction model among fog nodes has not yet been examined, in spite of our notion that nodes' collaboration may reduce communication/processing overhead as well as enable advanced fog computing services. Therefore, it is necessary to develop a new network structure of the fog layer.

Second, a programming model that allows fog nodes to run multiple, heterogeneous functions efficiently must be designed. For instance, a fog node is installed at the 2nd floor in a five-story building to provide fog computing services to an IT team. The node may record energy usage of individual staff and provide personal energy management services. It can communicate with presence sensors so that it triggers an energy saving operation upon detecting that there is no one at the office. After obtaining weather information from the web and brightness data from sensors attached on windows, the node can also control window blinds in a fine-grained manner to provide the best comfort to staff. When a fog

node is placed at the lobby of the building, it may communicate with all other fog nodes in the building and compute an energy summary for the entire building in addition to performing conventional energy services. It is obvious that fog nodes installed in various places provide different functions for different services, and a node will offer multiple functions simultaneously. We call this situation a *functional heterogeneity* issue in the fog layer. Programming models in existing research, however, are initially designed to implement an individual function and to evaluate the performance of each service. They do not take into account the requirement of functional heterogeneity inside the fog layer and thus cannot develop a fog node running heterogeneous, multiple functions efficiently. If new sensors and actuators are deployed and a new user requirement emerges, a fog node needs to implement additional functions on the fly. It is necessary to develop a new programming model that eases the deployment and management of node functions in a flexible way.

3. Fog BEMS: A Proposed Computing Architecture

This section proposes Fog BEMS, a fog computing architecture for a BEMS. For scalable services, it designs a fog layer where we propose (i) a new network structure to resolve the device scalability issue and (ii) a new programming model to resolve the functional heterogeneity issue.

3.1. Location-Aware, Hierarchical Structure

The goal of a fog network (i.e., a network for the fog layer) design is to deploy and coordinate fog nodes properly so that the fog layer can provide computing services efficiently to end users. The network should be able to minimize processing/communication overhead as well as enable nodes' collaboration.

3.1.1. Hierarchical Deployment of Fog Nodes in Zone, Building, and Campus

We design a fog network that consists of three levels in a hierarchy, where the bottom level directly interacts with end devices, as shown in Figure 2. One of the outstanding features of fog computing is location awareness. This implies that a fog node is not only able to learn knowledge about a location but also provide services using the knowledge in the vicinity. We call such a physical boundary of location a *zone*. Examples of a zone include a classroom, a hallway, a lobby, a specific section in a parking lot, the front of a library, etc. The boundary can also be interpreted in logical terms; a zone may refer to an area providing common services to an IT team. The zone becomes a basic place where a fog node is deployed. A zone may be full of end devices, mobile users, and services. Thus, it is also possible that two fog nodes are deployed in a crowded zone and are configured to collaborate to provide services together. By using the zone concept, we can identify functionalities and services required in each zone and thus distribute functions among nodes appropriately, which eventually results in balancing the processing overhead. For easy understanding, this paper assumes a zone where a fog node provides computing services to the IT team whose staff members sit close to each other at a public office.

We define a next level boundary, *building*, that consists of all the zones in a building. It is also a location where a fog node is deployed. The fog node installed in the building, named a building fog, acts as a zone head. It collects data from zone fogs (fog nodes installed in zones) and performs data processing and analysis at a building level. The building fog can store data collection in a local database if necessary and/or forward a building summary to a fog node at a higher level. It also distributes data to zone fogs. For instance, it can send a demand response message or energy price data to zone fogs so that they execute their own operation policy to save energy consumption. The building fog can be implemented by a separate hardware with a more powerful computing capability than conventional zone fogs, or a zone fog can be selected to play the building fog role. A fog node can also be installed at the highest level boundary, *campus*, named a campus fog. It acts as a building head by communicating with building fogs. The campus fog is implemented by a powerful server, a reliable database, and an energy management system so that it can run operations to maximize energy

efficiency on a campus level. It is also responsible for communicating with the cloud if necessary and with a utility company.

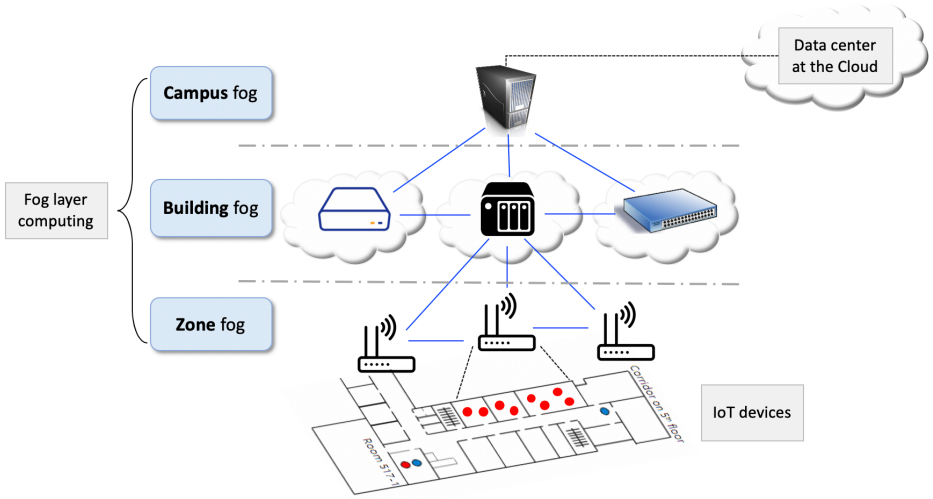


Figure 2. Fog building energy management system (BEMS) defines a fog layer of a location-aware, hierarchical network structure consisting of three layers: zone, building, and campus. Fog nodes deployed at these levels perform different tasks.

3.1.2. Data Merging Algorithm

This subsection proposes a data merging algorithm to reduce network traffic in the fog layer with a number of fog nodes. We note that the largest portion of traffic transmitted inside the layer is energy data collected from end devices and is forwarded to the database in the campus fog. Moreover, data is collected periodically, and a data packet is small in size. As more devices that report energy data frequently are installed, the fog network will saturate rapidly. To avoid network overhead, a fog node in the proposed algorithm groups data collected from the same type of devices and merges it to one message before forwarding it to the upper level. This eventually reduces the number of packets transmitted in the layer. The merging algorithm works well with many end devices of similar types, which is the case of the BEMS. For instance, a smart light generally reports three points of data: on/off status, dimming, and color. Once 100 lights are installed in a zone, 100 data packets are generated and delivered periodically. Since the data uses the same message format, the zone fog can merge data into one message easily. In a similar way, data from 100 smart plugs can be grouped. The algorithm operates as follows. When a fog node receives data from an end device, it obtains a tuple of device name, point name, point value, and type of device. It then creates a key using the device type and the point name and searches for the same key in the list of existing dictionary keys. If found in the list, the node updates (saves) the tuple to the dictionary variable. Otherwise, it adds the key to the list.

3.2. An Agent-Based, Interoperable Programming Model

The goal of a programming model design is to ease the development, deployment, and management of node functions so that a fog node can run heterogeneous, multiple functions in a flexible manner. The model should allow fog nodes to communicate with each other seamlessly to encourage collaboration.

To achieve the goal, a node function must be designed and developed in a modular way. A function in a fog node may operate alone or be integrated with other functions to create a complex service together. In the meantime, a new function can be added to the node and an existing function may be updated or replaced with a new one. For instance, a fog node communicates with a smart light to collect data, stores data in a local database, and displays the status information of the light on a screen; that is, it runs three functions. When the node needs to communicate with other fog

nodes, an appropriate function must be developed and added. If it allows a user to control the light via the touch screen, the display function must be updated. In this scenario, the development and deployment of such functions must be completed with minimized efforts, and any change in a function does not affect the operations of other functions. We believe that these requirements are achieved via a modular design.

In order to satisfy the requirements, this subsection proposes a programming model by adopting the concept of an MAS. Traditionally, an agent in an MAS is defined as an autonomous entity with intelligence [49]. An entity senses various parameters from the external environment that are then used to make a decision. It can also take an action based on the decision if necessary. The benefit of agents is harnessed most when multiple agents interact and collaborate with each other to solve complex problems that are difficult for an individual agent to solve.

This paper applies the concept of an MAS to a programming model for a fog node in contrast to existing approaches that adopt MASs as distributed system models. That is, the proposed model enables the development of a fog node that runs multiple agents, each of which performs its own function. To this end, we release the requirement that an agent perform multiple functions and be autonomous. An agent in a fog node is responsible only for performing a single function, for instance, controlling a smart light. It is not necessarily intelligent. Instead, the node running a group of agents acts as an autonomous entity; agents in the node are properly organized and related closely so that the node, as an agent head, behaves intelligently. To this end, the proposed model categorizes agent types as described in Table 3. We take the object-oriented feature of an agent for implementation so that agents are not tightly coupled, which achieves the modular design and enables easy development.

Table 3. A fog node can have different types of agents in it.

Agent Type	Functioning Role
Sensing Agent	Collects environment data from sensors.
Action Agent	Performs actions on the environment (e.g., turns a light off).
User Agent	Interacts with a user (especially a human being).
Memory Agent	Manages a local database.
Intelligent Agent	Analyzes data and makes an important decision.
Communication Agent	Connects to other fog nodes and agents for collaboration.
Internet Agent	Communicates with external services via the Internet (e.g., obtaining weather information).
Application Agent	Can develop application-specific functions (e.g., handling demand response signals).

The proposed model also adopts the advantages of MASs: distribution (local view) and decentralization. Agents (and fog nodes) are deployed in geographically distributed areas. An agent can obtain knowledge about the local environment and act as a computing resource to control the environment. Other agents in different nodes can consult the agent to access that knowledge and use it for their own purposes. Agent communications operate in a decentralized manner; an agent can connect to any other agent in a network without a centralized authority. To make them possible, agents must be loosely coupled enough and be able to make seamless communications on demand. Out of many candidates for communications between agents, our model uses a messaging protocol. Also known as a message queueing service or a message-oriented middleware, it supports both point-to-point transmission and broadcast communication. In particular, the publish-and-subscribe pattern in the protocol allows publishers (data senders) and subscribers (receivers) to access queues and/or topics without prior knowledge of communication counterparts when exchanging data. The messaging protocol, moreover, provides a unified way of communication among agents by hiding details of network protocols. It is an open application programming interface (API) model, implying that it does not rely on the underlying system. Agents in a fog node communicate via a common message queue. For agent communications in different nodes, a communication agent in the proposed model implements the messaging protocol, helping other agents interact over a network.

