IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received October 23, 2020, accepted November 2, 2020, date of publication November 19, 2020, date of current version December 8, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3039368

Scalable IoT Platform for Heterogeneous Devices in Smart Environments

ASAD JAVED[1], (Member, IEEE), AVLEEN MALHI[1], TUOMAS KINNUNEN[1], AND KARY FRÄMLING[1,2]

[1]Department of Computer Science, Aalto University, 02150 Espoo, Finland
[2]Department of Computer Science, Umeå University, 901 87 Umeå, Sweden

Corresponding author: Avleen Malhi (avleen.malhi@aalto.fi)

This work was supported in part by the European Union's (EU's) Horizon 2020 Research and Innovation Program under Grant bIoTope: 688203, in part by the Horizon 2020 Project FINEST TWINS under Grant 856602, and in part by the Academy of Finland (Open Messaging Interface) under Grant 296096.

ABSTRACT The Internet of Things (IoT) is envisioned as a ubiquitous computing infrastructure in which everything becomes connected, enabling gigantic information exchange among Things and people. These connected smart Things generate an enormous amount of data which need to be efficiently managed to form a unified global IoT. Unfortunately, due to the lack of acceptable open standards, communication protocols, and support for device/service discovery, the recent IoT deployments in smart environments (e.g., smart home, smart building, smart city) are posing imperative challenges related to interoperability, discovery, and the configuration of deployed objects, since the number of objects is expected to grow over time. Therefore, it is of utmost importance to provide open and scalable solutions for the discovery of devices (i.e., Things), their configuration, and data management. This paper introduces an open and scalable IoT platform by adopting the modular characteristics of edge computing for smart environments. This paper: (i) performs a systematic literature review of IoT-based infrastructures and analyzes the scalability requirements; (ii) proposes a layered IoT platform for smart environments that fosters heterogeneity, interoperability, discovery, and scalability; and (iii) demonstrates the applicability of the proposed solution by relying on a comprehensive study of a Väre smart building use case at Aalto University.

INDEX TERMS Internet of Things, hashing, edge computing, discovery, communication standards, scalability.

I. INTRODUCTION

The evolution of the Internet of Things (IoT) technology has triggered the currently marketed digital world in which everything becomes connected, leading to the distinct methods of ephemeral device communications on the platform of technologies such as GSM, WiFi, ZigBee, and Bluetooth [1]–[3]. Nonetheless, the objective is not only to facilitate device communications but also to be informed about non-communicable real-world objects in the surrounding environments. This paradigmatic shift unfastens additional services for the future as stated by Mark Weiser [4]: ''The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it''. This perception is the motivation for the current communication channels and miniaturized technologies. Hence, the future lies in hundreds of physical objects (i.e., Things) for each person effectively communicating with each other to assist the current smart way of living. Further, the IoT kindles the necessity of scalable solutions for constantly changing environments, such as smart home, smart city, or smart building, to efficiently integrate an enormous amount of data collected from heterogeneous IoT devices [5]. Such relevant heed is also exhibited by the emergence of fog/edge computing which contains decentralized edge nodes for supporting escalated scalability and low latency [6], [7]. This pushes for solutions based on edge computing that can perform data publication and consumption on the network edge, as cloud-based IoT deployments are unable to meet the increasing demands of clients, due to their complexity and cost. Typically, IoT-based platforms should be able to manage the volume, velocity, and variety of real-time data produced

The associate editor coordinating the review of this manuscript and approving it for publication was Liang-Bi Chen.

by heterogeneous IoT devices at the edge-level. In fact, they should be able to process millions of user and service requests for optimal systems performance.

Unfortunately, the recent IoT deployments in smart environments lead to four challenges due to the lack of acceptable open standards, edge computing-based data management, and no support for device/service discovery. (i) Scalability: The IoT applications should be able to scale up with respect to the increasing number of users and computing environments, thereby handling a large number of user requests which cover a larger virtual or physical space required for sensing, aggregation, and computation of data. (ii) Data management: The real-time data should be handled at the edge of the network to optimize data publication, data consumption, and network latency. (iii) Interoperability: There is no unified communication model for overcoming vertical silos (data being siloed in a unique system and staying there [8]), which poses interoperability and openness issues. (iv) Device discovery: In smart environments, mobile devices should be able to discover themselves to publish their data and/or services. It is also crucial to automatically discover IoT devices when they move from one location to another.

To overcome the above-mentioned challenges, this paper proposes an open and scalable IoT platform for heterogeneous devices in smart environments that provides heterogeneity, interoperability, discovery, and scalability. Overall, the paper offers three contributions: (i) It performs a comparative study of the existing state-of-the-art IoT frameworks based on different characteristics and formulates the scalability requirements. (ii) Based on these requirements, it introduces a scalable IoT platform by adopting a consistent hashing mechanism [9] for balancing real-time data closer to the data sources (i.e., at the edge). We also employ the Open Messaging Interface (O-MI) and Open Data Format (O-DF) standards, proposed by the Open Group consortium [10], [11], as a unified IoT edge node/server[1] for providing interoperability and data semantics. The proposed solution allows data publication and consumption on the edge, provides efficient load balancing of sensor data, and dynamically discovers IoT mobile devices. (iii) The paper demonstrates the applicability of the proposed solution in a Väre smart building use case at Aalto University campus.

This article is further organized into five sections. Section II details the state-of-the-art IoT architectures proposed in the current literature. Section III proposes an open and scalable IoT platform for smart environments which solves scalability, interoperability, data management, and discovery challenges. The practicability and replicability of the proposed platform have been showcased by applying it to the Väre smart building use case in Section IV. Section V illustrates the experimental results, accompanied by conclusions in Section VI.

[1]In this paper, node and server for O-MI are used interchangeably

II. BACKGROUND AND RELATED WORK

The advancements in the IoT paradigm have yielded several core concepts to provide more accurate insights into real-world IoT implementations in smart environments. The edge computing concept enables an intelligent computing infrastructure by placing computation and data storage closer to data sources [12]. In contrast, fog computing provides a decentralized computing platform in which the data, compute, storage, and network servers are located somewhere between the data source and the cloud [13]. This location for computation can either be on a network gateway or the edge. On the other hand, cloud computing facilitates the on-demand delivery of various services/data through the Internet, including data storage, servers, and databases [14]. FIGURE 1 lists the basic features for IoT applications at the edge, gateway, and cloud levels. In the following, we review existing state-of-the-art IoT frameworks that provide scalability, discovery, interoperability, and data management solutions at one or more of these levels.

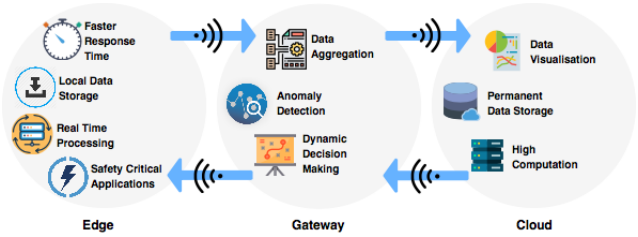


FIGURE 1. Basic features of edge, gateway, and cloud for IoT applications.

A. SCALABILITY

To foster the cities in providing smarter services for their citizens, many technological challenges are posed by IoT [15], cloud [16], and big data [17] in the IoT-based developments. A manageable, scalable, and transparent IoT architecture is proposed by Guo *et al.* [18] as a response to these challenges. The architecture is evaluated as an efficient solution in terms of energy consumption for resource management and on-demand service provisioning of IoT objects. Similarly, a cloud-based architecture for coordinated emergency management in smart cities is designed, which adopts cloud computing to manage data computation and storage resources [19]. An open-source microservices-based IoT platform called InterSCity is proposed for enabling cooperative development of extensive services, systems, and applications needed for future smart cities [20]. The authors also mention a smart city simulator for generating realistic workloads for a Smart Parking scenario in Brazil. In another study, a scalable modular framework applying machine learning is put forth to reduce latency and processing redundancy with minimal impact on accuracy [21]. Many context-aware IoT-based systems have also been examined for real-time decision making and scalability concerns in smart environments [22], indicating that scalability is affected by different architectural choices. Furthermore, a CEB IoT architecture for

cloud-sensor systems is proposed along with an event-based programming model for the development of IoT applications [23]. The experiments are conducted to demonstrate scalability features and capability of dynamic load adaption for cloud-sensor systems. Another IoT/M2M platform called oneM2M has been designed based on fog computing by organizing the middle nodes into highly scalable hierarchical container-based fog nodes [24]. In this solution, a mechanism for dynamically scaling the middle nodes is also designed to provide scalable solutions. For developing reliable systems, a fault tolerant and scalable approach is designed for health monitoring application to address the traffic bottleneck and sink node hardware malfunction caused by high-receiving data rates [25]. Another study analyzes the FIWARE performance under different platform configurations to compare cloud-only and edge/cloud cases for smart water management in agriculture [26]. The experimental results demonstrate that the system performance is not always improved by fog computing, making it worse in some cases. Similarly, another scalable and flexible IoT architecture based on network function virtualization is proposed for processing huge amounts of sensor data [27]. Other widely-adopted platforms including Microsoft Azure, Amazon AWS, and Google Cloud can also be referenced, which deliver extensively integrated cloud services along with IoT deployment, configuration, and scalability; however, these platforms are proprietary cloud systems and do not provide interoperable solutions. TABLE 1 lists some basic characteristics of these platforms. All three cloud systems follow pay-as-you-go pricing model, which depends on the instance type, volume usage, storage, and other factors. As an example, TABLE 1 only shows on-demand pricing based on the standard instance type. In contrast, our platform is free and open-source.

