GRIFFITH UNIVERSITY

Hierarchical Decentralised Edge Interoperability

Author

Azad, Tanzima, Newton, MA Hakim, Trevathan, Jarrod, Sattar, Abdul

Published

2023

Journal Title

IEEE Internet of Things Journal

Version

Accepted Manuscript (AM)

DOI

10.1109/jiot.2023.3340298

Rights statement

This work is covered by copyright. You must assume that re-use is limited to personal use and that permission from the copyright owner must be obtained for all other uses. If the document is available under a specified licence, refer to the licence for details of permitted re-use. If you believe that this work infringes copyright please make a copyright takedown request using the form at https://www.griffith.edu.au/copyright-matters.

Downloaded from

http://hdl.handle.net/10072/427946

Griffith Research Online

https://research-repository.griffith.edu.au

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

Hierarchical Decentralised Edge Interoperability

Tanzima Azad[1,5,†], M A Hakim Newton[2,6,7,†], Jarrod Trevathan[3,5,6], Abdul Sattar[4,5,6]

***Abstract*—The Internet of Things (IoT) has many important applications in multiple domains that include home automation, smart cites, healthcare, agriculture, and environment. IoT comprises a wide range of sensors and actuators that communicate with each other over cloud, fog and edge level networks. Moreover, these devices use various communication protocols and are made by different manufactures. To deal with these diversities, IoT essentially needs interoperable communication interfaces among devices. Unfortunately, existing interoperability solutions are centralised and use fog or cloud level computing resources, making IoT communications latency-prone and poorly scalable. These issues could be handled effectively, if edge level devices could be made interoperable within the edge level and without needing fog or cloud level access. This paper proposes a decentralized interoperability solution that stays fully within the edge level. The solution relies on controller devices that work on the interface boundaries of the edge devices. Unlike existing solutions, the proposed solution adopts a hierarchical interoperability model to handle interoperability at network, syntactical, semantic, and organizational levels. Our solution is non-proprietary, generic over vendors and platforms, and easily extendable to new devices. We compare our proposed solution with existing interoperability solutions for edge devices and show its mobility, efficiency and flexibility.**

***Index Terms*—Internet of Things; Edge Devices; Syntactic Interoperability; Semantic Interoperability; Network Interoperability; Organizational Interoperability.**

I. Introduction

THE *Internet of Things* (IoT) is a global infrastructure for interconnections among solely identifiable things over existing and evolving *interoperable* information and communication technologies [1], [2]. Things in IoT are *sensors* and *actuators* that can respectively collect information from and effectuate actions on the surroundings. IoT has applications in many domains that include home automation, smart cities [3], supply chains [3], healthcare [4], agriculture [5], and environmental monitoring [6]. However, interoperability is a significant challenge [7] for IoT solutions: sensors and actuators from different vendors use diverse non-standard communication protocols and cannot communicate easily [8].

Interoperability ensures devices with varying parameters can communicate with each other to give and take required services. A modular farming system could have monitoring and controlling devices for heating, ventilation, and air conditioning. These devices could be from different vendors, supporting different network technologies, passing messages in different formats, using different message passing protocols, and having different semantics of the terminologies used by each other. Fig. 1 shows an overall diversity scenario in IoT. In such a scenario, intermediate interfaces are required to enable effective information exchange. Unfortunately, there is often no satisfactory adapter or translator to achieve bridging on demand between various IoT devices or applications.