By applying an MAS, the proposed programming model has the following advantages.

- Agents do not need to receive information about unnecessary agents by making them aware of other agents.
- The model provides the flexibility of adding and developing new agents (functions) without affecting the existing structure of a fog node through plug and play.
- Any changes in a fog node do not affect the behavior of any node in a fog layer.
- A fog layer has optimized features for distributed systems by performing autonomously and collaborating at the same time.

4. Implementing the Proof of Concept

This section provides the proof of concept (PoC) of the proposed Fog BEMS; we developed a prototype and built a testbed on a campus. The prototype demonstrates that agents and fog nodes can be easily developed and deployed in a fog layer, which resolves the functional heterogeneity. It also shows the feasibility of a hierarchical computing architecture on a campus. To this end, we exploited an existing open source software platform, developed a list of agents, deployed fog nodes, and ran energy management operations.

4.1. Software Platform for Fog Nodes

There are many platforms and frameworks for developing MASs [50]. For example, the Java Agent Development Environment (JADE) is a well-known open source software framework for agent development [51]. It provides a standard method for library delivery and agent-to-agent communication to create agents. JADE is fully implemented in Java. Out of many candidates, this paper makes use of Volttron, a distributed agent execution framework, designed by the Pacific Northwest National Laboratory (PNNL), that is specialized for applications of energy systems [52,53]. A Volttron platform is implemented in Python utilizing many open source libraries, and a set of utilities helps speed up the development and deployment of agents in a scalable and efficient manner. Each agent performs its own functions and/or completes complex operations via cooperation as needed. Agent communications are established through a central "information exchange bus", as shown in Figure 3, and data is transferred in the form of topics and subtopics. A topic is a feature of a hierarchical structure that is dynamically generated and can be in any format, providing flexibility [54]. An example is "topic/subtopic/subtopic/"="weather/location/temperature/", and data is encoded in Python's dictionary format.

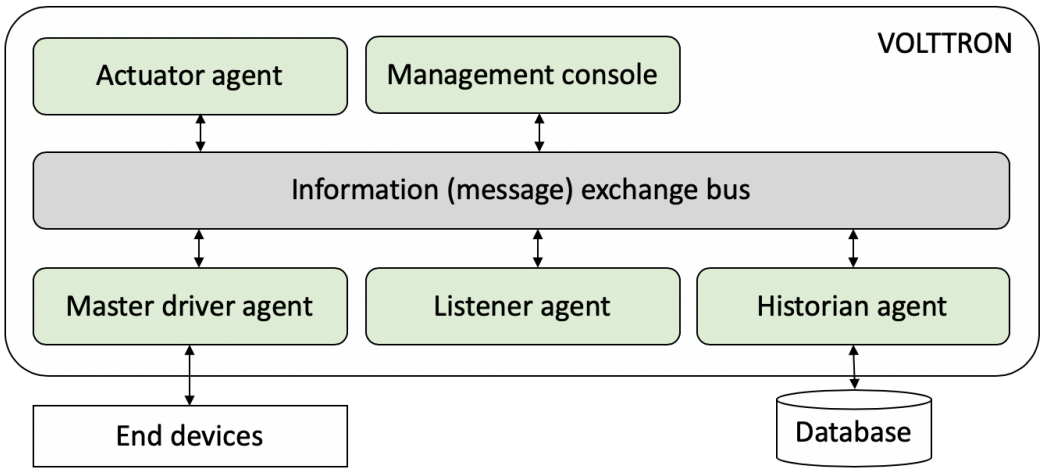


Figure 3. Volttron is a development framework that contains representative base classes.

Volttron provides a development framework that helps develop different types of agents. For instance, a Master Driver Agent represents a base class, allowing us to develop agents that manage and control all connected end devices, and coordinates them. Likewise, a Historian Agent

is responsible for receiving data and storing it in a database. Its open source includes platform services providing underlying functionalities used by agents to perform their tasks. They are also in the form of an agent; for instance, a base class for a Listener Agent can receive data published to the bus from other agents. Volttron has four salient features: scalability, interoperability, security, and cost-effectiveness [55]. It is scalable enough to install and manage multiple devices in one building or a fleet of buildings, it provides the ability to interact and connect with various systems both in and outside the energy sector, it is underpinned with a robust security foundation to combat today's cyber vulnerabilities and attacks, and it is lightweight enough to be hosted on inexpensive computing resources such as Raspberry Pi.

4.2. Message Protocols

Our prototype uses two different message protocols for agent communications. ZeroMQ has been used with Volttron, while RabbitMQ is under development. This subsection briefly describes their technical details. Their performance is evaluated and compared in detail in Section 5.

4.2.1. ZeroMQ

ZeroMQ is a high-performance message library that provides a socket-style API and a message queue for message exchanges [56]. It is designed for high throughput and low latency. Background threads perform input/output asynchronous processing and use a single call queue or proxy for communications. It is lightweight enough to process data fast and thus is suitable for small numbers of large messages as well as for a large volume of small messages [57]. ZeroMQ supports three types of communication patterns without a message broker. It allows a request-reply pattern using a remote procedure call (RPC); a publish/subscribe pattern is supported by an asynchronous queue; and in a push-pull pattern, a sender distributes a message to the multiple, pulling receivers arrayed in a pipeline.

Volttron extends ZeroMQ and implements a routing protocol, named the Volttron Interconnect Protocol (VIP). VIP uses a router-dealer pattern in ZeroMQ. A router is a fully asynchronous, non-blocking socket that accepts a request from a sending node and forwards it to a message queue. An interconnecting code passes the request to a dealer that is another, fully asynchronous, non-blocking socket. The dealer manages the distribution of such a request to receiving node(s). In the meantime, the router tracks every connection to the sending node so that it can pass the response back to the requestee, when a response comes back. VIP defines how nodes connect to (bind) two sockets and the messages they exchange.

4.2.2. RabbitMQ

RabbitMQ is an open source software that implements an Advanced Messaging Queuing Protocol (AMQP), a protocol designed for providing loose coupling, asynchronous publish/subscribe pattern communications based on topics [58]. Figure 4 illustrates the structure of RabbitMQ, which consists of many components: publisher, consumer, broker, exchange, queue, and binding (route). A publishing node sends a message with a list of attributes describing itself and the content of the message to the exchange in the broker. The exchange then performs the task of distributing a copy of the message to an appropriate queue after applying the attributes to a rule called binding. The broker forwards the message to a consumer subscribing to the queue, or a consumer fetches/pulls the message from the queue on demand. When the exchange fails to forward the message to other queues, the exchange removes the message or returns to the publisher.

RabbitMQ is not fully adopted in Volttron yet, but is expected to provide a few advantages. It, as a network protocol, allows publishers, consumers, and brokers to be deployed in distributed locations and provides mechanisms for distributed collaboration, such as clusters and federation. These make RabbitMQ more scalable than ZeroMQ. In addition, RabbitMQ uses TCP, which enables

reliable transmission through acknowledgement. Consumption confirmation informs the broker of whether a consumer can process the message, which reduces the number of unprocessed messages.

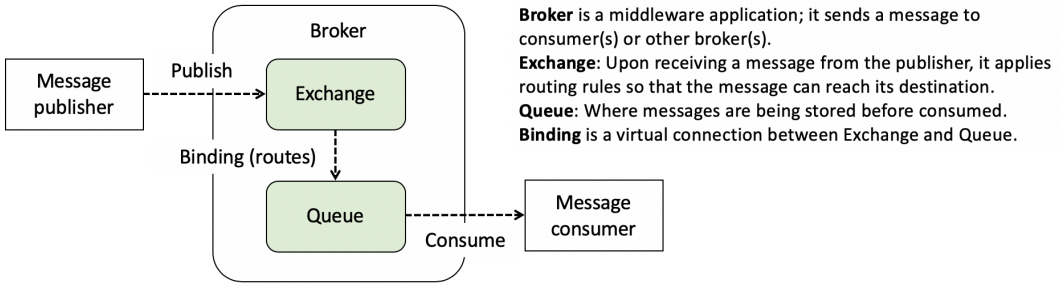


Figure 4. RabbitMQ includes many components: publisher, consumer, broker, exchange, queue, and binding (route).

4.3. Developing Agents

This subsection develops a few agents that can run together on a fog node: two sensing/action agents, two Internet agents, an intelligent agent, and a user agent.

4.3.1. Sensing and Action Agent

A sensing/action agent directly communicates with end devices and is responsible for both collecting data from them and sending commands to control their status. Devices may use different protocols and communication technologies. To accommodate the diversity, our prototype implements two sensing/action agents. A Modbus agent communicates with end devices via Modbus, a de facto standard communication protocol in connecting industrial electronic devices. We also developed an XML agent that transfers messages in the form of XML or JSON, and this type of agent is currently widely used in end devices such as smart lights and smart plugs. These two agents periodically sense the status of end devices and publish the data records to the information exchange bus. In the meantime, they cache the status information. The agents also receive commands from other agents via the bus. If a command includes a new state of an end device that is different from the cached record, they send out a new message to the device to change its status. Each agent maintains an interface table for each device that specifies detailed information about the device, including data points, an interface port, and a brief description. This way, an agent registers, communicates, and manages end devices.

4.3.2. Internet Agent

Our prototype implements two Internet agents: a price agent and a weather agent. A price agent obtains a system marginal price (SMP) from the Korea Power eXchange (KPX) [59] and distributes it to other agents via the message bus. The SMP is determined by the power market price applied to the electric power of each transaction hour. It changes dynamically; for instance, the price becomes high on high demands. The agent also provides the time of use (ToU) price that changes slowly. The KPX provides a ToU table every year, and the price agent stores it in a local file. It sends the price data to agents only upon receiving requests. A weather agent obtains weather data from the Korea Meteorological Administration (KMA) [60] and publishes it to the message bus so that other agents can use the weather information for their own purposes. The KMA updates weather data every 3 h starting at 2:00 a.m. Data specify a base date and time and include forecasting information in 13 categories in Table 4.

Table 4. Weather forecasting data in 13 categories provided from the Korea Meteorological Administration (KMA).