TABLE 1. Basic features of IoT middleware along with our proposed platform.

Features	Microsoft Azure	Amazon Web Services	Google Cloud	Proposed
Comm. protocols	HTTP, MQTT, AMQP over WebSockets	HTTP, MQTT, WebSockets	HTTP, MQTT	O-MI/O-DF over HTTP, WebSockets
IoT key offerings	Azure IoT Hub: • Connectivity • Authentication • Device monitoring • Device management	AWS IoT Core: • Connectivity • Authentication • Rules engine • Device gateway	Cloud IoT Core: • Connectivity • End-to-end security • Device management	Layered Design: • Connectivity • Device gateway • Interoperability • Load balancing
Edge computing solution	IoT Edge	IoT Greengrass	Edge TPU	With containers, hashing, and O-MI/O-DF
Discovery solution	DNS-SD, mDNS	Amazon Elastic Container Service (ECS)	Google Kubernetes Engine (GKE)	WiFi hotspot
On-Demand pricing	$0.166 (per hour)	$0.192 (per hour)	$0.214 (per hour)	Open source

B. DISCOVERY

Several frameworks for IoT discovery have been proposed in the literature, which enable IoT applications to access data without the need to know the data source, sensor description, or location [28]–[30]. In complex, heterogeneous, and constantly changing environments, such as smart home or smart building, it is crucial to devise efficient discovery schemes. In the existing literature, discovery mechanisms can be classified into two categories [31], [32]. (i) *Device discovery* allows a machine (or user) to retrieve a list of IoT devices available in the residing network. The most widely-adopted technologies for device discovery are UPnP (Universal Plug and Play) [33], mDNS (multicast DNS) [34], and DLNA (Digital Living Network Alliance) [35]. (ii) *Service discovery* allows a machine (or user) to retrieve and understand services provided by the previously identified IoT devices or systems. The most common service discovery protocols are UDDI (Universal Description, Discovery, and Integration) [30], OWL (Web Ontology Language) [36], WSMO (Web Service Modeling Ontology) [37], and SPARQL [38]. Some technologies cover both device and service discovery, such as AllJoyn [33] and DPWS (Devices Profile for Web Services) [39].

C. INTEROPERABILITY

To make Things accessible through a single universal communication protocol, it is crucial to ensure the interoperability of data models, which define the format, syntax, and semantics of given data, thereby resolving vertical silos. According to [42], interoperability covers six distinct levels: 1-Technical, 2-Syntactic, 3-Semantic, 4-Pragmatic, 5-Dynamic, and 6-Conceptual. In the literature, interoperability among different IoT systems mainly investigates: (i) *Syntactic* level, which includes data interchange and language-independent formats, such as O-DF [11], JSON(-LD), or RDF/XML; and (ii) the *Semantic* level, which addresses the ontology- and vocabulary-based approaches, such as SSN and schema.org. To this end, a multi-layered platform is proposed to overcome the scalability and interoperability issues for IoT applications [43]. Another interoperable and distributed IoT architecture is designed to address the heterogeneity of IoT devices and enable seamless inclusion of new devices in IoT applications [40].

D. BIG DATA MANAGEMENT

Since many devices are interconnected in IoT-based systems, they generate huge volumes of data, which need to be managed for efficient Big Data management. A NoSQL-based centralized data management system is proposed by Li *et al.* [44] for storing enormous amounts of IoT data. Another NoSQL-based distributed database called Apache Cassandra is used in the literature for managing large-scale IoT data which can take an unstructured, structured, or semi-structured form [45]. Similarly, another alternative is MongoDB,[2] which is a general-purpose document-based NoSQL database used to store huge volumes of data in JSON-like documents. These databases adopt a hashing-based system to balance the load across multiple servers. However, although both Cassandra and MongoDB facilitate frequent read and

[2][Online]. https://www.mongodb.com/, last accessed in Nov, 2020

TABLE 2. State-of-the-art comparative analysis between the existing frameworks and the proposed solution.

Ref.	IoT Application	Type of Scalability	Level of Scalability	Technologies	Communication Protocol	Discovery	Virtualization
[27]	OPerating room Innovation Center	Horizontal	Cloud	Bluetooth, ZigBee	HTTP	-	Hypervisor-based
[40]	Smart home, smart transportation	Horizontal	Cloud	Composite virtual objects	-	Device	Hypervisor-based
[24]	Industrial IoT	Horizontal	Gateway (Fog)	Docker Containers	HTTP	Service	Container-based (Docker)
[20]	Smart city	Horizontal	Cloud	GCloud, Kubernetes, RabbitMQ	HTTP, AMQP	Service	Container-based (Docker)
[25]	Health monitoring	Horizontal	Gateway (Fog)	SoC, Contiki OS	6LoWPAN	-	-
[26]	Smart water management for Agriculture	Increasing Workload (Load scaling)	Cloud and Gateway (Fog)	Docker, SenSE tool, MongoDB, Mosquitto	MQTT	-	Container-based (Docker)
[22]	Smart city (Padua, Italy)	Horizontal	-	SenSE tool	MQTT	-	Hypervisor-based
[21]	Residential smart grid	Horizontal	Edge	-	-	-	-
[23]	City-wide smart home	Horizontal and Load scaling	Cloud and Edge	E-SODA programming	HTTP	Service	Hypervisor-based
[41]	Smart city	Horizontal	Edge	WiFi	-	-	-
Proposed	Building automation	Horizontal and Vertical	Edge and Gateway (Fog)	Docker, Hashing, SQLite	O-MI/O-DF	Device and Service	Container-based (Docker)

write operations with a key-value storage, it features no data subscription functionality for automatic data/services updates in contrast to other systems, such as O-MI and O-DF standards. Furthermore, another distributed mechanism for load-balancing in cloud data centers is proposed by Tian *et al.* [46], allocating physical and virtual servers by considering memory, CPU usage, and network bandwidth. Similarly, Singh *et al.* [47] propose another load-balancing algorithm for agile data centers, which can manage the hierarchical data distribution to multiple servers. However, the functionality of distributing subscription-based data remains underexamined in this approach. To balance the load in mobile edge computing, Yu *et al.* [48] implement a scalable and dynamic software load balancer based on minimal perfect hashing. Although this solution improves the network performance, there is no support for publish/subscribe messaging, which is the core building block of our proposed IoT platform.

E. COMPARATIVE ANALYSIS

Based on the above-mentioned challenges and discussions, we review state-of-the-art technologies, protocols, and standards for scalable IoT-based applications. This comparative analysis is elaborated in TABLE 2. The proposed methodology based on various characteristics is also compared with the existing IoT-based platforms. As seen in TABLE 2, the *IoT application* lists the use cases on which any particular technique has been tested or validated. The *scalability type* can be classified into either horizontal (i.e., if IoT devices are increased), vertical (i.e., if the number of resources is increased), or Load scaling (increasing the workload). The *level of scalability* refers to the system on which any particular method has been applied, and the levels in our case are cloud, edge, and gateway (fog). Further, another criterion is the *communication protocol* adopted by the platform and support for *IoT discovery*, such as device and service discovery. Finally, the *virtualization* criteria verifies whether the existing solutions support hypervisor- or container-based virtualization.