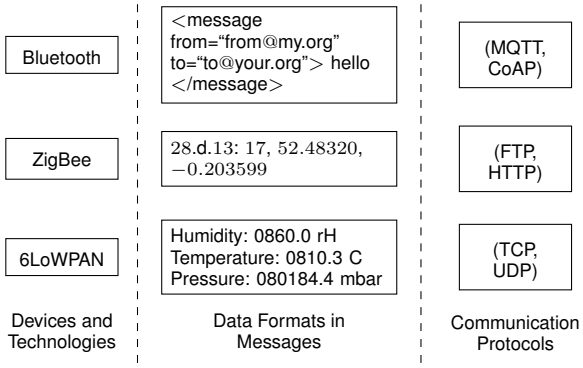


Fig. 1. Diversity and interoperability in IoT communication

IoT devices usually communicate over *cloud*, *fog*, and *edge* levels. In cloud networks, devices send and receive data to and from heavy duty centralised processing servers. Edge networks reduce communication latency by keeping data transmission and processing locally at low duty distributed computing devices. Fog networks are at intermediate levels to obtain a balance between the communication latency of the cloud levels and the processing limitation of edge devices. IoT interoperability solutions essentially need to take networks of varying granularity into account. So far significant work has been done to address IoT interoperability issues, particularly for cloud and fog networks [9]–[11], but not for edge networks. In this paper, considering mission and time critical applications [12] such as health or environment monitoring, we study interoperability issues in edge-level devices and edge-level networks, so that data processing and decision making can be done locally within the edge level and in real-time. In this respect, edge intelligence [13] is relevant. Edge intelligence refers to deploying artificial intelligence (AI) algorithms for processing data and making decisions locally, near the edge of the network or directly at the edge devices, rather than relying on a cloud-based system or a centralised server. Edge intelligence thus enables real-time analysis, reduced latency, and efficient use of bandwidth.

Existing solutions for interoperability of edge devices include TinySOS [14], EdgeX Foundry [15] and mobile gateways [16]–[18]. TinySOS [14] is a compact web server for IoT devices, enabling them to host simplified Open Geospatial

[1]tanzima.azad@griffithuni.edu.au, [2]mahakim.newton@newcastle.edu.au
[3]j.trevathan@griffith.edu.au, [4]a.sattar@griffith.edu.au
[5]School of Information and Communication Technology, Griffith University, 170 Kessels Road, Nathan, QLD 4111, Australia
[6]Institute for Integrated and Intelligent Systems, Griffith University, 170 Kessels Road, Nathan, QLD 4111, Australia
[7]School of Information and Physical Sciences, The University of Newcastle, University Drive, Callaghan, NSW 2308, Australia
[†]These two authors are joint first-authors

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

Consortium (OGC) Sensor Observation Services and interact with other systems using XML and cloud-based OGC service for actuator tasks. EdgeX Foundry [15] is a decentralized, open-source microservice framework that interacts with physical devices, but requires substantial memory, storage, and specific operating systems, limiting its compatibility with resource-constrained devices like microcontrollers. A user-space programmable IoT gateway [16] connects wireless sensor networks to mobile networks or the Internet, supporting data forwarding and protocol conversion, but requires users to install server software on their PC to access sensor data. A World Wide Web Consortium (W3C) Web of Things (WoT) extension [17], based on the Arrowhead Framework (AHF) [19], addresses IoT interoperability challenges through local cloud deployment and edge computing services, and evaluates the approach in a small-scale edge computing testbed. However, WoT does not support IP address change associated with IoT device mobility. The European Telecommunications Standards Institute (ETSI) Machine to Machine (M2M) Gateway [18] on a mobile device addresses the challenges of interpreting and implementing M2M middleware standards while providing libraries for easier deployment of IoT applications. However, both approaches consume excessive energy.

The above summary shows that most existing edge level interoperability solutions suffer from two major drawbacks:

1) There is no local processing or decision-making feature in existing cloud-based solutions. The architectures of the solutions are complex in nature and can not be used for edge devices without the need for the cloud.
2) Existing edge interoperability solutions are not dynamic and does not work with limited bandwidth, uncertain internet connectivity. They are also not compatible with various edge devices that are resource constrained and that use diverse platforms and protocols.

This paper seeks to address the aforementioned problems by proposing a solution to interconnect IoT edge devices, to support local decision-making, and to incorporate all related services uniformly in one application. This paper proposes a decentralized interoperability solution that stays fully within the edge level. The solution relies on controller devices that work on the interface boundaries of the edge devices. Unlike existing solutions, the proposed solution adopts a hierarchical interoperability model to handle interoperability at network, syntactical, semantic, and organizational levels. Our solution is non-proprietary, generic over vendors and platforms, and easily extendable to new devices. We compare our proposed solution with existing interoperability solutions for edge devices and show its mobility, efficiency and flexibility.

This paper is organised as follows: Section II provides certain terminologies used in this paper. Section III covers related work on existing solutions to address interoperability issues in IoT. Section IV proposes a solution that connects IoT edge devices and enables localized decision-making by integrating all relevant services into a single application. Section V provides a comparison of the proposed solution with existing solutions. Section VI presents a summary of key findings and suggests potential future research directions.