Index	Description	Index	Description
POP	Precipitation probability	S06	6 h of precipitation
PYT	Precipitation type	R06	6 h of snow cover
T3H	3 h temperature	TMX	Daytime maximum air temperature
REH	Humidity	TMN	Daytime minimum air temperature
WSD	Wind velocity	UUU	Wind velocity (east or west)
VEC	Wind direction	VVV	Wind direction (north or south)
SKY	Sky condition		

4.3.3. Intelligent Agent

Our prototype implements an intelligent agent in which any data processing and analysis algorithms can be applied. A machine learning algorithm can run by consuming data received from the message bus, or a rule can be developed in the agent that evaluates sensor data and controls end devices automatically. Our agent is configured with two rules. It adjusts the brightness of a smart light after reading data from a light sensor attached in an office. It also changes the set temperature of a thermostat along with the real-time SMP transmitted from the price agent.

4.3.4. User Agent

A user agent in the prototype displays data to a human user and accepts inputs from the user. To this end, it is implemented as a web page and thus can run on a fog node that implements a web service component. It receives data from the message bus (e.g., price, weather, and status of devices) and shows it in various formats, e.g., as a graph and as text. The user agent allows a user to control the status of end devices; for instance, they can turn off a smart light and adjust a set temperature on a thermostat.

4.4. Building a Testbed with Prototype

After developing a prototype of a fog node with multi-agents, we built a testbed of Fog BEMS by deploying fog nodes hierarchically and installing a set of end devices on our campus.

4.4.1. Fog Layer Deployment

The testbed deployed seven fog nodes in a hierarchy as shown in Figure 5. The building in the figure, out of 30 buildings on a campus, is a home for three departments of IT technology and has five stories. Each floor has around 25 rooms: 5∼6 classrooms and offices for faculty members, staff, and graduate students. Classrooms are either small or medium in size; 50∼60 students can sit in the biggest classroom with computers. Around 1300 students, enrolled in the departments, use the building every day. We set five zones at 5th floor in the building, each of which had a fog node (i.e., five zone fogs). Two additional fog nodes were installed at the building level and at the campus level: a building fog and a campus fog. Each zone fog was implemented on a Raspberry Pi and ran sensing and action agents so that it could interact with end devices. It performed the data merging algorithm and periodically forwarded the data of end devices to the campus fog via the building fog. It also executed an intelligent agent performing locally automated energy management. The building fog ran Internet agents so that it obtained both weather data and price data from the Web and distributed them to the zone fogs on demand. It also had a small screen through which a user agent could interact with a human user. The campus fog was implemented on a desktop PC with a memory agent so that it permanently stored the collected data of end devices. All the fog nodes ran on the aforementioned software platform and executed a communication agent implementing the message protocol for agent communications.

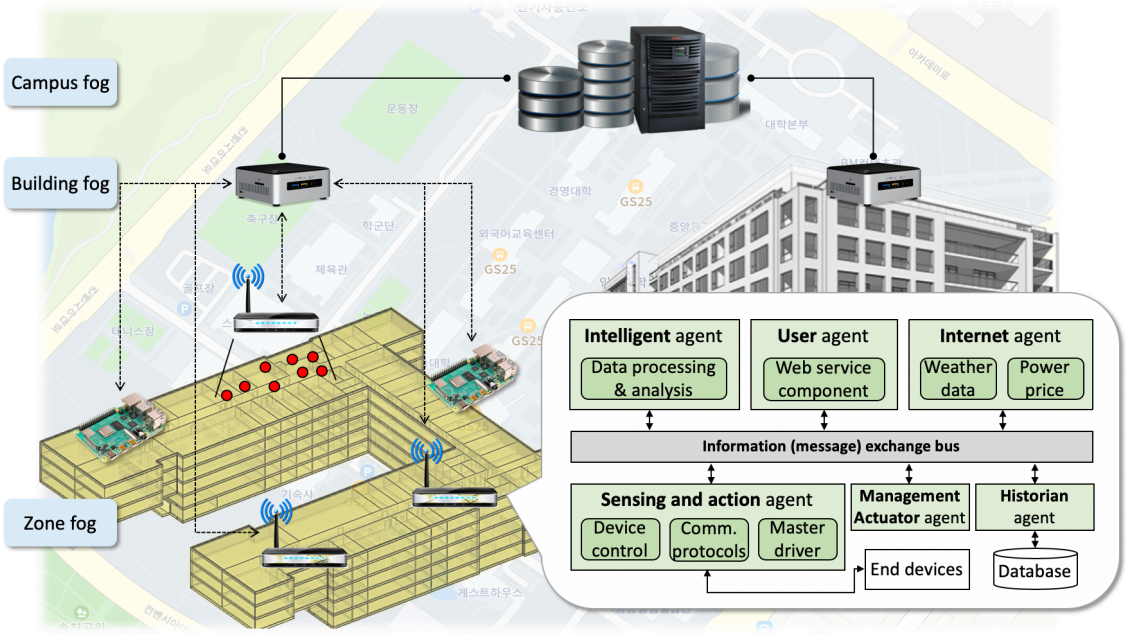


Figure 5. A Fog BEMS testbed was built by deploying fog nodes developed and IoT devices on a campus.

4.4.2. End Devices

The testbed installed a set of sensors. A temperature/humidity sensor, a brightness sensor, and a presence sensor placed at each zone communicated with a zone fog via Bluetooth. Each zone also had a list of smart devices: Philips Hue lights, smart plugs, and a thermostat, as pictured in Figure 6. Each device maintained multiple data points in it, and these data are summarized in Table 5. The light bulb was able to turn itself on/off and change its brightness and color. The thermostat maintained five data points with which it could adjust its own status in various ways. These two devices communicated with the XML agent via WiFi. The plug measured data related to power and energy and used a Modbus/TCP protocol for communications. We also developed a Modbus simulator from which operations of a variety of energy devices could be implemented. This enabled the deployment of many devices connecting to the Modbus agent, helping us to evaluate the performance of the testbed on a large scale.

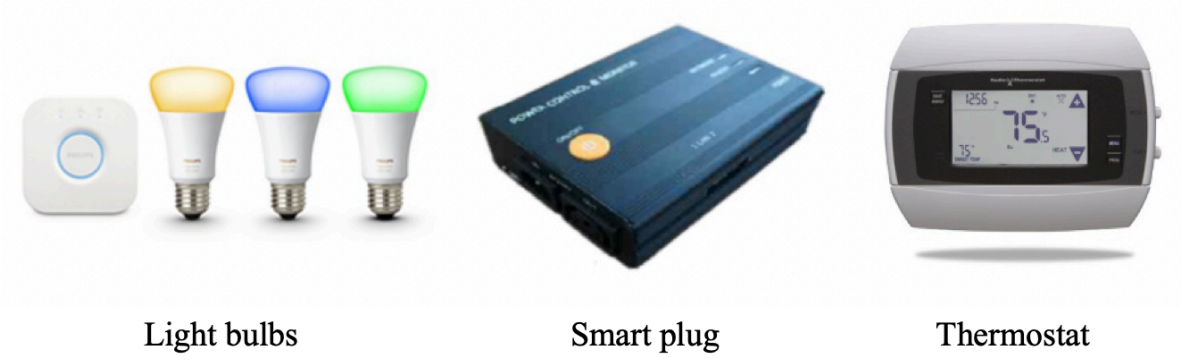


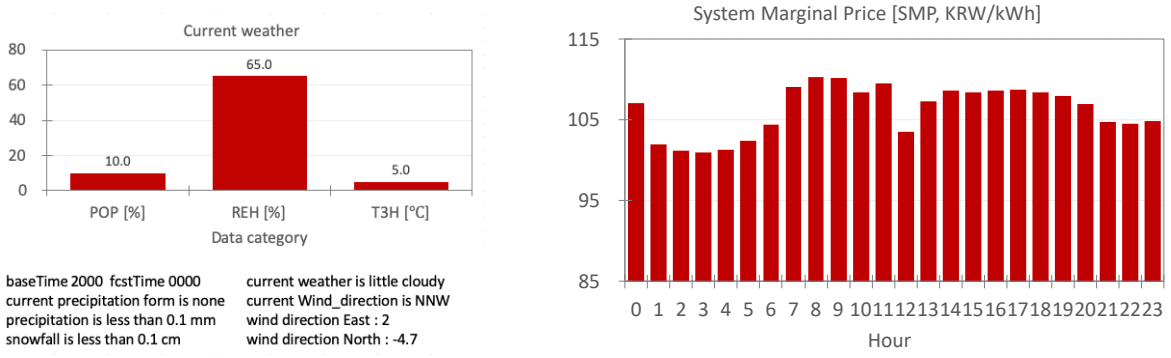
Figure 6. Smart devices used in the testbed: light bulbs, smart plugs, and thermostats.

Table 5. Each device maintained multiple data points providing different functionalities.