III. THE SCALABLE IoT PLATFORM

Most of the related works in the literature only present cloud-based architectures for scalability and focus mainly on the design and implementation at the cloud-level. Also, they seldom analyze these approaches on real-life use cases. Therefore, based on the above-mentioned comparisons, we propose an open and scalable IoT platform for smart environments by adopting the modular characteristics of edge computing. The proposed framework balances the load by relying on consistent hashing, combined with a distributed storage mechanism, thus providing discovery and scalability at the edge-level (i.e., where data are published and consumed). In addition, this layered design supports interoperability using open messaging standards, which are described in Section III-D. TABLE 3 lists the scalability requirements of our proposed solution, which are derived from earlier related work in various domains [49]–[51]. FIGURE 2 shows the high-level overview of our proposed solution, which consists of three main Planes: Discovery, Storage, and Service.

A. DISCOVERY PLANE

The Discovery plane provides device discovery and data translation features, which allow IoT devices and/or smart

TABLE 3. Scalability requirements for IoT applications.

• Able to **persist data locally** on the edge in the case of network faults or a node malfunctions.
• Ability to **discover mobile IoT devices** when they move from one network to another.
• Capable of **minimizing latency peaks** and **additional delays** to ease data publication and consumption.
• Able to **scale up** the number of servers for managing extensive data computation.
• Capable of handling **data semantics** for interoperable operations between various systems.
• A server crash or network fault **should not** corrupt the data.

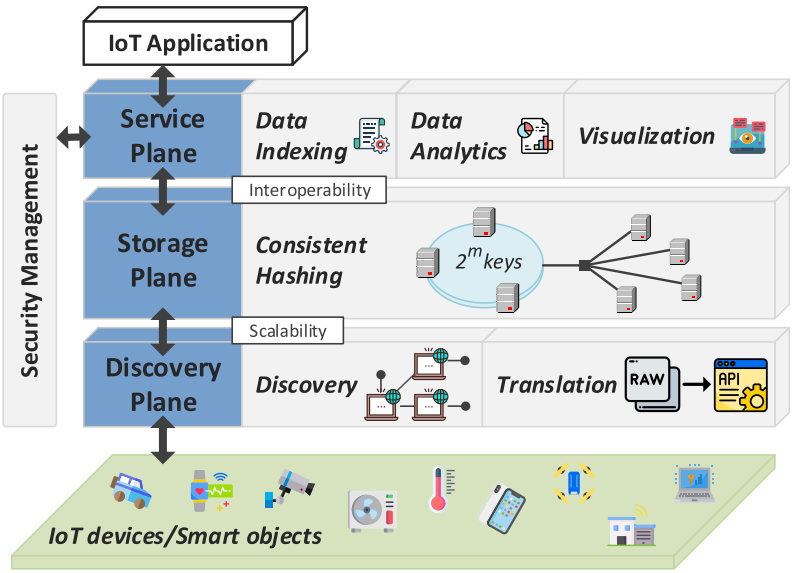


FIGURE 2. Scalable IoT platform containing three planes.

objects to communicate openly with the entire system without any vendor lock-in obstacles.

1) DEVICE/SERVICE DISCOVERY

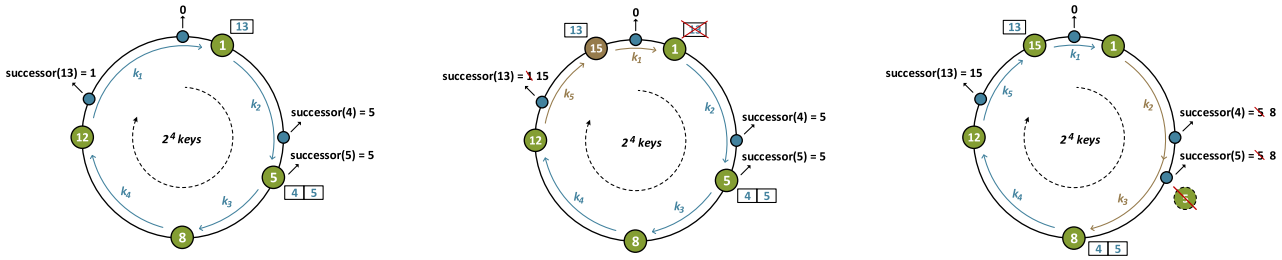
In IoT, discovery mechanisms refer to the naming and locating schemes for identifying available devices in the network, the services they offer, and ways to connect with them [28], [32]. In large, heterogeneous, and constantly changing IoT deployments, it is important for the devices to be discoverable on the site to publish/consume their data, otherwise the installation and maintenance costs could be too high. We implement a device discovery mechanism to handle the WiFi connection configuration of devices in a WiFi-based IoT network as the WiFi operates at a much faster rate. The mechanism enables the IoT network to share configuration to new devices automatically instead of requiring the installer personnel to run manual configuration steps. As compared to a WiFi-based network, we can also adopt Bluetooth technology as an alternative to provide wireless connectivity between devices. This technology is available at low cost in many hardware.

2) DATA TRANSLATION

Once the devices have been discovered, the published real-time data need to be processed before converting them into a standardized API format. The standardization of data is imperative as the data can be made clear and persistent through this method, which ensures the relative understanding, identification, and management of related data by employing the common format and terminology [52]. Our proposed work (i) acquires the data items in raw format from different data sources, for instance, XML and JSON; (ii) converts the data into standardized format (e.g., RDF and O-DF); and (iii) stores the translated data using distributed storage mechanism, as explained in Section III-B.

B. STORAGE PLANE

The Storage plane is responsible for distributing the load to multiple servers. We adopt a consistent hashing mechanism to evenly distribute data items among diverse servers; the proposed design is analogous to Apache Cassandra. Nevertheless, our design offers the publish/subscribe messaging which is understudied in Cassandra. In consistent hashing, each server is allocated for the administration of multiple distinct key ranges. As an example, the optimal load-balancing behavior can be ensured by assigning an adequate number (15+) of key ranges per server [9]. The consistent hashing is adapted to attain higher accuracy in contrast to the other load balancing mechanisms (e.g., Round Robin [53]) as it has a comparatively low number of data changes in the case of servers joining or leaving the system.



(a) Mapping of four servers on a circle (b) Mapping of another server 15 joining the circle (c) When server 5 leaves the circle

FIGURE 3. An example of identifier circle with $m = 4$; servers 1, 5, 8, 12, 15, respectively, manage key ranges $k_1 \ldots k_5$.

Consistent hashing can be conceptualized as an *identifier circle* which supplies a key space for mapping keys (e.g. server or data) in a clockwise fashion. The same key space is used to map both the server and data IDs. Generally, a hash function (e.g. SHA-2 [54]) is calculated for assigning an *m-bit* key identifier to each individual ID. To ensure that no two IDs have similar key identifiers, the length of m should be sufficiently large. The hashing operation works as follows. First, the keys are ordered in an *identifier circle* with the help of modulo 2^m. Then, the key k is assigned in a clockwise fashion to the first server with an identifier equal to key k. In the process, the next server that follows the previous key k is used for data mapping if the server with key k is unavailable. An exemplary hashing circle in which the key space has 16 keys ($m = 4$) from 0 to 15 can be seen in FIGURE 3. FIGURE 3(a) shows the mapping of four servers to keys 1, 5, 8, 12 in which each server is in charge of at least one key range (from ranges k_1, k_2, k_3, k_4). The blue circles correspond to the data keys. During the mapping process, the successor of key 4 is 5 as this key belongs to the k_2 range, which is handled by the server 5. A similar case is followed for keys 13 and 0 as the successor for both keys is server 1. From the implementation viewpoint, these data and their identifiers are stored as key-value pairs in a Distributed Hash Table (DHT), providing lookup and storage schemes similar to a hash table.

FIGURE 3(b) depicts a scenario in which a new server is added to the system and mapped to key 15. As a result, the key range k_1 is sliced into a new key range k_5 (shown in brown color) and assigned to server 15. Accordingly, the data location with key 13 is shifted to the new server. Similarly, FIGURE 3(c) depicts the behavior of removing the server, which causes the server with key 8 to manage both k_2 and k_3 key ranges. For both addition and removal cases, the data need to be migrated to a new server, thereby showing that only a certain portion of key range is re-assigned without affecting the entire key space.