II. Terminologies

We describe key terminologies used in this paper.

1) **Model:** An interoperability model provides a categorisation of interoperability issues from various perspectives. The categorisation could be hierarchical or non-hierarchical, vertical or horizontal, or even be mixed.
2) **Level:** An interoperability level denotes a category in a hierarchical or vertical interoperability model. Interoperability levels are typically sequential in the hierarchy, having larger grannular ones towards the top and each level typically building on the level just underneath.
3) **Class:** An interoperability class denotes a category in a horizontal interoperability model. The classes are all at the same level, having no mutual direct dependence.
4) **Solution:** An interoperability solution denotes an implementation framework that potentially implements a given interoperability model. An interoperability solution could essentially denote an actual implementation, but actual implementations are out of scope of this paper.
5) **Service:** An implementation service is an algorithmic method that implements a particular data or execution functionality for an interoperability solution.
6) **Layer:** An implementation layer is for a vertical organization of the services needed for an interoperability solution. Each layer is responsible to implement a set of services sharing certain traits of functionalities.
7) **Environment:** An IoT environment denotes a connected system comprising physical objects, devices, sensors, and actuators that interact and share data with each other.
8) **Interoperable Environment:** An interoperable environment ensures interoperability among devices, sensors, and actuators so that communications, collaborations, and data sharing could take place easily and seamlessly.
9) **Data and Message:** Data is read from or write to devices. Messages are transmitted over the network and contains data and routing information.

III. Related Work

We discuss interoperability categorisations both in the general and the IoT specific contexts. We also cover existing cloud, fog, and edge based interoperability solutions.

A. Interoperability Categorisations

There exists interoperability models for the general contexts. The Organizational Interoperability Maturity Model (OIMM) [20] focuses on human-related aspects such as shared objectives, similar approaches, mutual comprehension, and established methods of interaction. The OIMM classifies interoperability as independent, ad hoc, collaborative, combined, and unified. The Levels of IT Systems Interoperability (LISI) model [21] focuses on physical connections between systems. The LISI model comprises of isolated, connected, distributed/functional, integrated/domain, and enterprise levels. The Levels of Conceptual Interoperability Model (LCIM) [22] addresses interoperability from a data standpoint. The LCIM has five levels but its enhanced and expanded version [23]

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

has seven levels: no interoperability; and technical, syntactic, semantic, pragmatic, dynamic, and conceptual interoperability.

In the IoT specific context, interoperability has been categorised from various perspectives such as *organizational*, *systems*, and *data* perspectives [24]. IoT interoperability also has *syntactic*, *semantic*, *network*, *device*, and *platform* viewing angles [11]. Note both of these categorisations are horizontal and generic over IoT levels such as cloud, fog, and edge.

B. Cloud-Based Interoperability Solutions

There are more than 300 cloud based platforms [25] that include Brillo from Google, HomeKit from Apple, Jasper from Cisco, AWS IoT from Amazon, Watson from IBM, AllJoyn from Qualcomm, and Azure IoT from Microsoft. Interoperability solutions need to consider this huge number.

A middleware-based interoperability solution [26] at the cloud level uses an event-driven approach to solve semantic and syntactic interoperability by presenting a communication gateway to interact with IoT sensors. The main limitation of this work is it uses a single wireless communication protocol (WiFi) to connect the sensors and the applications. So it has not been able to address the network interoperability problem. Later, it has been extended [27] by adding two other network technologies to increase network interoperability. However, the extended solution is still not able to connect devices using other network communication protocols. The Managed Ecosystems of Networked Objects (MENO) [28] provides seamless communication between sensors, actuators, and other IP-based devices. Internet of Things Virtual Network (IoT-VN) [29] has utilized MENO's ideas to connect devices to the internet with limited resources. This design provides seamless end-to-end connection, but the main concerns are integration of existing technologies and scalability.