Devices	Classification of Data Points	Description
Smart light	power control	to turn on/off the light
	dimming control	to dim the brightness
	color control	to change the color
Smart plug	power control	to turn on/off the plug
	voltage status	to access voltage data
	current status	to access current data
	energy status	to access energy data in Watt
Thermostat	power control	to turn on/off the device
	current temperature	to access data of current temperature
	target temperature	to set target temperature in a cool mode
	fan mode control	to turn on/off a fan mode
	status mode control	to change the mode status

4.4.3. Demonstration

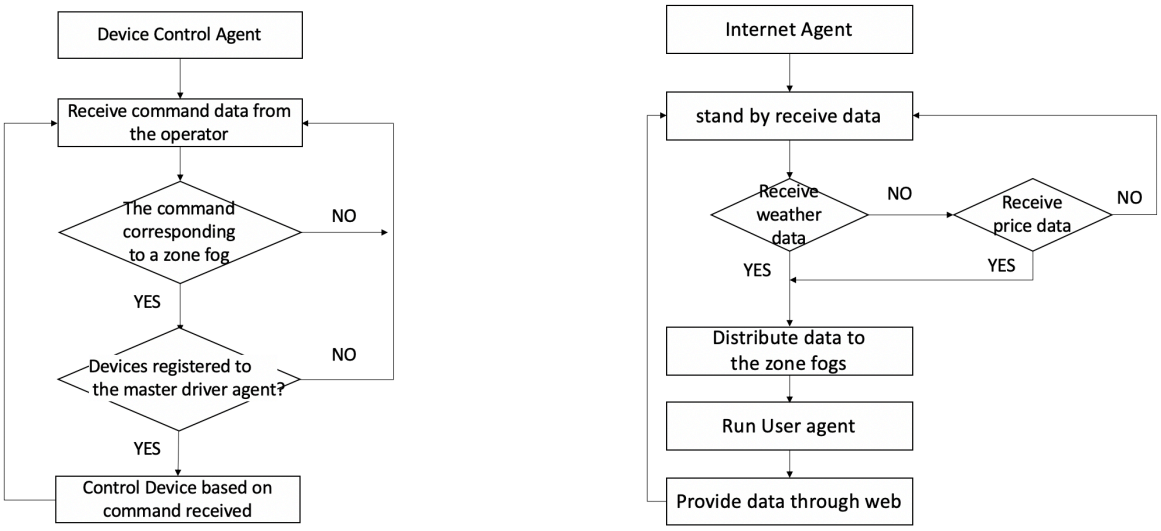
The testbed supported three types of energy management operations. It displayed data on a screen in the building fog and allowed a user to control end devices manually. The intelligent agent performed two rules, automating device controls along with external inputs (brightness data and price data). As an example, Figure 7 shows weather data (on the left) and price data (on the right) that the Internet agent obtained in the winter and forwarded to the user agent. "baseTime" in the weather data indicates the time when the data was updated by the KMA, while "fcstTime" indicates the forecasting time. That is, data announced at 8:00 p.m. forecasted weather at midnight. Forecasting data are represented in bars and texts. Precipitation probability was 10%, humidity was 65%, and 3 h temperature was 5 °C. On the right is the SMP obtained and published by the price agent. The x-axis denotes the transaction time over 24 h, while the y-axis represents the price values in KRW/kWh. Figure 8 shows activity diagrams. A device control agent takes commands from an operator and executes a control action, whereas an Internet agent fetches data from the Internet and delivers it to a user agent for additional display.



(**a**) Weather data

(**b**) System marginal price [KRW/kWh] over 24 h

Figure 7. User agents displayed weather data and price data on a screen.



(**a**) To control devices (**b**) To handle weather and price data

Figure 8. Activity diagrams for controlling devices and for processing data received from the Internet.

5. Performance Evaluation

This section evaluates the performance of the proposed system, the Fog BEMS, especially with respect to scalability and communication overhead.

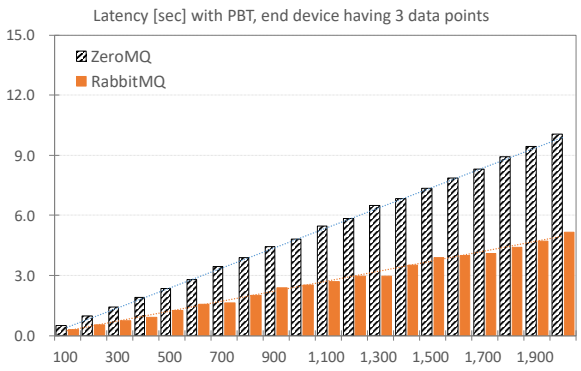
5.1. Scalability of Fog BEMS

We evaluated the scalability performance of Fog BEMS. This subsection is focused on device scalability, the performance of which is measured by the maximum number of devices that a fog node can communicate with in a given time deadline. To this end, we considered a data collection scenario where a fog node is devoted to collecting data from end devices. The more end devices connected to the node there are, the more time it takes. In our experiments, we increased the number of devices and the number of data points in a device and then recorded the latency for data collection, a period of time during which the node contacts all connected devices and gathers data from them. We then computed the maximum number of devices with a latency of 1 s.

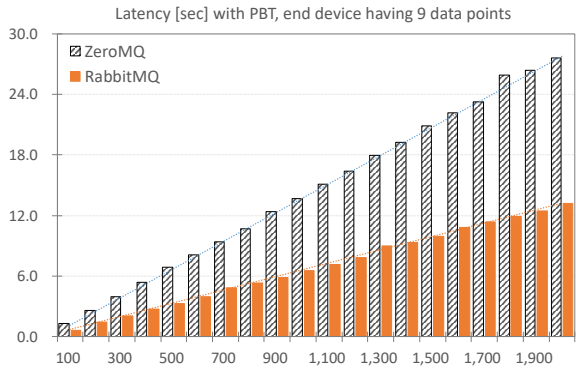
We deployed a fog node that ran a sensing/action agent on top of a Linux machine with an Intel i7-7700HQ CPU @ 2.80 GHz. End devices were deployed and connected to the node; we used simulated devices for scalable experiments. The node supported two different types of transmission methods when collecting energy data from end devices and publishing it to a message bus. A point-based transmission (PBT) creates a topic message for each data point and publishes it to the bus. Table 5 shows that a smart light maintains three data points: status, dimming, and color. In a PBT, data are published in three topic messages—"light/status/", "light/dimming/", and "light/color/." Other agents can read their current values from these messages. That is, the node publishes three separated messages individually. In a device-based transmission (DBT), on the other hand, the node merges three data into one topic, "light/all/", and publishes one message only. A DBT minimizes the number of messages to be sent, but it can waste message space (size) if other agents do not require all three data. We also applied two message protocols, ZeroMQ and RabbitMQ, to the node in order to compare their performance. In the experiments, the number of devices increased from 100 to 2000, and the number of data points in a device increased from 3 to 9 and 18. The latency was then measured and evaluated.

5.1.1. Point-Based Transmission

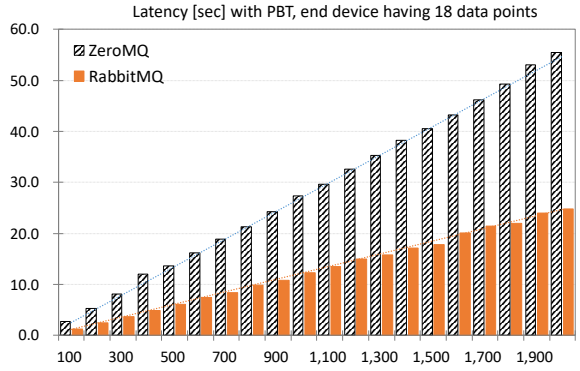
The first experiments used a PBT; a fog node published a message for each data point. Figure 9 demonstrates the experimental results. The first graph shows the experimental results when the end device had three data points. The x-axis represents the number of devices, and the y-axis represents the measured latency in seconds. With 100 devices, 300 data were collected and published individually. It took 0.5 and 0.28 s when using ZeroMQ and RabbitMQ, respectively. Latency increased to 4.8 and 2.54 s when the number of devices increased to 1000. With 2000 devices, it took 10.06 and 5.18 s with ZeroMQ and RabbitMQ, respectively. Latency with 3 data points increased at a rate of 18%, as the number of devices increased in both message protocols. Comparing the two protocols, ZeroMQ takes longer than RabbitMQ; the mean difference is 2.55 s. With ZeroMQ, it takes 1.65 ms on average for a fog node to collect and publish one data (i.e., a message in a PBT). In the case of RabbitMQ, the transmission time per data (TPD) was 0.88 ms. This shows that that RabbitMQ is 1.87 times faster than ZeroMQ. This result is against our notion about the protocols. ZeroMQ is lightweight enough to process data fast, so it is known to provide low latency, while RabbitMQ has an advantage in reliability by using an acknowledgement, which may degrade latency. We investigated this further and provide our analysis at the end of this subsection.



(**a**) Each device has 3 data points. (PBT)



(**b**) Each device has 9 data points. (PBT)



(**c**) Each device has 18 data points. (PBT)

Figure 9. With a point-based transmission (PBT), a fog node publishes one message for each data point. The x-axis represents the number of devices, and the y-axis represents the latency measured in seconds.

The experimental results with end devices with nine data points are shown in Figure 9b. In the case of 100 devices, 900 data were collected and published individually, and ZeroMQ and RabbitMQ took 1.32 and 0.62 s, respectively. When increasing the number of devices to 2000 (i.e., 18,000 data points), it took 27.59 and 13.17 s with ZeroMQ and RabbitMQ, respectively. Results also show that RabbitMQ

is 2.14 times faster than ZeroMQ; the TPD was 1.51 ms on average with ZeroMQ and 0.72 ms with RabbitMQ. The next experiment with 18 data points also showed similar values, which were then compared with those in the previous experiment, 1.65 and 0.88 ms. The average difference of 0.15 ms is mainly attributed to the data collection procedure. When an end device sent data to a fog node, it transmitted a packet that contains all the information of every data point , just like the DBT. Therefore, it can save time with more data points, and our results discover that this procedure can benefit latency when the device maintains at least nine data points.

The bottom graph shows the experimental results of the case with 18 data points. With 2000 devices, 36,000 data were collected and published individually. It took 55.5 and 24.74 s when using ZeroMQ and RabbitMQ, respectively. The growth rates in latency were 19 and 18% on average with ZeroMQ and RabbitMQ, and these values are almost the same as those in the two previous settings. This indicates that latency is linearly proportional to the number of end points in a PBT.

5.1.2. Device-Based Transmission

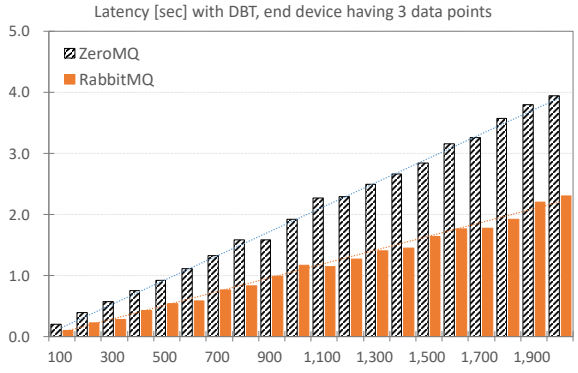
The next experiments used the DBT; a fog node published a message for each device. Figure 10 demonstrates the experimental results. The first graph shows the experimental results when the end device has three data points. With 100 devices, 300 data were collected, and 100 messages were published. It took 0.2 and 0.1 s when using ZeroMQ and RabbitMQ, respectively. Latency increased to 3.95 and 2.31 s when the number of devices was increased to 2000. The growth rate in latency was 18–20% on average in both protocols, and the following settings also led to the same values. This indicates that latency is linearly proportional to the number of end devices in a DBT. Results show that RabbitMQ is 1.78 times faster than ZeroMQ; the TPD values in ZeroMQ and RabbitMQ were 0.66 and 0.37 ms. These values were compared with those in the PBT: 1.65 and 0.88 ms, as summarized in Table 6. Using ZeroMQ, the TPD was almost 1 ms, a reduction of 60%; using RabbitMQ, the TPD was 0.51 ms, a reduction of 57.95%. The savings are mainly attributed to the fact that a DBT reduced the number of messages to be published from 300 to 100. These savings are specially highlighted in the following results, as the number of data points increases.