C. SERVICE PLANE

The Service plane of our proposed platform is responsible for creating and managing different services that are accessible to end-users or IoT applications. It provides three functions. First, data indexing allows efficient retrieval of data selected by the users, thus optimizing the performance of the system. Deep learning is the most widely-adopted concept for data tagging and indexing, commonly adopted to identify patterns and extract features from complex unsupervised data without any human interaction [55]. Second, data analytics allows analyzing large-scale data sets with varying data properties, thereby extracting substantial conclusions and actionable insights. Third, data visualization allows presenting raw data in a meaningful manner that is derived from different data streams. Some IoT data visualization tools and techniques can be referenced, such as Grafana, Google charts, and Databox [56].

D. UNIFIED MESSAGING STANDARDS

To ensure interoperability in the proposed IoT platform, O-MI is adopted as the application-level messaging protocol which supports both client/server and publish/subscribe communication models. O-MI supports the CRUD (Create, Read, Update, Delete) model, which is the key in any given IoT application and provides six communication interfaces (operations), as listed in TABLE 4. One of the core properties of O-MI is that it is *protocol agnostic*, meaning that it can be implemented over different protocols, such as HTTP, WebSocket, SOAP, or TCP-IP [57], [58]. An open-source reference implementation for the O-MI standard has been developed,[3] which contains three main components: (i) the *API endpoint* handles user requests and currently supports both HTTP and WebSocket protocols; (ii) the *Agent system* contains multiple software agents, which are used to pull/push data (e.g., sensor data values) from and to the internal database; and (iii) the *User/Web interface* is used for executing O-MI operations, listed in Table 4. Furthermore, FIGURE 4 shows an example of the O-MI read message, which applies the read operation (see line 2) for requesting the temperature data. The requested services use a standard complementary to O-MI named O-DF (Open Data Format) [11], as highlighted in FIGURE 4. Just like the HTTP protocol that can embed formats other than HTML, O-MI is designed to be independent of O-DF and can embed any other payload formats.

The O-DF standard is defined using the XML schema to represent any IoT object. It is intentionally presented

[3][Online]. O-MI implementation available: https://github.com/AaltoAsia/O-MI, last accessed in Oct, 2020

TABLE 4. O-MI Communication Interfaces (supported operations).

Operation	Description
1- Write	For storing information updates coming from events, sensors, or other devices into O-MI nodes.
2- One-Time Read	For retrieving historical or recent information from O-MI nodes.
3- Read (Subscription)	A specialized read operation to retrieve information periodically. The subscription can be of two types: • *Subscription with callback address:* The callback address is used for receiving the requested data with a clearly identified interval (event- and interval-based are implemented). • *Subscription without callback address:* The subscribed node is used to store the data until the valid subscription. Then, a new read request is issued to retrieve the data.
4- Method Call	For calling functional parts of O-DF structures with given parameters.
5- Cancel	The cancellation of subscription before expiration.
6- Delete	For deleting some components of O-DF hierarchy.

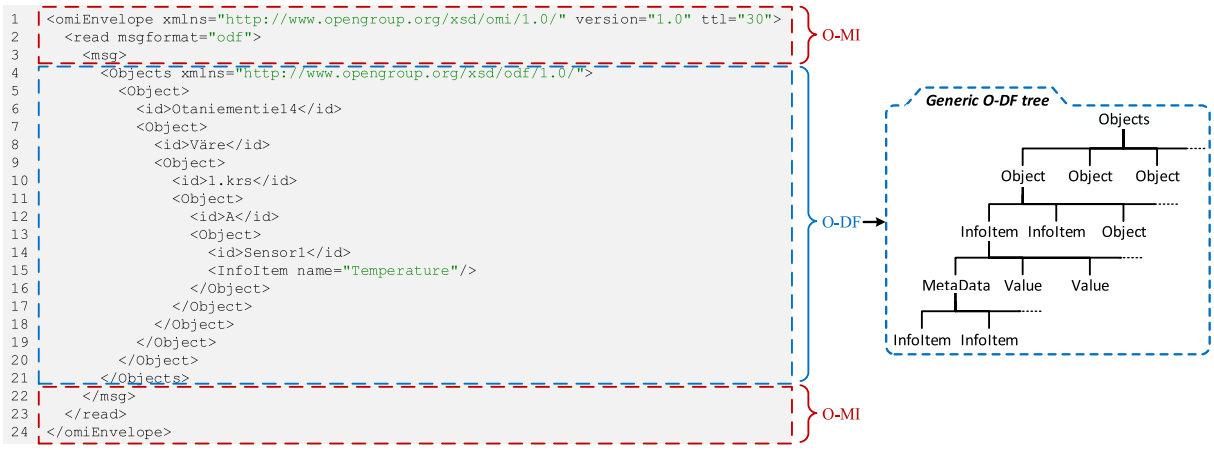


FIGURE 4. O-MI/O-DF read message requesting for temperature data.

similarly to data structures in object-oriented programming. The hierarchical tree structure is organized as follows (refer to FIGURE 4): it starts with its top element as `Objects` element which can contain `Object` sub-elements of any number. In a similar vein, any number of properties (referred to as `InfoItems`) can be contained within the `Object` elements as well as any number of `Object` sub-elements. Therefore, the resultant `Object` tree can consist of a number of levels. There is a mandatory sub-element, called `id` by every `Object` used for identification of the `Object`. It is ideally unique for a specific application or a residing network. From a semantic viewpoint, O-DF can be extended with domain-independent and/or domain-dependent semantic vocabularies, such as SSN and MobiVoc [59]. In the message given in FIGURE 4, O-DF can be enriched with schema.org tags in the Objects level (using ''type'' parameters).

The security management component of our proposed platform is designed by integrating it with the O-MI security module developed by Aalto University,[4] which provides access control (i.e., authorization) and authentication features. This security module consists of two components:

- *Authentication:* enables new user registrations and manages user sessions. It currently supports three methods: (i) password-based authentication; (ii) OAuth 2.0-based registration using any service provider; and (iii) LDAP-based authentication.
- *Authorization:* grants control privileges to the O-MI users for managing the data/service items. It further provides two functionalities: (i) access control to verify that the user is allowed to process and access the requested data items; and (ii) administrator console for O-MI node owners to manage access policies.

[4][Online]. O-MI Authentication and Authorization modules: https://github.com/AaltoAsia, last accessed in June, 2020

IV. REAL-LIFE USE CASE: VÄRE SMART BUILDING

The proof of concept is accomplished for the Väre building in the Aalto University campus with the entire building comprising 24 blocks on three floors with a total property area of 34,000m^2. The interactive system views of the dimensional representation are depicted in FIGURE 5 including two main building components: the building floor plan and 3D model. These interactive views help in understanding and interpreting the visual information in the building spaces. The Väre building data are collected from multiple room facilities including heating and cooling valve, presence and temperature sensors, together with sensor data assembled from five electrical systems installed throughout the building and data dashboard to visualize the collected data. Different technologies, such as WiFi, O-MI reference implementation, and sensors enable connections in the Väre building for intelligently

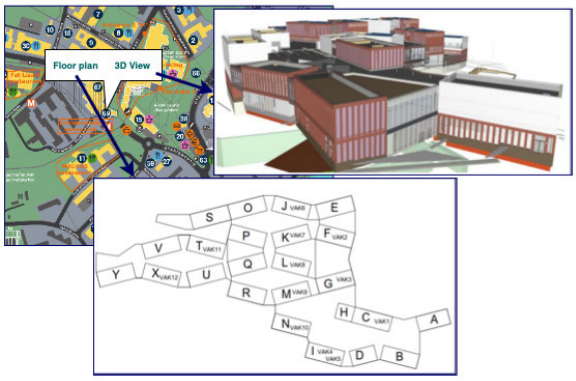


FIGURE 5. The floor plan and 3D view of Väre smart building at Aalto University campus.

managing and storing the abundant data produced by the sensors for smart observations. Data exchange is performed by the data messaging interface and data format standards (i.e. O-MI and O-DF) for seamlessly integrating heterogeneous sensors.