In IoT interoperability, Wireless Sensor Networks (WSNs) and Sensor Web play a vital role [30]. In order to achieve universal interoperability, a promising solution is to use open standards for the Sensor web. In this regard, the Open Geospatial Consortium (OGC) is one of the leading organizations in the Sensor Web standardization. OGC has been providing geospatial interoperability since 1994. Moreover, The Sensor Web Enablement (SWE) is a collection of OGC standards designed for IoT. It encompasses various standards, including Observations and Measurements (O&M) [31], Sensor Observation Service (SOS) [32], Sensor Model Language (SensorML) [33], Sensor Planning Service (SPS) [34], SensorThings API [35], and PUCK protocol [36]. However, the architecture of the SWE standards considers large-scale sensor deployments except for the SensorThings API, which still uses the cloud-based architecture and has response latency.

C. Fog Based Interoperability Solutions

A fog-based solution [37] to deal with semantics in IoT aims to leverage interoperability by moving certain frequently used cloud services to the fog. The objective is to decrease task execution time and energy consumption. However, its service delays increase as the number of fog nodes grows. Another fog-based abstract solution [38] proposes a Web of Virtual Things (WoVT) server, which can be implemented in the fog layer to address IoT interoperability issues. However, in this solution, devices using even the same communication protocol cannot exchange data, if the vendor-specific device information is stored in incompatible formats.

D. Edge Based Interoperability Solutions

TinySOS [14] is a small web server that hosts a simplified Open Geospatial Consortium (OGC) Sensor Observation Service (SOS). IoT devices can describe themselves to a TinySOS web server, operate independently, and interact with other devices. However, TinySOS keeps the data in XML format, which is complex for edge devices. Also, to send any task to the actuators, TinySOS needs to use other services of OGC, which is actually a cloud based architecture.

INTER-IoT [39] proposes to achieve IoT interoperability by combining various systems across six levels: technical, syntactic, semantic, pragmatic, dynamic, and conceptual. The goal is achieved through three key components responsible for compatibility among devices, networks, middlewares, applications, services, data and semantics. The INTER-IoT project employs a hybrid cloud-edge structure, allowing certain components like the devices and middlewares to operate at the edge while certain other components like applications and services to stay in the cloud. Overall, INTER-IoT is a complex framework that can be difficult to use for simple IoT systems. Also, developing and deploying IoT systems using the INTER-IoT project can be expensive, especially for large-scale systems.

EdgeX Foundry [15] comprises a set of decentralized, open source microservice assemblages that are situated at the network's periphery. These assemblages have the ability to engage with the tangible realm encompassing devices, sensors, and actuators. EdgeX has been used in developing a monitoring system to address the challenge of managing various communication protocols and limited cloud computing resources. However, EdgeX requires minimum of 1 GB of memory, minimum of 3 GB of space to run the EdgeX Foundry containers, and approximately 32GB of storage is minimally recommended to start. The operating system requirements are Windows 7 and higher, Ubuntu Desktop/Server 14 and higher, Ubuntu Core 16 and higher and Mac OS X. This means EdgeX Foundry has limitations to support resource-constrained devices like microcontrollers.

IV. Proposed Edge Interoperability

To deal with edge interoperability, we propose an *interoperable environment*, a *hierarchical model* and a *layer-based solution*. Nevertheless, several key considerations behind our proposed solution are as follows:

1) All sensors, devices, and actuators should be identified uniquely within the interoperable environment;
2) Currently available devices that are using any standard or proprietary solutions should be efficiently and conveniently connected within the interoperable environment;
3) Incorporation of various protocols within the interoperable environment should be straightforward and easy.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

A. Proposed Interoperable Environment

To deal with various levels of heterogeneity among edge devices, an interoperable environment arguably needs an interface device having some processing power. We propose to use a *controller* device for this. A controller device is typically a microcontroller such as arduino uno, arduino nano, arduino mega, raspberry pi, esp8266, and esp32. The processing power is needed for data transformation, message transmission, and to provide various related services at the interface points. It is worth noting here that some actuators or sensors might have in-built processing power or a controller might have in-built sensing or actuation capabilities. For convenience, in this paper, based on the functionalities, we just consider each such device as multiple distinct devices. Nevertheless, in an interoperable environment, edge devices such as sensors and actuators are connected via a controller device. Fig. 2 shows an example connectivity between a sensor and an actuator via a controller. Such a connectivity in an interoperable environment could be achieved through a wired or a wireless network.