Table 6. Performance comparison with three varying parameters: message protocols, transmission methods, and the number of data points.

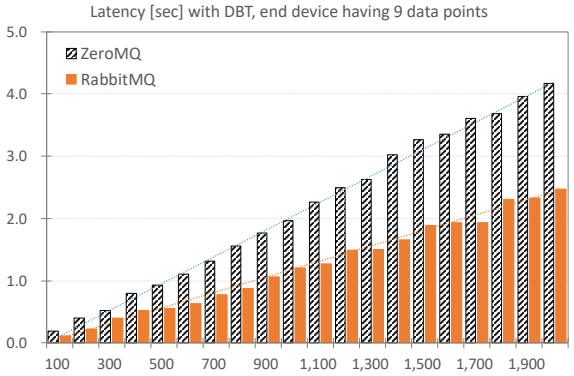
	Transmission Methods	**Point-Based Tx. (PBT)**			**Device-Based Tx. (DBT)**		
	Number of Data Points (dp)	3 dp	9 dp	18 dp	3 dp	9 dp	18 dp
ZeroMQ	Total latency w/2k devices [sec]	10.06	27.59	55.05	3.95	4.17	4.51
	Tx. time per data (avg) [ms]	1.65	1.51	1.53	0.66	0.22	0.13
RabbitMQ	Total latency w/2k devices [sec]	5.18	13.17	24.74	2.31	2.47	2.86
	Tx. time per data (avg) [ms]	0.88	0.72	0.70	0.37	0.13	0.08

The following two graphs (Figure 10b,c) show the experimental results with end devices with 9 and 18 data points, respectively. When using ZeroMQ and 2000 end devices, it took 4.17 and 4.51 s with 9 and 18 data points (18,000 and 36,000 data in total), respectively. The DBT substantially saved collection latency by 23.42 and 50.99 s compared with that in the PBT. This impact is also represented in TPD values in the DBT. The TPD with 9 and 18 data points measured 0.22 and 0.13 ms on average, respectively, which is a reduction of 85.43% (1.29 ms) and of 91.5% (1.4 ms) compared to the PBT cases. The more data points an end device maintains, the stronger the impact becomes. Note that the TPD was 0.66 ms, and the improvement was 60% with 3 data points. In particular, when there are more than 9 data points, the benefit is maximized by over 85%. It is also interesting to observe that the collection latency with 2000 devices measures 3.95, 4.17, and 4.51 s as number of data points increases. The differences primarily explain the overhead due to the increased payload size in a published message. The saving effect in RabbitMQ is summarized as follows. The collection latency values

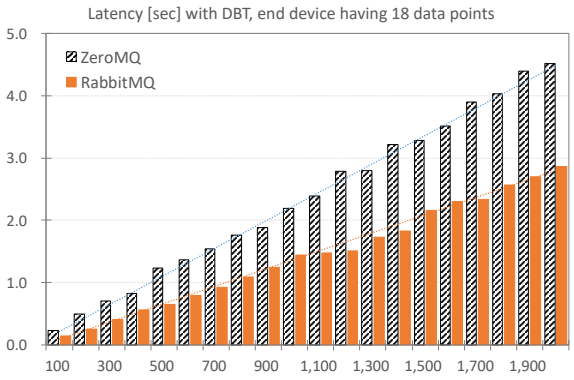
with 9 and 18 data points are, respectively, 2.47 and 2.86 s, which are decreased from 13.17 and 24.74 s in the PBT. The TPD values are 0.13 and 0.08 ms, which are decreased from 0.72 and 0.7 ms; they are improved by 81.94% and 88.57%, respectively. When comparing the two protocols, RabbitMQ outperforms ZeroMQ; in particular, the TPD goes below 100 μs when there are more than 18 data points. On the other hand, a DBT benefits ZeroMQ more; there is a 91.5% improvement at maximum with ZeroMQ, higher than the 88.57% improvement with RabbitMQ. The following explains this.



(**a**) Each device has 3 data points. (DBT)



(**b**) Each device has 9 data points. (DBT)



(**c**) Each device has 18 data points. (DBT)

Figure 10. With a device-based transmission (DBT), a fog node publishes one message per device. The message may include multiple data once the device maintains multiple data points. The x-axis represents the number of devices, and the y-axis represents the latency measured in seconds.

Previous research has reported that ZeroMQ is faster than RabbitMQ [61], but our experiments show the opposite. This is mainly attributed to a specialized implementation of ZeroMQ on the software platform. After an agent in the platform publishes a message via ZeroMQ, it receives an acknowledgement that is not defined in the specification of the protocol. The additional implementation aims at reliable communications that ZeroMQ does not provide. With acknowledgement, an agent can notice when a buffer overflow occurs and thus slow down the message transmission rate. On the other hand, RabbitMQ relies on TCP, so acknowledgement is automatically provided for reliable message transmission. This difference also affects the performance improvement in the two protocols. A DBT reduces the number of messages published, and so does the overhead due to the acknowledgement. Thus, ZeroMQ can improve performance more than RabbitMQ in a DBT.

5.1.3. Device Scalability

The goal of the scalability experiments was to figure out the maximum number of devices with which a fog node can communicate in a given time deadline. Table 7 lists the number of end devices and data points that are processed in less than 1 s, with different settings of message protocols and transmission methods. As shown, the DBT outperforms the PBT. When using ZeroMQ with a DBT, a fog node can communicate with 500 end devices. If RabbitMQ is used, up to 900 devices can be deployed in a zone and connected to the node. Therefore, RabbitMQ outperforms ZeroMQ in terms of scalability. Our experimental results can be utilized when developing a fog layer. The first questions faced in the development are of where a fog node should be deployed and of how many nodes are necessary. They become a problem when deciding the size of a fog zone. Given the number of end devices deployed, our results can provide a guideline for answering these question.

Table 7. We computed the maximum number of devices with which a fog node can interact in 1 s.

	Number of Data Points	**Number of Devices**	**Latency Time [s]**
ZeroMQ w/PBT	3	200	0.97
	9	-	-
	18	-	-
ZeroMQ w/DBT	3	500	0.92
	9	500	0.93
	18	500	0.83
RabbitMQ w/PBT	3	400	0.93
	9	100	0.62
	18	-	-
RabbitMQ w/DBT	3	900	0.98
	9	800	0.88
	18	700	0.93

5.2. Communication Overhead in a Fog Layer

We evaluated the performance of the proposed data merging algorithm. The eventual goal of the algorithm is to diminish the network overhead in a fog layer. This is achieved by reducing the volume of energy data periodically collected from end devices and forwarded to a central database.

We set up a testbed system for a fog layer by deploying fog nodes hierarchically and installing end devices in a zone. We considered a university building scenario, where each zone takes care of each floor in a building. To this end, three types of simulated devices were deployed in a zone: 350 smart lights with three data points, 100 smart plugs with four data points, and 50 thermostats with five data points. They periodically reported their status information to a fog node in the zone (i.e., a zone fog). Upon collecting data from the same type of devices, the fog node grouped the same types of data points and merged them into one message before forwarding that message to an upper level. A campus fog then stored the received data in a permanent database. In order to assess performance, in our experiments, we recorded the total numbers of messages that the fog node forwarded and their sizes in bytes. We also measured the same two metrics when using traditional transmission methods: PBT and DBT. By comparing the outputs, we show how much communication overhead is diminished in the fog layer. We note that more experiments have been conducted with two more scenarios: a residential house and an office room. However, those results are not presented here due to space limitations. Instead, we used them when representing the performance benefit of the proposed algorithm in a generalized form at the end of this evaluation.

5.2.1. Traffic Volume at an Individual Device

Each smart light maintains three data points: status, dimming, and color (please refer to Table 5). In the PBT, the data of each point is converted to a message that the node forwards individually. In our experiment, we measured the size of a message; the sizes of the status, dimming, and color messages were 619, 625, and 615 bytes, respectively. Since 350 devices were installed, 1050 messages in total were forwarded, and their message sizes totaled 650,650 bytes. Table 8 summarizes the experimental results with 350 smart lights. In the DBT, the data of three points was combined into a message whose size was 794 bytes. With 350 devices, the node transmitted 350 messages totaling 277,900 bytes over a fog network. In the proposed algorithm, the node merges data of the same point from 350 devices into a message. Thus, the sizes of the status, dimming, and color messages were 6062, 6063, and 6061 bytes, respectively. However, the total number of messages sent out reduced to 3, and their size amounted to 18,186 bytes only. The proposed algorithm reduced the total number of messages by 99.7% and 99.14% compared with the PBT and the DBT, respectively. In terms of the total message size, it improved performance by 97.2% and 93.46%.

Table 8. We evaluated communication overhead by measuring two metrics with 350 smart lights, 100 smart plugs, and 50 thermostats.