A. IMPLEMENTATION DETAILS

The discovery mechanism is depicted in FIGURE 6 along with our proposed algorithm in Algorithm 1.[5] The first installed device checks for a hidden WiFi access point. If it is not found, the device needs manual configuration via a hot spot access point with WiFi credentials for the device to connect to. After configuration, the device connects to the given WiFi but also configures a hidden WiFi access point known only to devices intended for this IoT system. The next devices can then find the hidden WiFi if installed nearby and they will use standard protocols, such as O-MI/O-DF, to retrieve the configuration. Therefore, they can automatically connect to the correct WiFi network and access the rest of the IoT system. It should be noted that the hidden WiFi should not be accessible for outsiders, otherwise they could find the WiFi password. The WiFi credentials need to be kept secret and encrypted for security against physical attacks. For encryption, the security features of the processor should be used in the IoT device. For example, the ESP32 micro-controller has a flash encryption and secure boot feature, which immensely

[5]Implementation available [Online]. https://github.com/AaltoAsia/wifiDiscovery, last accessed in Sep, 2020

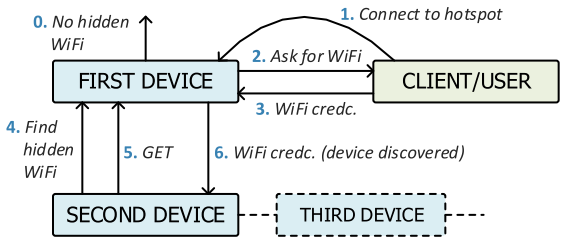


FIGURE 6. Discovery mechanism based on WiFi hotspot.

Algorithm 1: Device Discovery Based on WiFi

```
Function device.discovery()
    Input: device1, device2, device3,...
    // Three different ways to connect
    connSuccess ← false
    if ! SSID.isEmpty() then
        // 1. Try to connect to saved
           WiFi
        WiFi.begin()
        connSuccess ← WiFi.waitForConnectResult()
    end
    if ! connSuccess then
        // 2. Connect to WiFi
           credentials provider
        connSuccess ← connectWifiProvider()
    end
    if ! connSuccess then
        // 3. Using WiFiManager portal
           for asking credentials
        startWifiManagerPortal()
    end
    // Connected
    setupWifiProvider()
```

Algorithm 2: Data Mapping and Storage

```
Function map.dataItem()
    Input: data, hashTable[hashKey,server]
    hashKey ← SHA-2(data)
    foreach element e of the hashTable[hashKey] do
        hashKey′ ← hashTable[e]
        if hashKey ≤ hashKey′ then
            server ← hashTable(hashKey′)
        else
            server ← hashTable(min(hashKey′))
        end
    end
    // If data write/update request
    write(server, data)
    // If data read request
    data = read(server)
```

increases the difficulty of stealing security keys from the program code in the flash chip. For enhanced communication security, the TLS server and client certificates could be used for encryption and better trust between the credential provider and the receiver.

Algorithm 2 depicts a mechanism in which the data are distributed to multiple servers. The *data* and *hashTable* are the inputs to this algorithm, resulting in data storage in servers. Initially, the servers are mapped with the corresponding random SHA-2 keys in the *hashTable*. The hash table entries are then sorted from smallest to largest keys. The algorithm then proceeds as follows by (i) generating the SHA-2 key

of the requested *data*, (ii) searching for the corresponding server in the table, and (iii) storing and/or retrieving the data item to/from the server. We consider SQLite as a persistent database system for implementing the hash table [60].

V. PERFORMANCE ASSESSMENT AND DISCUSSIONS

To assess the performance of our proposed platform, we consider the Väre building use case. We measure scalability under two conditions: (i) *Strong scaling* indicates the increase in the number of servers for an application, and (ii) *Load scaling* indicates the change in the number of user requests for an application. For each condition, we calculate the latency and throughput. FIGURE 7 displays the experimental setup, which has four components. (i) Väre APIs for collecting real-time data from sensors attached to each room. We use ESP8266 microchips that collect real-time data and send them to our framework. There are around 100, 000 data items collected from numerous sensors all over the blocks (i.e., A to Y). These sensors include an air conditioning system (data related to CO2 concentration, inlet pressure, exhaust air) and a heating system (data related to heating temperature, valve adjustment, set point, snow melting pressure). (ii) Our implementation is executed on an HP EliteBook Windows laptop with specifications: RAM 8GB and 2.40GHz Intel Core i5 CPU. (iii) The O-MI nodes run inside Docker containers (v17.12) on a separate Linux machine with specifications: RAM 16GB and 3.20GHz 8-cores Intel CPU. In addition, the O-DF objects hierarchy of the Väre building data is depicted in one of the O-MI nodes user interface. (iv) The Apache JMeter (v5.0) runs on MacBook Pro with a 16GB RAM and 2.30GHz Intel CPU. This open-source software is used to send concurrent requests, which helps in simulating massive loads on the O-MI nodes.

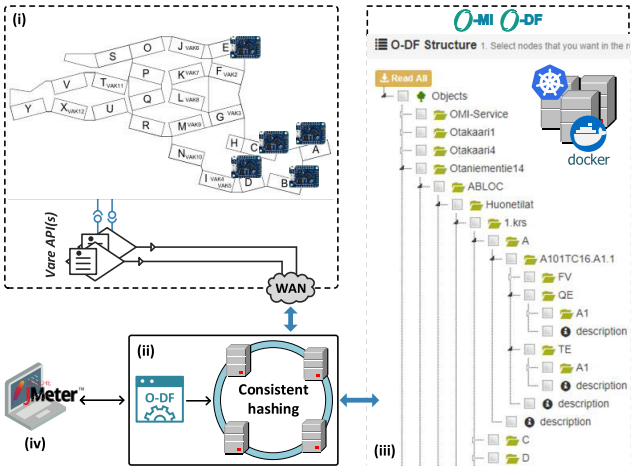
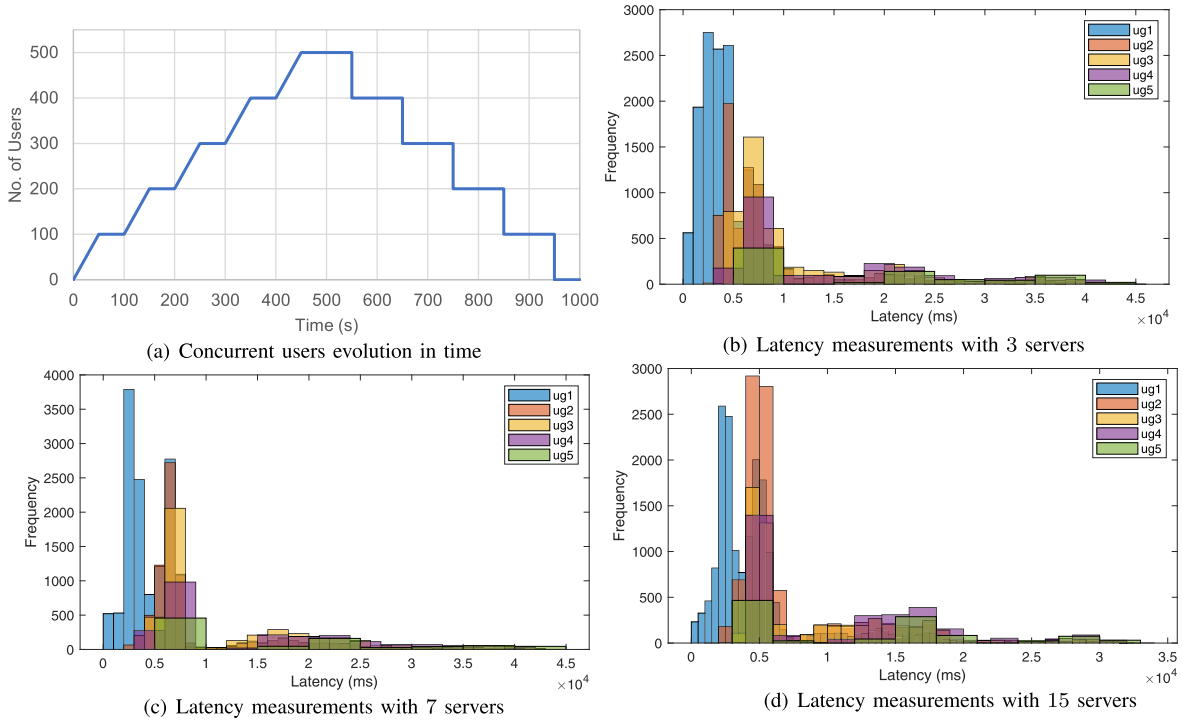


FIGURE 7. Running implementation of our proposed platform for Väre smart building.