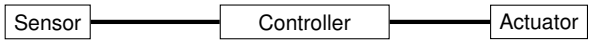


Fig. 2. Connectivity in the proposed interoperable environment between a sensor and an actuator via a controller

Each device at the edge level has tighter resource constraints than any fog or cloud level devices. Controller devices for the proposed interoperability environments are no exception. So a controller essentially has a limitation on the number of devices it can manage or even be connected with. When the number of devices exceeds this limit, any further devices are connected to a new controller. The controllers are then connected with each other to enable communication between devices connected to them. Note that the connection among the controllers could be wired or wireless and could use various communication protocols. Fig. 3 depicts an example connectivity network among interconnected controllers.

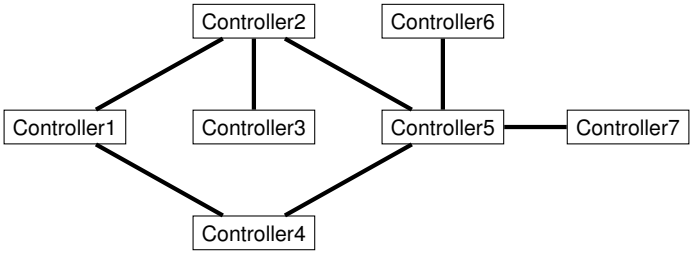


Fig. 3. Communication among devices through controllers

With the type of connectivity that has been shown in Fig. 3 among controllers, it is reasonable to ask about the routing issue. We propose to use a simple routing approach for this. Controllers will store the information about their neighbours and also the information about the source and destination devices for each possible communication event. Given that an edge network is typically small in size, computing and storing the routing information is not difficult. We discuss further details of the routing algorithm later in the paper.

B. Proposed Interoperability Model

As we have seen in the related work, existing IoT interoperability models [11], [24] have made attempts to classify interoperability at a horizontal level. Horizontal level classifications essentially do not respect any natural granularity levels presented in the interoperability issues. Moreover, managing all types of interoperability issues at the same level does not appear to be conceptually simple in terms of mutual relations among various issues. So in this paper, we propose to take a vertical or hierarchical approach to edge interoperability. Our proposal is also inspired by the Open Systems Interconnection (OSI) model for systems interconnection. Fig. 4 illustrates our proposed hierarchical edge interoperability model. It has 4 levels: *network level*, *syntactic level*, *semantic level*, and *organizational level*. Below we briefly describe the levels.

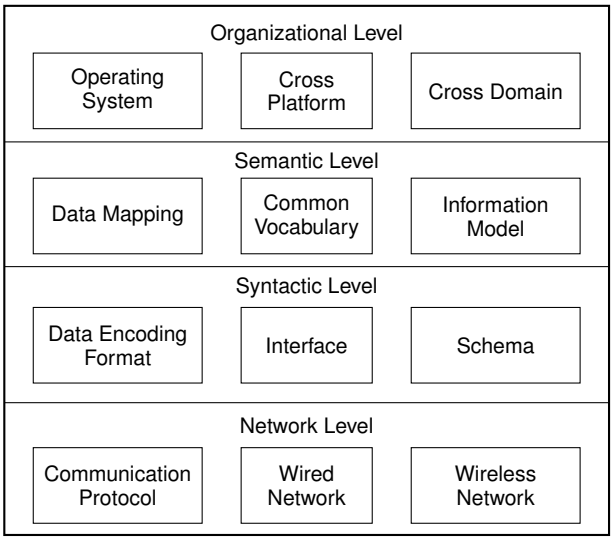


Fig. 4. Hierarchical interoperability levels

1) Network Level: The network level handles diverse networking technologies to ensure connection between edge devices and establish end-to-end communication. Examples of networking technologies include wired technologies such as Ethernet, Serial Communication, and USB and wireless technologies such as WiFi, Bluetooth, and ZigBee. Note that message routing among interconnected controllers will also be handled at the network level.

2) Syntactic Level: The syntactic level deals with the diversity in the formats of the messages to be exchanged between edge devices. Examples of message formats are XML, JSON, and CSV. A controller device could receive the message in one format and after translation, could send the message in another format depending on the sender and receiver. The syntactic level clearly works on top of the network level.