Device Types	Evaluation Metrics	Traditional Transmission Method		Proposed Algorithm
		Point-Based Tx.	Device-Based Tx.	Data Merging
Smart lights	Number of messages Tx. (total)	1050	350	3
	Message size (total) [byte]	650,650	227,900	18,186
Smart plugs	Number of messages Tx. (total)	400	100	4
	Message size (total) [bytes]	244,600	87,000	8803
Thermostats	Number of messages Tx. (total)	250	50	5
	Message size (total) [bytes]	155,500	48,950	6627

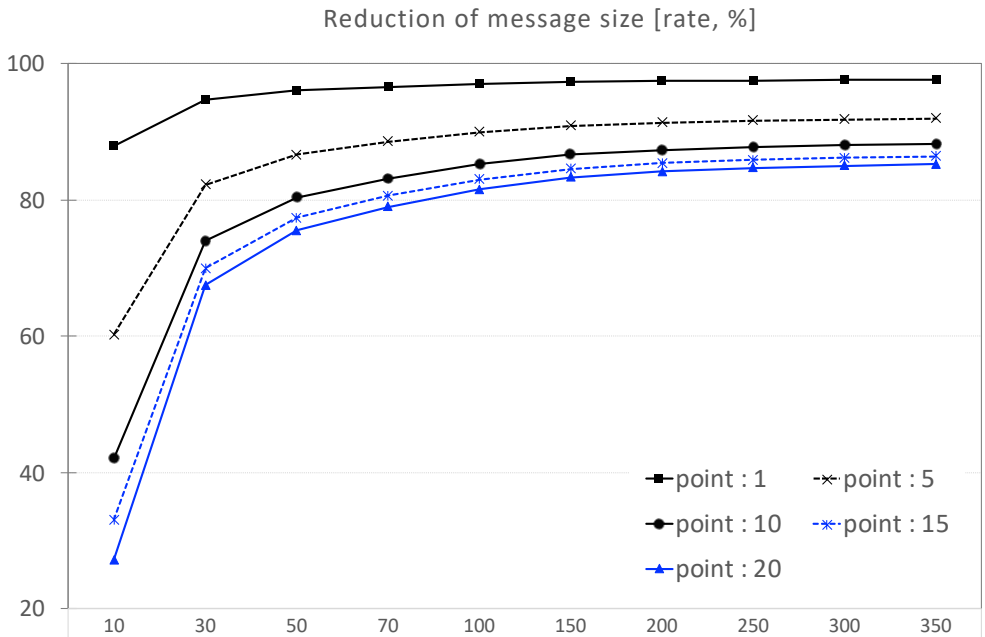
Each smart plug maintains four data points: status, voltage, current, and energy. Table 8 also summarizes the experimental results with 100 smart plugs. In the PBT, the sizes of the status, voltage, current, and energy messages were 618, 614, 622, and 612 bytes, respectively. With 100 devices, 400 messages totaling 244,600 bytes were forwarded. In the DBT, the fog node transmitted 100 messages totaling 87,000 bytes over the fog network. In the proposed algorithm, the four merged messages of status, voltage, current, and energy were 2022, 2018, 2027, and 2016 bytes in size, respectively. The proposed algorithm reduced the total number of messages by 99% and 96%, compared with the PBT and the DBT, respectively. In terms of the total message size, it improved performance by 96.4% and 89.88%.

Each thermostat maintains five data points: status, currentTemp, targetTemp, fanMode, modeStatus. Our experiment measures that PBT transmits 250 messages totaling 155,500 bytes, while the DBT transmits 50 messages totaling 48,950 bytes. In the proposed algorithm, the fog node forwards 5 messages totaling 6627 bytes, which is a reduction of 98% and 90% in terms of the total number of messages and 95.74% and 86.46% in terms of the total message size, compared with the PBT and the DBT, respectively.

5.2.2. Overhead Reduction With Varying Numbers of Devices and Data Points

As mentioned before, we conducted experiments in three scenarios in total, and each scenario varied the number of end devices and data points. Based on all of the results, we took the total size of the messages transmitted in the DBT and the proposed data merging algorithm. We then computed the reduction in the amount of network traffic; for instance, when the message size decreased from 22,670 bytes with the DBT to 16,500 bytes with the merging algorithm, the traffic overhead was reduced by 27.22%.

Figure 11 illustrates the reduction rate of message sizes. The x-axis denotes the number of devices. Values are given with different numbers of data points: 1, 5, 10, 15, and 20 points. The graph shows that the reduction rate increased as the number of devices increased, and this rate eventually saturated. When there were more than 50 devices, there was a traffic reduction of more than 70% over all data points, and of more than 80% with 200 devices. In a large scale environment such as the university building scenario, many devices of the same type are likely to be deployed, so the merging algorithm can diminish traffic overhead significantly. When deploying 50 smart lights with three points in a hallway, the algorithm reduced the traffic volume by more than 86.69%. The reduction rate increased as the number of data points decreased. When there were 10 devices, the rates were 87.92%, 60.28%, 42.05%, 33.06%, and 27.22% with 1, 5, 10, 15, and 20 points, respectively. The maximum difference was 60.7%, which is explained as follows. With one data point, a fog node transmitted one message, and it transmits 20 messages with 20 points. As the number of devices increased to 30, the difference reduced rapidly to 27.28%. It decreased to 12.35% with 350 devices. This implies that the merging algorithm does not diminish the traffic overhead much in a small scale environment with a very small number of devices with many data points. However, such a setting receives the biggest benefit as the number of devices increases. We note that the proposed Fog BEMS is expected to be deployed in a large scale campus environment. Many fog nodes comprise a fog network, and a fog layer includes a number of end devices. Therefore, the merging algorithm can contribute to the alleviation of communication overhead in a new BEMS.



	10	30	50	70	100	150	200	250	300	350
point : 1	87.92 %	94.76 %	96.06 %	96.62 %	97.03 %	97.31 %	97.45 %	97.53 %	97.59 %	97.63 %
point : 5	60.28 %	82.29 %	86.69 %	88.58 %	89.97 %	90.91 %	91.39 %	91.66 %	91.85 %	91.99 %
point : 10	42.05 %	73.97 %	80.41 %	83.19 %	85.26 %	86.63 %	87.33 %	87.74 %	88.02 %	88.21 %
point : 15	33.06 %	69.98 %	77.41 %	80.61 %	83.00 %	84.60 %	85.39 %	85.87 %	86.19 %	86.42 %
point : 20	27.22 %	67.48 %	75.53 %	78.99 %	81.59 %	83.31 %	84.17 %	84.69 %	85.04 %	85.28 %

Figure 11. The reduction rate saturates as the number of devices (x-axis) increases.

6. Discussion

6.1. Security Concerns

Since Fog BEMS encourages interactions among nodes and agents for collaborations, it is of utmost importance to ensure that communications are secured and protected appropriately. With respect to message protocols, VIP makes use of security features of ZeroMQ to implement encrypted

and authenticated communications over a shared socket. It extends the ZeroMQ Authentication Protocol (ZAP) [62] by including the ZAP User-Id in the VIP payload, which also allows nodes to authorize access based on ZAP credentials. It uses CurveZMQ for secure messaging across the Internet [63]. RabbitMQ supports Secure Socket Layer (SSL), which allows for a secure service for exchanging data between agents and between fog nodes. Each node has a single self-signed root CA (Certificate Authority, a root certificate) and maintains a server certificate for RabbitMQ. Every time an agent is started, it creates a pair of private and public certificates that is signed by the root CA. When the agent communicates via the information exchange bus, RabbitMQ validates its public certificate. Because there is only a single root CA in the node, all the agents in it communicate with the bus over SSL. In order to secure communications between nodes, they share individually owned root certificates. A node (say, *X*) maintains a trusted CA certificate as a file. When another node (say, *Y*) wants to connect to *X*, *Y*'s root certificate is appended to the file in advance. *Y* presents its root certificate during connection, so that *X*'s RabbitMQ knows it is safe to talk to *Y*. It is also possible to develop a special agent that plays a major role in monitor communications. An agent can alert an administrator when an abnormal topic, which is not in a predefined whitelist, is published to the message bus. This agent can be extended to an anomaly detection system.

6.2. Use Cases

In terms of energy, a smart campus represents a system that aims at maximizing energy efficiency and comfort of campus residents. One of the outstanding features is that it consumes data that had barely been in digital format but now can be digitalized by emerging IoT technologies as well as traditional energy data. This paper proposes a distributed computing model that enables to acquire and process such new types of data. We note that the proposed model addresses computing resources on which various energy algorithms and energy functionalities are developed. The proposed computing model differs from conventional computing model like Cloud in that the new one operates in a distributed environment and actively interacts with our physical environment whose data is digitalized by end devices. Thus, it is tightly related to devices, which requires that the model should be informed of contextual information of the devices in advance. This is achieved by defining a *zone* in our model. As noted in Section 3.1, the *zone* enables computing resource to connect to nearby devices and to identify a physical boundary. At the same time, the resource learns how to communicate with the devices. A *building* is defined as a collection of related *zone*s, and a *campus* as a collection of *building*s and *zone*s that are related physically or logically. A new use case can be created by defining a zone after deploying end devices. Section 5.2 considers a campus building scenario. Three types end devices (smart lights, smart plugs, and thermostats) were deployed, and a zone was defined. Since these devices are generally instrumented in most buildings, this scenario can be applied to different buildings. Note that a designer can add more devices and apply different criteria to define the zone. As an example, Table 9 lists a set of end devices used when defining a zone for a mid-sized smart office in our research.

Table 9. A list of end devices used in experiments of a smart office scenario. We conducted the experiments, but performance results are omitted due to space limitation.

Devices	Classification of Data Points	Num of Devices
HVAC	power [on/off]; mode [cool/heat]; fan; current temp; target temp	11
Smart light	power control; dimming control; color control	180
Smart plug	power control; voltage status; current status; energy status	70
Presence sensor	power [on/off]; presence [y/n]	6
Fire alarm	power [on/off]; heat sense; spark sense; fire sense	10

6.3. Extension

The proposed Fog BEMS model shows a list of salient features; decentralized operations, context-awareness, local intelligence, and open architecture. These help smart grid functionalities such as demand response. We note that our model does not replace an existing Cloud computing architecture and any energy management systems using the Cloud. Instead, concepts and portions of our implementations can be applied to other energy systems. For instance, decentralization can support peer-to-peer energy management platforms. As noted in [64], a decentralized approach for resource allocation and load scheduling leads to a scalable solution for the management of the multi-commodity smart energy system. Fog BEMS works as a computing platform for decentralized processing of such energy functions. A fog node can become intelligent by adopting intelligence algorithms. This implies that the node is able to manage both power generation and consumption within its boundary and to make an energy-related decision. In this way, it eventually supports the behavior of a prosumer, i.e., both a consumer and producer of energy in the same entity [65]. Context-awareness can support higher level energy management platforms such as urban energy management decision support systems [66,67]. Calvillo et al. [66] points that it is the most challenging to minimize power consumption without compromising user comfort. To this end, a manager must take into account user demand in energy operations. A fog node in the proposed Fog BEMS interacts with end users and devices and is able to acquire local context. The manager may use such context to understand user demand, which is helpful for developing operation strategies. Open architecture can support lower level energy management platforms such as building/home subsystems. Many excellent algorithms have been proposed for efficient operations of demand side management systems, and most of them have been validated through a simulated case study [68,69]. Fog BEMS can provide an implementation technology on which the algorithms can be realized. A robust model predictive control based system [68] assumes incorporation of a number of interconnected smart homes. Since Fog BEMS has an open software architecture, it is able to connect to heterogeneous types of devices that have been hardly incorporated with proprietary architectures. A fog node can easily implement various energy functions like demand response, prediction, long-term scheduling, and real-time control [69]. For example, a Virtual End Node (VEN) for Open Automated Demand Response (OpenADR) is developed as either a communication agent or an intelligent agent in our proposed system.