A. SCENARIO 1: LOAD SCALING

In this scenario, we perform load testing by varying the number of users. FIGURE 8(a) shows the concurrent users progression with time, in which the number of users gradually increases to 500 (at t=450s) in a group of 100 users. At t=550s, the users begin to decline with a ratio of 100-by-100 until the end of the experiment (at t=1000s). With this setup, we measure the latency and throughput for each user group (ug) by sending random read messages. The data in the O-DF format are exchanged between the users and our framework, in which the users send O-DF messages and receive the requested data (i.e., in O-DF response). We use the compression feature of HTTP to reduce the request size. The size of the data is ≈950 bytes per O-DF InfoItem, which also varies by the amount of InfoItem(s) in the same read request. The graphs in FIGURE 8 represent the latency measurements distribution, providing data read latency as "latency" bins on the horizontal axis, and the frequency as the number of cases/requests obtained in each latency bin on the vertical axis for five different user groups. In this paper, latency is the time taken by the O-MI server to respond to a user's read request. As seen in FIGURE 8(b), the latency significantly rises when we increase the load as more time is needed to process thousands of concurrent requests. The highest number of requests (i.e., 2, 700 requests) is handled with latency 0.3×10^4ms/request. When the load is at its maximum (i.e., at t=500s when all user groups are active), around 1, 500 requests are processed with the latency of less than 1×10^4ms/request. Similarly, more requests (i.e., 3, 800 requests) are processed in the case of 7 servers in FIGURE 8(c) with the latency of 0.3×10^4ms/request. In addition, when all user groups are active at t=500s, the latency is almost similar to the 3-server case with value less than 1×10^4ms/request. This shows that the system is scalable; even when the number of servers increases, the latencies do not often change. Compared to FIGURE 8(b) and FIGURE 8(c), FIGURE 8(d) shows enhanced latency of value 0.5×10^4ms/request when the load is at its maximum. However, the number of requests is significantly smaller than the requests for 7- and 3-server cases. Overall, it can be observed that when the number of servers increases, the latency decreases but the number of requests becomes lower. Subsequently, the solution is sufficiently scalable and can adapt itself to the increasing number of servers as well as the number of concurrent users (i.e., load).

In addition to the latency distribution graphs, we measure the throughput in requests/sec (req/s) and average latency for each ug for 1, 3, 7, and 15 servers. The results are presented in TABLE 5. As can be seen, when the O-MI node scales up, the throughput increases as well, which shows that more requests are handled. In the case of 15 servers, the average latency for all user groups is higher, which is also the case with the throughput values. For ug1, the latency is 5182ms with a throughput of 18.6req/s (i.e., throughout the experiment in which the load is increasing and then decreasing). Although there is no significant difference in the throughput and latency for 1 and 15 servers, they allow us to confirm the previous results. However, the single O-MI node is unable to complete the request (*timeout* = *300s*) for ug1, ug2, and



(a) Concurrent users evolution in time

(b) Latency measurements with 3 servers

(c) Latency measurements with 7 servers

(d) Latency measurements with 15 servers

FIGURE 8. Latency measurements distribution for five user groups (ug) in three different server configurations. Frequency refers to the number of read requests in each "latency" bins obtained in our experiment.

TABLE 5. Throughput analysis for various user groups with increase in the number of servers. "∗" refers to timeout.

User group	O-MI node	# Requests	Average (ms)	Min (ms)	Max (ms)	Throughput (req/s)
1	1	11613	**8589**	15	∗	**11.3**
	3	15902	**5812**	20	45287	**16.7**
	7	15020	**6160**	18	42934	**15.7**
	15	17771	**5182**	59	32430	**18.6**
2	1	6977	**12830**	2408	∗	**7.5**
	3	8132	**8929**	3027	45086	**8.83**
	7	7600	**9567**	2216	42232	**10.1**
	15	9666	**7518**	1796	32137	**12.8**
3	1	4901	**16295**	4458	∗	**5.9**
	3	4538	**11649**	3841	43893	**8.1**
	7	4259	**12467**	4789	42000	**7.5**
	15	5740	**9235**	3445	31768	**10.2**
4	1	2874	**11577**	4468	39125	**7.8**
	3	2219	**14953**	3971	44651	**6.0**
	7	2303	**14439**	5002	42848	**6.1**
	15	2951	**11244**	3616	32035	**8.0**
5	1	1047	**12866**	4543	38857	**5.9**
	3	764	**17592**	5747	44731	**4.3**
	7	834	**16208**	5199	42295	**4.7**
	15	1034	**12779**	3987	32229	**5.9**

ug3 with the errors of 0.86%, 1.43%, and 2.04%, respectively. Hence, it can be observed that our proposed solution can significantly improve the throughput by efficiently distributing the load to multiple O-MI edge nodes.

B. SCENARIO 2: STRONG SCALING

This scenario increases the number of concurrent users from 20 to 100 in steps of 20 and measures the data read latency. We also increase the number of servers to analyze the impact on read latency of O-MI servers. The experimental results are shown in FIGURE 9. These graphs display a boxplot with the minimum, 1^{st} quartile, median, 3^{rd} quartile, and maximum of the latency calculated over the increasing number of users for each 1, 3, 7, and 15 server. For each user case (i.e., 20, 40, 60, 80, 100), the experiment was executed for 600s each. The latency values remain almost constant for all the

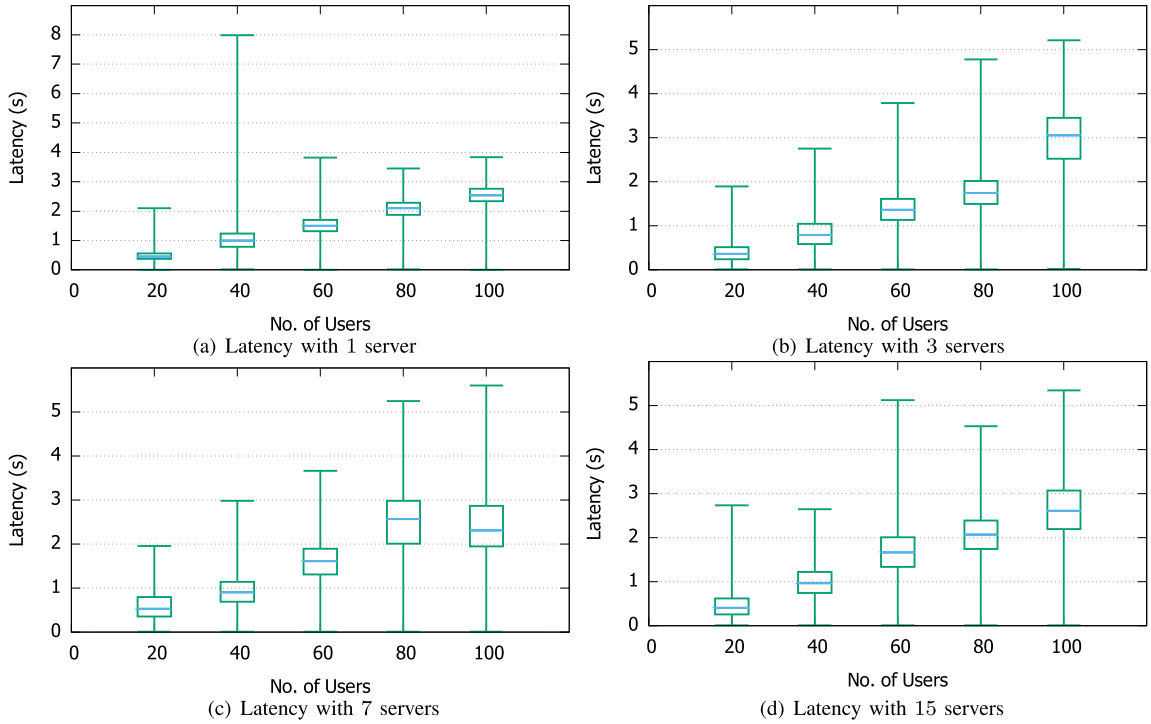


FIGURE 9. Impact of data read latency by increasing the number of concurrent users from 20 to 100 in steps of 20 with respect to the increasing number of O-MI servers.

O-MI servers for the fixed number of users. When the number of O-MI servers increases in FIGURE 9(d), there is no significant increase in the latency values for other server cases. This explains why our framework is sufficiently scalable as the latency does not increase even with a smaller number of servers. We are saving the cost of more servers as the data load is handled much efficiently with fewer servers without the need of more servers. Hence, it can be observed that our proposed solution is capable of handling the load much efficiently even with a smaller number of servers.