3) Semantic Level: The semantic level deals with the ambiguity in the meanings of the same word in various messages from various devices operating in various contexts. A word "minute" could mean a time unit for a clock device but could mean an angle unit for an angular measuring device, and this ambiguity is to be resolved. The semantic level works with the messages and so is on top of the syntactic level.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

4) Organizational Level: The organisation level deals with heterogeneity of platforms, and domains at the organisation level. A smart building system within a smart city system might need an interoperable environment to deal with varieties of platforms in use. In a cross-domain scenario, a smart agriculture farming system might need to communicate with a smart supply chain management system. The organisational level works on top of the semantic level.

Our interoperability model has some similarities with an existing interoperability model [11]. For example, both models use the terms "syntactic" and "semantic". However, our model views the semantic level on top of the syntactic level within a hierarchy, while that model casts the terms at a horizontal level and does not clearly mention their interrelations. The network interoperability in the other model, besides message routing, addresses resource optimization, security, quality of service (QoS), and mobility support. In contrast, the network level in our model deals with establishing communication between devices using various network technologies. Our organisational level deals with cross-platform and cross-domain aspects while the platform interoperability in the other model mainly deals with data structures, programming languages, and application developments. Moreover, that model also considers low-end and high-end capacity issues for the devices as it is generic for IoT. In contrast, our model specifically focuses on resource-constrained devices at the edge level.

C. Proposed Layer-Based Solution

To implement the proposed edge interoperability model, we further propose a layer-based solution as shown in Fig. 5. The proposed solution has five layers: *physical*, *profile*, *registration*, *service*, and *control* layer. We provide further details.

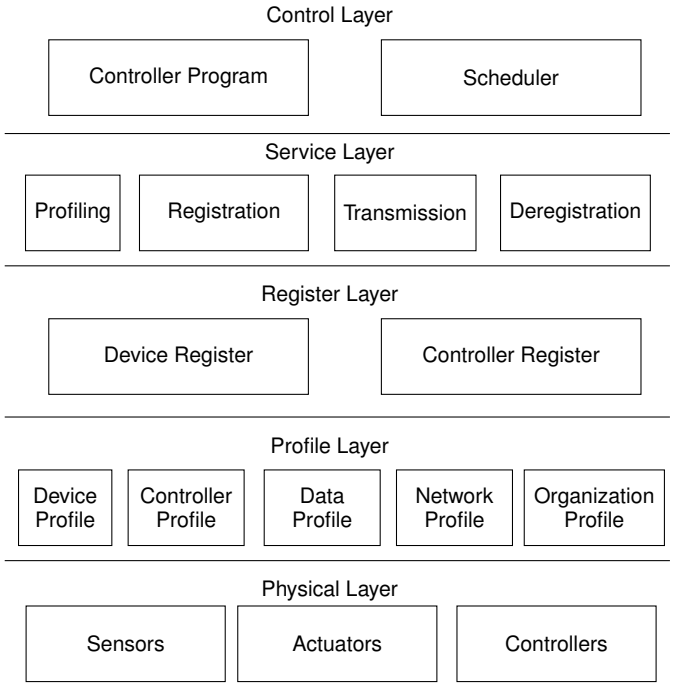


Fig. 5. The proposed layered solution for IoT interoperability

1) Physical Layer: This layer holds all physical devices such as sensors, actuators, and controllers. Sensors detect and measure physical or environmental parameters and convert them into electrical signals or digital data. Examples of sensors are temperature sensor, motion sensor, and humidity sensor. Actuators convert electrical signals or digital commands into physical action. Examples of actuators are electric Motor, servo Motor, and light-emitting diode (LED). We consider microcontrollers to be used as controllers in our interoperability solution. Microcontrollers are resource-constrained edge devices with limited resources and limited computation power. Examples of microcontrollers are: arduino uno, arduino nano, arduino mega, raspberry pi, esp8266, and esp32.

2) Profile Layer: This layer holds various types of profiles: device profile, controller profile, data profile, network profile and organization profile. The profiles hold specific *static* information about configuration parameters of the devices, the controllers, the data formats, the network protocols, and the organisations. Note that profiles are created for types (e.g. temperature sensor). Multiple instance (e.g. individual temperature sensor units) could be of the same type and thus could share the profile. The information in the profiles are later used to register a device and a controller and also by the service layer in creating an interoperable environment. Considering storage requirement and anticipated read-write flexibility level, the profile information of the devices will be stored in non-volatile memory, such as electrically erasable programmable read-only memory (EEPROM).

a) **Device Profile:** Example parameters for device profiles are given in Table I. Moreover, example parameter values for an example device are shown in Table II. Note that the formula parameters in the device profiles are for convenience represented using postfix notations.