7. Conclusions

This paper proposes Fog BEMS that applies an emerging fog computing architecture to an energy management system to cope with increasing numbers of end devices. Fog computing with distributed resources processes user data in real time and uses local contextual information, making a BEMS scalable. Fog BEMS tackles two limitations of existing fog computing models: device scalability and functional heterogeneity. To overcome them, we designed a fog layer of a location-aware, hierarchical architecture including a data merging algorithm to make a fog network lightweight and scalable. We also propose an agent-based interoperable programming model, which can make the development of fog nodes more flexible and easier. We developed a prototype of Fog BEMS and built a real-world testbed on a campus to demonstrate the feasibility of the proposed system. Scalability was evaluated in terms of the maximum number of end devices that a fog node could communicate within a given time. Experimental results showed that a RabbitMQ protocol with a DBT method connected to up to 900 devices and outperformed other options. Communication overhead in the fog layer was also assessed. The proposed data merging algorithm reduced traffic volume by 27.22–97.63%, with varying numbers of end devices and data points.

As a variety of IoT devices are going to be deployed in our surroundings, the future will also see many fog nodes. A group of nodes can communicate with each other to provide computing services for IoT applications. This work focused on how many fog nodes can be used together and how a fog node can offer various services efficiently. The prototype and testbed successfully demonstrated the feasibility of such collaboration and service provision within a hierarchical architecture. Experimental

results showed that a fog computing-based BEMS is able to operate with many IoT devices and to achieve scalable operations with minimum overhead. We hope that Fog BEMS can be used as a reference model in the research community.

There is potential for further advancement. For instance, a fog node can run a variety of applications. In edge analytics, artificial intelligence algorithms on a node can analyze energy data, make decisions, and execute control actions accordingly. As fog nodes are close to IoT devices, they can easily detect suspicious behavior in compromised devices, which critically improve energy security. A collaboration model among fog nodes is another topic that remains to be studied in future research.

Author Contributions: E.-K.L. contributed to conceptualization, project administration, investigation, and validation; U.N. contributed to the software, data curation, and visualization. All authors contributed to the writing. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported partially by Incheon National University Research Grant in 2019 and by a National Research Foundation of Korea (NRF) grant funded by the Korea government (Ministry of Science and ICT) (No. 2019R1H1A1101198 and No. 2016R1C1B1016084).

Conflicts of Interest: The authors declare that there is no conflict of interest.

Abbreviations

The following abbreviations and acronyms are used in this manuscript:

BEMS	Building Energy Management System
MAS	Multi-Agent System
DR	Demand Response
SOA	Service-Oriented Architecture
API	Application Programming Interface
PoC	Proof of Concept
JADE	Java Agent Development Environment
RPC	Remote Procedure Call
VIP	Volttron Interconnect Protocol
AMQP	Advanced Messaging Queueing Protocol
SMP	System Marginal Price
KPX	Korea Power eXchange
ToU	Time of Use
KMA	Korea Meteorological Administration
PBT	Point-Based Transmission
DBT	Device-Based Transmission
TPD	Transmission Time per Data
ZAP	ZeroMQ Authentication Protocol
SSL	Secure Socket Layer
CA	Certificate Authority
VEN	Virtual End Node
OpenADR	Open Automated Demand Response

References

1. Scenarios for the IoT Marketplace. Gartner, Inc. 2019. Available online: https://www.gartner.com/en/newsroom/press-releases/2019-08-29-gartner-says-5-8-billion-enterprise-and-automotive-io (accessed on 18 March 2020).
2. Vaquero, L.M.; Rodero-Merino, L.; Caceres, J.; Lindner, M. A Break in the Clouds: Towards a Cloud Definition. *ACM Sigcomm Comput. Commun. Rev.* **2009**, *39*, 50–55. [CrossRef]
3. Zhang, Q.; Cheng, L.; Boutaba, R. Cloud Computing: State-of-the-Art and Research Challenges. *J. Internet Serv. Appl.* **2010**, *1*, 7–18. [CrossRef]
4. Gutierrez-Garcia, J.; Ramos-Corchado, F.; Koning, J. An obligation-based framework for web service composition via agent conversations. *Int. J. Web Intell. Agent Syst.* **2012**, *10*, 135–150. [CrossRef]

5. Kendrick, P.; Baker, T.; Maamar, Z.; Hussain, A.; Buyya, R.; Al-Jumeily, D. An Efficient Multi-Cloud Service Composition Using A Distributed Multiagent-based, Memory-driven Approach. *IEEE Trans. Sustain. Comput.* **2018**. [CrossRef]
6. Sim, K.M. Grid Resource Negotiation: Survey and New Directions. *IEEE Trans. Syst. Man Cybern.* **2010**, *40*, 245–257.
7. Luong, N.C.; Hoang, D.T.; Wang, P.; Niyato, D.; Kim, D.I. Data Collection and Wireless Communication in Internet of Things (IoT) Using Economic Analysis and Pricing Models: A Survey. *IEEE Commun. Surv. Tutorials* **2016**, *18*, 2546–2590. [CrossRef]
8. Li, H.; Dong, M.; Ota, K. Data Collection and Wireless Communication in Internet of Things (IoT) Using Economic Analysis and Pricing Models: A Survey. *IEEE Commun. Surv. Tutorials* **2015**, *2*, 42–50.
9. Bera, S.; Misra, S.; Rodrigues, J. Cloud Computing Applications for Smart Grid: A Survey. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 1477–1494. [CrossRef]
10. Agarwal, S.; Yadav, S.; Yadav, A.K. An Efficient Architecture and Algorithm for Resource Provisioningin Fog Computing. *Int. J. Inf. Eng. Electron. Bus.* **2016**, *1*, 48–61.
11. Lee, E.K.; Shi, W.; Gadh, R.; Kim, W. Design and Implementation of a Microgrid Energy Management System. *Sustainability* **2016**, *8*, 1143. [CrossRef]
12. Simmhan, Y.; Aman, S.; Kumbhare, A.; Liu, R.; Stevens, S.; Zhou, Q.; Prasanna, V. Cloud-Based Software Platform for Big Data Analytics in Smart Grids . *IEEE Comput. Sci. Eng.* **2013**, *14*, 38–47. [CrossRef]
13. Ji, L.; Lifang, W.; Li, Y. Cloud Service based Intelligent Power Monitoring and Early-Warning System. In Proceedings of the IEEE PES Innovative Smart Grid Technologies, Tianjin, China, 21–24 May 2012.
14. Lee, E.K. Advancing Building Energy Management System to Enable Smart Grid Interoperation. *Int. J. Distrib. Sens. Netw.* **2016**, *8*. [CrossRef]
15. Lv, H.; Wang, F.; Yan, A.; Cheng, Y. Design of Cloud Data Warehouse and its Application in Smart Grid. In Proceedings of the IEEE International Conference on Automatic Control and Artificial Intelligence, Xiamen, China, 3–5 March 2012.
16. Fang, X.; Yang, D.; Xue, G. Evolving Smart Grid Information Management Cloudward: A Cloud Optimization Perspective. *IEEE Trans. Smart Grid* **2013**, *4*, 111–119. [CrossRef]
17. Rusitschka, S.; Eger, K.; Gerdes, C. Smart Grid Data Cloud: A Model for Utilizing Cloud Computing in the Smart Grid Domain. In Proceedings of the IEEE International Conference on Smart Grid Communications, Gaithersburg, MD, USA, 4–6 October 2010.
18. Roman, R.; Lopez, J.; Mambo, M. Mobile Edge Computing, Fog et al.: A Survey and Analysis of Security Threats and Challenges. *Future Gener. Comput. Syst.* **2018**, *78*, 680—698. [CrossRef]
19. Jiao, L.; Friedman, R.; Fu, X.; Secci, S.; Smoreda, Z.; Tschofenig, H. Cloud-based Computation Offloading for Mobile Devices: State of the Art, Challenges and Opportunities. In Proceedings of the Future Network and Mobile Summit, Lisboa, Portugal, 3–5 July 2013.
20. Bonomi, F.; Milito, R.; Zhu, J.; Addepalli, S. Fog Computing and Its Role in the Internet of Things. In Proceedings of the ACM Workshop on Mobile Cloud Computing, Helsinki, Finland, 17 August 2012.
21. Ni, J.; Zhang, K.; Lin, X.; Shen, X.S. Securing Fog Computing for Internet of Things Applications: Challenges and Solutions. *IEEE Commun. Surv. Tutor.* **2018**, *20*, 601–628. [CrossRef]
22. Dastjerdi, A.V.; Gupta, H.; Calheiros, R.N.; Ghosh, S.K.; Buyya, R. Fog Computing: Principles, Architectures, and Applications. *Comput. Res. Repos.* **2016**, abs/1601.02752.
23. Zhanikeev, M. A Cloud Visitation Platform to Facilitate Cloud Federation and Fog Computing. *IEEE Comput.* **2015**, *48*, 80–83. [CrossRef]
24. Kapsalis, A.; Kasnesis, P.; Venieris, I.S.; Kaklamani, D.I.; Patrikakis, C.Z. A Cooperative Fog Approach for Effective Workload Balancing. *IEEE Cloud Comput.* **2017**, *4*, 36–45. [CrossRef]
25. Krishnan, Y.; Bhagwat, C.N.; Utpat, A.P. Fog Computing - Network Based Cloud Computing. In Proceedings of the IEEE International Conference on Electronics and Communication Systems, Coimbatore, India, 26–27 February 2015.
26. Hong, K.; Lillethun, D.; Ramachandran, U.; Ottenwälder, B.; Koldehofe, B. Mobile Fog: A Programming Model for Large–Scale Applications on the Internet of Things. In Proceedings of the ACM Sigcomm Workshop Mobile Cloud Computing, Hong Kong, China, 12–16 August 2013.