VI. CONCLUSION, LIMITATIONS, AND FUTURE DIRECTIONS

In this paper, we propose an open and scalable IoT platform for smart environments based on edge computing. It allows dynamic discovery of devices (i.e., Things, actuators), their configuration, and data management. Our proposed solution offers four features. (i) Scalability with respect to the number of users/inputs and computing environment, which implies that it can handle an increased number of users. (ii) The platform overcomes vertical silos by providing standardized communication protocols and the semantic data models that are open and should not change from one smart environment to another. (iii) The device discovery in which the mobile IoT devices are able to discover themselves to publish/consume data and/or services. (iv) The data management is performed at the edge of the network by relying on a consistent hashing load balancing to optimize data publication, data consumption, and network latency. This new IoT platform is composed of Discovery, Storage, and Service planes. The proposed solution is evaluated on the Väre building use case in the Aalto University campus by calculating latency and throughput measurements. The key component of this use case is the O-MI node, which is developed by adopting the O-MI and O-DF messaging standards. It can be concluded from the experimental results that the proposed platform improves latency without any additional overhead of adding more servers, ensuring an interoperable and scalable solution.

A. LIMITATIONS

Despite commendable improvements in the IoT technology for smart computing systems, some limitations still remain to be addressed in the proposed system:

- For the discovery mechanism, there is no method of automatically retrieving the WiFi SSID when the network has been changed for mobile devices. It can be performed by creating a list of possible WiFi SSIDs and programming it into the system.
- The mobile devices should be aware of the endpoint for their discovery and data publication. This can be achieved with any local area network service discovery protocol after connecting to the correct WiFi network. For example, DNS-SD (DNS Service Discovery) can be used to advertise and discover the endpoint. This has been implemented in the O-MI node reference implementation.
- All mobile devices need to have a shared secret, which is used to keep the WiFi credentials (SSID and password)

safe from any other users. If this secret leaks, either by employee or by hardware reverse engineering, all devices will be vulnerable to local attacks.

B. FUTURE WORK

Our future work will address the above limitations for enhanced discovery and security features. Further, the related future research avenues could include:

- The optimization of a consistent hashing module by implementing WebSocket communications.
- The integration of our platform Service plane with the O-MI security module to provide authentication- and authorization-related security management.
- Methods for more secure automatic sharing of WiFi credentials during the WiFi network discovery.

REFERENCES

[1] M. Collotta and G. Pau, "Bluetooth for Internet of Things: A fuzzy approach to improve power management in smart homes," *Comput. Electr. Eng.*, vol. 44, pp. 137–152, May 2015.
[2] X.-Y. Chen and Z.-G. Jin, "Research on key technology and applications for Internet of Things," *Phys. Procedia*, vol. 33, pp. 561–566, 2012.
[3] A. Čolaković and M. Hadžialić, "Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues," *Comput. Netw.*, vol. 144, pp. 17–39, Oct. 2018.
[4] M. Weiser, "The computer for the 21st century," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 3, no. 3, pp. 3–11, Jul. 1999, doi: 10.1145/329124.329126.
[5] F. Al-Turjman and J. P. Lemayian, "Intelligence, security, and vehicular sensor networks in Internet of Things (IoT)-enabled smart-cities: An overview," *Comput. Electr. Eng.*, vol. 87, Oct. 2020, Art. no. 106776.
[6] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.
[7] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities," *Future Gener. Comput. Syst.*, vol. 87, pp. 278–289, Oct. 2018.
[8] S. Kubler, J. Robert, A. Hefnawy, K. Framling, C. Cherifi, and A. Bouras, "Open IoT ecosystem for sporting event management," *IEEE Access*, vol. 5, pp. 7064–7079, 2017.
[9] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Trans. Netw.*, vol. 11, no. 1, pp. 17–32, Feb. 2003.
[10] K. Främling. The Open Group, Standard. (Dec. 2019). *Open Messaging Interface (O-MI)*. [Online]. Available: www.opengroup.org/library/c19e
[11] The Open Group, Standard. (Dec. 2019). *Open Data Format (O-DF)*. [Online]. Available: www.opengroup.org/library/c19d
[12] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[13] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, "Fog computing: Principles, architectures, and applications," in *Internet Things*. Amsterdam, The Netherlands: Elsevier, 2016, pp. 61–75.
[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[15] T. Anagnostopoulos, A. Zaslavsky, K. Kolomvatsos, A. Medvedev, P. Amirian, J. Morley, and S. Hadjieftymiades, "Challenges and opportunities of waste management in IoT-enabled smart cities: A survey," *IEEE Trans. Sustain. Comput.*, vol. 2, no. 3, pp. 275–289, Jul. 2017.
[16] A. Botta, W. De Donato, V. Persico, and A. Pescapé, "Integration of cloud computing and Internet of Things: A survey," *Future Gener. Comput. Syst.*, vol. 56, pp. 684–700, 2016.
[17] M. V. Moreno, F. Terroso-Sáenz, A. González-Vidal, M. Valdés-Vela, A. F. Skarmeta, M. A. Zamora, and V. Chang, "Applicability of big data techniques to smart cities deployments," *IEEE Trans. Ind. Informat.*, vol. 13, no. 2, pp. 800–809, Apr. 2017.
[18] H. Guo, J. Ren, D. Zhang, Y. Zhang, and J. Hu, "A scalable and manageable IoT architecture based on transparent computing," *J. Parallel Distrib. Comput.*, vol. 118, pp. 5–13, Aug. 2018.
[19] F. Palmieri, M. Ficco, S. Pardi, and A. Castiglione, "A cloud-based architecture for emergency management and first responders localization in smart city environments," *Comput. Electr. Eng.*, vol. 56, pp. 810–830, Nov. 2016.
[20] A. de M. Del Esposte, E. F. Z. Santana, L. Kanashiro, F. M. Costa, K. R. Braghetto, N. Lago, and F. Kon, "Design and evaluation of a scalable smart city software platform with large-scale simulations," *Future Gener. Comput. Syst.*, vol. 93, pp. 427–441, Apr. 2019.
[21] J. Venkatesh, B. Aksanli, C. S. Chan, A. S. Akyürek, and T. S. Rosing, "Scalable-application design for the IoT," *IEEE Softw.*, vol. 34, no. 1, pp. 62–70, Jan. 2017.
[22] I. Zyrianoff, F. Borelli, G. Biondi, A. Heideker, and C. Kamienski, "Scalability of real-time IoT-based applications for smart cities," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jun. 2018, pp. 688–693.
[23] Y. Xu and A. Helal, "Scalable cloud–sensor architecture for the Internet of Things," *IEEE Internet Things J.*, vol. 3, no. 3, pp. 285–298, Jun. 2016.
[24] C.-L. Tseng and F. J. Lin, "Extending scalability of IoT/M2M platforms with fog computing," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 825–830.
[25] T. N. Gia, A.-M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Fault tolerant and scalable IoT-based architecture for health monitoring," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, Apr. 2015, pp. 1–6.
[26] I. Zyrianoff, A. Heideker, D. Silva, and C. Kamienski, "Scalability of an Internet of Things platform for smart water management for agriculture," in *Proc. 23rd Conf. Open Innov. Assoc. (FRUCT)*, Nov. 2018, pp. 432–439.
[27] I. Miladinovic and S. Schefer-Wenzl, "A highly scalable IoT architecture through network function virtualization," *Open J. Internet Things*, vol. 3, no. 1, pp. 127–135, 2017.
[28] C. Cabrera and S. Clarke, "A self-adaptive service discovery model for smart cities," *IEEE Trans. Services Comput.*, early access, Sep. 27, 2019, doi: 10.1109/TSC.2019.2944356.
[29] C. Vandana and A. A. Chikkamannur, "Study of resource discovery trends in Internet of Things (IoT)," *Int. J. Adv. Netw. Appl.*, vol. 8, no. 3, p. 3084, 2016.
[30] D. Kourtesis and I. Paraskakis, "Combining SAWSDL, OWL-DL and UDDI for semantically enhanced Web service discovery," in *Proc. Eur. Semantic Web Conf.* Berlin, Germany: Springer, 2008, pp. 614–628.
[31] K. Lee, S. Kim, J. P. Jeong, S. Lee, H. Kim, and J.-S. Park, "A framework for DNS naming services for Internet-of-Things devices," *Future Gener. Comput. Syst.*, vol. 92, pp. 617–627, Mar. 2019.
[32] Engineering Research Center for the Internet of Things, "AC02—'Naming, addressing, search, discovery,'" Eur. Res. Cluster Internet-of-Things, Oslo, Norway, Tech. Rep., 2014.
[33] R. Khan and S. U. Khan, "Design and implementation of UPnP-based energy gateway for demand side management in smart grid," *J. Ind. Inf. Integr.*, vol. 8, pp. 8–21, Dec. 2017.
[34] S. Cheshire and M. Krochmal, *Multicast DNS*, document RFC 6762, Internet Engineering Task Force, 2013.
[35] J.-T. Kim, Y.-J. Oh, H.-K. Lee, E.-H. Paik, and K.-R. Park, "Implementation of the DLNA proxy system for sharing home media contents," *IEEE Trans. Consum. Electron.*, vol. 53, no. 1, pp. 139–144, Feb. 2007.
[36] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. (2004). *OWL-S: Semantic Markup for Web Services*. [Online]. Available: https://www.w3.org/Submission/OWL-S/
[37] J. Sangers, F. Frasincar, F. Hogenboom, and V. Chepegin, "Semantic Web service discovery using natural language processing techniques," *Expert Syst. Appl.*, vol. 40, no. 11, pp. 4660–4671, Sep. 2013.
[38] E. Prud'hommeaux and A. Seaborne. (Jan. 2008). *SPARQL query language for RDF, W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/
[39] H. Bohn, F. Golatowski, and D. Timmermann, "Dynamic device and service discovery extensions for WS-BPEL," in *Proc. Int. Conf. Service Syst. Service Manage.*, Jun. 2008, pp. 1–6.
[40] C. Sarkar, S. N. A. U. Nambi, R. V. Prasad, and A. Rahim, "A scalable distributed architecture towards unifying IoT applications," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Mar. 2014, pp. 508–513.
[41] M. Gheisari, G. Wang, and S. Chen, "An edge computing-enhanced Internet of Things framework for privacy-preserving in smart city," *Comput. Electr. Eng.*, vol. 81, Jan. 2020, Art. no. 106504.