27. Abedin, S.F.; Alam, M.G.R.; Tran, N.H.; Hong, C.S. A Fog based System Model for Cooperative IoTNode Pairing using Matching Theory. In Proceedings of the IEEE Asia-Pacific Network Operations and Management Symposium, Busan, Korea, 19–21 August 2015.
28. Aazam, M.; Huh, E.N. Dynamic Resource Provisioning Through Fog Micro Datacenter. In Proceedings of the IEEE International Conference on Pervasive Computing and Communication Workshops, St. Louis, MO, USA, 23–27 March 2015.
29. Hassan, M.A.; Xiao, M.; Wei, Q.; Chen, S. Help Your Mobile Applications with Fog Computing. In Proceedings of the IEEE International Conference on Sensing, Communication, and Networking Workshops, Seattle, WA, USA, 22–25 June 2015.
30. Ye, D.; Wu, M.; Tang, S.; Yu, R. Scalable Fog Computing with Service Offloading inBus Networks. In Proceedings of the IEEE International Conference on Cyber Security and Cloud Computing, Beijing, China, 25–27 June 2016.
31. Monteiro, A.; Dubey, H.; Mahler, L.; Yang, Q.; Mankodiya, K. Fit: A Fog Computing Device for Speech Tele-Treatments. In Proceedings of the IEEE International Conference on Smart Computing, St. Louis, MO, USA, 18–20 May 2016.
32. Gerla, M.; Lee, E.K.; Pau, G.; Lee, U. Internet of Vehicles: From Intelligent Grid to Autonomous Cars and Vehicular Clouds. In Proceedings of the IEEE World Forum on Internet of Things, Seoul, Korea, 6–8 March 2014.
33. Li, J.; Jin, J.; Yuan, D.; Palaniswami, M.; Moessner, K. EHOPES: Data-centered Fog Platform for Smart Living. In Proceedings of the IEEE International Telecommunication Networks and Applications Conference, Sydney, Australia, 18–20 November 2015.
34. He, Q.; Zhang, C.; Ma, X.; Liu, J. Fog-Based Transcoding for Crowdsourced Video Livecast. *IEEE Commun. Mag.* **2017**, *55*, 28–33. [CrossRef]
35. Tang, M.; Gao, L.; Pang, H.; Huang, J.; Sun, L. Optimizations and Economics of Crowdsourced Mobile Streaming. *IEEE Commun. Mag.* **2017**, *55*, 21–27. [CrossRef]
36. 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: Berlin/Heidelberg, Germany, 2016; pp. 169–186.
37. Dastjerdi, A.V.; Buyya, R. Fog Computing: Helping the Internet of Things Realize Its Potential. *IEEE Comput.* **2016**, *49*, 112–116. [CrossRef]
38. Jalali, F.; Vishwanath, A.; de Hoog, J.; Suits, F. Interconnecting Fog Computing and Microgrids for Greening IoT. In Proceedings of the IEEE Innovative Smart Grid Technologies—Asia, Melbourne, Australia, 28 November–1 December 2016.
39. Naranjo, P.G.V.; Shojafar, M.; Vaca-Cardenas, L.; Canali, C.; Lancellotti, R.; Baccarelli, E. Big Data Over SmartGrid—A Fog Computing Perspective. In Proceedings of the SoftCOM Workshop, Split, Croatia, 22–24 September 2016.
40. Li, G.; Wu, J.; Li, J.; Guan, Z.; Guo, L. Fog Computing-Enabled Secure Demand Response for Internet of Energy Against Collusion Attacks Using Consensus and ACE. *IEEE Access* **2018**, *6*, 11278–11288. [CrossRef]
41. Khalid, A.; Aslam, S.; Aurangzeb, K.; Haider, S.I.; Ashraf, M.; Javaid, N. An Efficient Energy Management Approach Using Fog-as-a-Service for Sharing Economy in a Smart Grid. *Energies* **2018**, *11*, 3500. [CrossRef]
42. Javed, A.; Rana, O.; Marmaras, C.; Cipcigan, L. Fog Paradigm for Local Energy Management Systems. *Lect. Notes Inst. Comput. Sci.* **2017**, *189*.
43. Zahoor, S.; Javaid, S.; Javaid, N.; Ashraf, M.; Ishmanov, F.; Afzal, M.K. Cloud–Fog–Based Smart Grid Model for Efficient Resource Management. *Sustainability* **2018**, *10*, 2079. [CrossRef]
44. Wang, P.; Liu, S.; Ye, F.; Chen, X. A Fog-based Architecture and Programming Model for IoT Applications in the Smart Grid. *CoRR* **2018**, abs/1804.01239.
45. Yue, J.; Hu, Z.; He, R.; Zhang, X.; Dulout, J.; Li, C.; Guerrero, J. Cloud-Fog Architecture Based Energy Management and Decision-Making for Next-Generation Distribution Network with Prosumers and Internet of Things Devices. *Appl. Sci.* **2019**, *9*, 372. [CrossRef]
46. Yan, Y.; Su, W. A Fog Computing Solution for Advanced Metering Infrastructure. In Proceedings of the IEEE/PES Transmission and Distribution Conference and Exposition (T&D), Dallas, TX, USA, 3–5 May 2016.
47. Moghaddam, M.H.Y.; Leon-Garcia, A. A Fog-Based Internet of Energy Architecture for Transactive Energy Management Systems. *IEEE Internet Things J.* **2018**, *5*, 1055–1069. [CrossRef]

48. Faruque, M.A.A.; Vatanparvar, K. Energy Management-as-a-Service Over Fog Computing Platform. *IEEE Internet Things J.* **2016**, *3*, 161–169. [CrossRef]
49. Dorri, A.; Kanhere, S.; Jurdak, R. Multi-Agent Systems: A Survey. *IEEE Access* **2018**, *6*, 28573–28593. [CrossRef]
50. Kravari, K.; Bassiliades, N. A Survey of Agent Platforms. *J. Artif. Soc. Soc. Simul.* **2015**, *18*, 1–18. [CrossRef]
51. Bellifemine, F.; Poggi, A.; Poggi, A. JADE: A FIPA2000 compliant agent development environment. In Proceedings of the ACM International Conference on Autonomous Agents, Montreal, QC, Canada, 28 May–1 June 2001; pp. 216–217.
52. Haack, J.; Akyol, B.; Tenney, N.; Carpenter, B.; Pratt, R.; Carroll, T. VOLTTRON: An Agent Platform for Integrating Electric Vehicles and Smart Grid. In Proceedings of the Int'l Conference on Connected Vehicles and Expo, Las Vegas, NV, USA, 2–6 December 2013.
53. VOLTTRON Documentation. Available online: https://volttron.readthedocs.io/en/develop/ (accessed on 18 March 2020).
54. Katipamula, S.; Haack, J.; Akyol, B.; Hernandez, G.; Hagerman, J. VOLTTRON: An Open-Source Software Platform of the Future. *IEEE Electrif. Mag.* **2016**, *4*, 15–22. [CrossRef]
55. VOLTTRON Brochure. Available online: https://volttron.org/sites/default/files/publications/VOLTTRON_Brochure_V11_WEB.pdf (accessed on 18 March 2020).
56. ZeroMQ. Available online: http://zeromq.org/ (accessed on 18 March 2020).
57. Bhatia, L. Message Queues in Industrial IoT. Available online: https://wiki.aalto.fi/download/attachments/116662239/Message%20Queues%20in%20Industrial%20IoT.pdf?version=1&modificationDate=1481638941661&api=v2 (accessed on 18 March 2020).
58. AMQP. Available online: https://www.rabbitmq.com/ (accessed on 18 March 2020).
59. Korea Power Exchange (KPX). Available online: https://www.kpx.or.kr/ (accessed on 18 March 2020).
60. Korea Meteorological Administration (KMA). Available online: http://www.kma.go.kr/ (accessed on 18 March 2020).
61. Happ, D.; Karowski, N.; Menzel, T.; Handziski, V.; Wolisz, A. Meeting IoT Platform Requirements with Open Pub/Sub Solutions. *Ann. Telecommun.* **2017**, *72*, 41–52. [CrossRef]
62. ZeroMQ Authentication Protocol. Available online: https://rfc.zeromq.org/spec:27/ZAP/ (accessed on 18 March 2020).
63. CurveZMQ. Available online: https://rfc.zeromq.org/spec:26/CURVEZMQ/ (accessed on 18 March 2020).
64. Blaauwbroek, N.; Nguyen, P.; Konsman, M.; Shi, H.; Kamphuis, R.; Kling, W. Decentralized Resource Allocation and Load Scheduling for Multicommodity Smart Energy Systems. *IEEE Trans. Sustain. Energy* **2015**, *6*, 1506–1514. [CrossRef]
65. Brusco, G.; Burgio, A.; Menniti, D.; Pinnarelli, A.; Sorrentino, N. Energy Management System for an Energy District With Demand Response Availability. *IEEE Trans. Smart Grid* **2014**, *5*, 2385–2393. [CrossRef]
66. Calvillo, C.; Sánchez-Miralles, A.; Villar, J. Energy management and planning in smart cities. *Renew. Sustain. Energy Rev.* **2016**, *55*, 273–287. [CrossRef]
67. Carli, R.; Dotoli, M.; Pellegrino, R.; Ranieri, L. Using multi-objective optimization for the integrated energy efficiency improvement of a smart city public buildings' portfolio. In Proceedings of the IEEE International Conference on Automation Science and Engineering, Gothenburg, Sweden, 24–28 August 2015.
68. Hosseini, S.M.; Carli, R.; Dotoli, M. A Residential Demand-Side Management Strategy under Nonlinear Pricing Based on Robust Model Predictive Control. In Proceedings of the IEEE International Conference on Systems, Man and Cybernetics, Bari, Italy, 6–9 October 2019.
69. Kang, S.J.; Park, J.; Oh, K.Y.; Noh, J.G.; Park, H. Scheduling-based real time energy flow control strategy for building energy management system Author links open overlay panel. *Energy Build.* **2014**, *75*, 239–248. [CrossRef]