[42] A. Tolk, S. Y. Diallo, and C. D. Turnitsa, "Applying the levels of conceptual interoperability model in support of integratability, interoperability, and composability for system-of-systems engineering," *J. Syst., Cybern., Informat.*, vol. 5, no. 5, pp. 65–74, 2007.
[43] A. Brutti, P. De Sabbata, A. Frascella, N. Gessa, R. Ianniello, C. Novelli, S. Pizzuti, and G. Ponti, "Smart city platform specification: A modular approach to achieve interoperability in smart cities," in *The Internet Things for Smart Urban Ecosystems*. Cham, Switzerland: Springer, 2019, pp. 25–50.
[44] T. Li, Y. Liu, Y. Tian, S. Shen, and W. Mao, "A storage solution for massive IoT data based on NoSQL," in *Proc. IEEE Int. Conf. Green Comput. Commun.*, Nov. 2012, pp. 50–57.
[45] N. Padalia, *Apache Cassandra Essentials*. Birmingham, U.K.: Packt Publishing, 2015.
[46] W. Tian, Y. Zhao, Y. Zhong, M. Xu, and C. Jing, "A dynamic and integrated load-balancing scheduling algorithm for cloud datacenters," in *Proc. IEEE Int. Conf. Cloud Comput. Intell. Syst.*, Sep. 2011, pp. 311–315.
[47] A. Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: Integration and load balancing in data centers," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2008, pp. 1–12.
[48] Y. Yu, X. Li, and C. Qian, "SDLB: A scalable and dynamic software load balancer for fog and mobile edge computing," in *Proc. Workshop Mobile Edge Commun.*, 2017, pp. 55–60.
[49] A. Javed, J. Robert, K. Heljanko, and K. Främling, "IoTEF: A federated edge-cloud architecture for fault-tolerant iot applications," *J. Grid Comput.*, vol. 18, no. 1, pp. 1–24, 2020.
[50] S. Kubler, K. Främling, and W. Derigent, "P2P data synchronization for product lifecycle management," *Comput. Ind.*, vol. 66, pp. 82–98, Jan. 2015.
[51] M. M. E. Mahmoud, J. J. P. C. Rodrigues, K. Saleem, J. Al-Muhtadi, N. Kumar, and V. Korotaev, "Towards energy-aware fog-enabled cloud of things for healthcare," *Comput. Electr. Eng.*, vol. 67, pp. 58–69, Apr. 2018.
[52] L. Vallad, "Oracle data quality for product data knowledge studio reference guide, release 5.6.2," Oracle, Santa Clara, CA, USA, Tech. Rep., 2011.
[53] V. Cardellini, M. Colajanni, and P. S. Yu, "Dynamic load balancing on Web-server systems," *IEEE Internet Comput.*, vol. 3, no. 3, pp. 28–39, May 1999.
[54] D. Eastlake, III, and T. Hansen, *US Secure Hash Algorithms (SHA and HMAC-SHA)*, document RFC 4634, Motorola Labs and AT &T Labs, 2006.
[55] B. Jan, H. Farman, M. Khan, M. Imran, I. U. Islam, A. Ahmad, S. Ali, and G. Jeon, "Deep learning in big data analytics: A comparative study," *Comput. Electr. Eng.*, vol. 75, pp. 275–287, May 2019.
[56] S. K. Peddoju and H. Upadhyay, "Evaluation of IoT data visualization tools and techniques," in *Data Visualization*. Singapore: Springer, 2020, pp. 115–139.
[57] K. Framling, S. Kubler, and A. Buda, "Universal messaging standards for the IoT from a lifecycle management perspective," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 319–327, Aug. 2014.
[58] A. Javed, N. Yousefnezhad, J. Robert, K. Heljanko, and K. Främling, "Access time improvement framework for standardized IoT gateways," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, Mar. 2019, pp. 220–226.
[59] N. Kolbe, J. Robert, S. Kubler, and Y. Le Traon, "PROFICIENT: Productivity tool for semantic interoperability in an open IoT ecosystem," in *Proc. 14th EAI Int. Conf. Mobile Ubiquitous Systems: Comput., Netw. Services*, 2018, pp. 116–125.
[60] M. Owens and G. Allen, *SQLite*. New York, NY, USA: Springer, 2010.

ASAD JAVED (Member, IEEE) received the B.S. degree in computer engineering from COMSATS University Islamabad (CUI), Pakistan, in 2013, and the M.Sc. degree in computer science with a major in embedded systems from the EIT Digital Master School, in 2016, which is the leading open innovation European organization. It was a double degree program with first year of master studies conducted at the KTH Royal Institute of Technology, Sweden, and the exit year at Aalto University, Finland. He is currently pursuing the Ph.D. degree with the Department of Computer Science, Aalto University, with a focus on the Internet of Things, edge computing, fault-tolerant distributed systems, and cloud technologies. His doctoral research aims to propose, implement, and evaluate microservices-based IoT applications that are scalable, fault-tolerant, and performance efficient.

AVLEEN MALHI received the M.Sc. and Ph.D. degrees in computer science engineering from Thapar University, India, in 2012 and 2016, respectively. She was employed as an Assistant Professor with Thapar University, for four years. She is currently working as a Postdoctoral Researcher with the Department of Computer Science, Aalto University. She worked on the security of Vehicular Ad-Hoc Networks on an Industry Sponsored project by TATA Consultancy Services, India, for four years during her Ph.D. studies. Her research interests include the IoT, machine learning, and information security. She has 17 SCIE journal publications and 20 international conferences mainly in the area of machine learning, the IoT, and security. She also plays a leading role in Business Finland Project, eParkly for smart parking management systems.

TUOMAS KINNUNEN received the B.Sc. degree in computer science from Aalto University, Finland. He is currently the Technical Lead of the O-MI Node software, which is the reference implementation of the adopted Open Group standards, Open Messaging Interface (O-MI), and Open Data Format (O-DF). He is also working and studying with the Department of Computer Science, Aalto University. He started programming at the age of 13 years. His current research interests include the Internet of Things, artificial intelligence, distributed systems, and functional programming.

KARY FRÄMLING is currently working as a Full Professor with Umeå University, Sweden, and an Adjunct Professor with Aalto University, Finland. He is one of the first movers in the space of IoT and has worked on joint projects with tens of industrial partners, such as BMW and Nokia. He is also a Founder of a successful IoT startup, ControlThings. He is also a Founder and a former Chairman of the IoT Work Group of The Open Group, which published the first IoT standards that address all IoT-connected systems in October 2014: the Open Messaging Interface (O-MI) and Open Data Format (O-DF). The standards were initially developed jointly with leading industrial partners but he is the main architect and author of these standards. The course will benefit from the insights and the network that he can provide and he is committed to this project beyond research and through commercialization.