TABLE I
EXAMPLE PARAMETERS FOR DEVICE PROFILES

Features	Parameters	Descriptions
Primary	TypeId	Unique identifier
	TypeName	String
	Category	Actuator/Sensor/Controller
	SubCategory	Temperature, Humidity, Pressure, Light, ...
Init	TransmissionMode	Read, Write, Both, ...
	Connectivity	Wired, Wireless
Read	DataType	Integer/Float/Boolean/...
	DataLength	Integer
	MeasuringUnit	°C, ...
	NeedFormula	Yes/No
	Formula	Conversion formula e.g. in postfix notation
	ReadMethod	Method to read data e.g. ReadData
Write	DataType	Integer/Float/Boolean/...
	DataLength	Integer
	MeasuringUnit	°C, ...
	NeedFormula	Yes/No
	Formula	Conversion formula e.g. in postfix notation
	WriteMethod	Method to write data e.g. WriteData

b) **Controller Profile:** Example parameters for controller profiles are shown in Table III and example parameter values for an example controller are shown in Table IV.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

TABLE II
EXAMPLE PARAMETER VALUES FOR A DEVICE PROFILE

Parameter	Value
TypeID	Type001
TypeName	DHT11
Category	Sensor
SubCategory	Temperature, Humidity
TransmissionMode	Read
Connectivity	Wired
DataType	Float
DataLength	10
MeasuringUnit	°, RH
NeedFormula	No
ReadMethod	ReadData

TABLE III
EXAMPLE PARAMETERS FOR CONTROLLER PROFILES

Features	Parameters	Descriptions
Primary	TypeId	Unique identifier
	TypeName	String
	Category	AVR/ARM/DSP/...
Configuration	Connectivity	Wired, Wireless
	NetworkProfile	Supported networks
	OrganizationProfile	Supported org. platforms

TABLE IV
EXAMPLE PARAMETER VALUES FOR A CONTROLLER PROFILE

Parameter	Value
TypeID	ContType001
TypeName	Arduino Uno
Category	AVR microcontroller
Connectivity	Wired, Wireless
NetworkProfile	SerialCommunication, Bluetooth, WiFi
OrganizationProfile	ArduinoMicrocontroller

c) **Data Profile:** Example parameters for data profiles are shown in Table V and example parameter values for an example data format are shown in Table VI.

TABLE V
EXAMPLE PARAMETERS FOR DATA PROFILES

Features	Parameters	Descriptions
Primary	Id	Unique identifier
	Name	String
	Version	Data format version
FieldMapping	DataType	String, Integer, Date, ...
	Encoding	UTF-8/ASCII...

TABLE VI
EXAMPLE PARAMETER VALUES FOR A DATA PROFILE

Parameter	Value
Id	Format001
Name	JSON
Version	JSON-LD
DataType	String, Integer, Date, ...
Encoding	UTF-8

d) **Network Profile:** Example parameters for network profiles are shown in Table VII and example parameter values for an example network profile are shown in Table VIII.

e) **Organization Profile:** Example parameters for organization profiles are in Table IX and example parameter values for an example organization profile are in Table X.

TABLE VII
EXAMPLE PARAMETERS FOR NETWORK PROFILES

Features	Parameters	Descriptions
Primary	Id	Unique identifier
	Name	String
	Version	Network technology version
Configuration	FrequencyBands	Range of radio frequencies
	RangeCoverage	Distance coverage
	DataTransferRate	Speed of data transmission

TABLE VIII
EXAMPLE PARAMETER VALUES FOR A NETWORK PROFILE

Parameter	Value
Id	Network001
Name	Bluetooth
Version	5.2
FrequencyBands	2.4 GHz
RangeCoverage	100 meters
DataTransferRate	50 Mbps

TABLE IX
EXAMPLE PARAMETERS FOR ORGANIZATION PROFILES

Features	Parameters	Descriptions
Primary	Id	Unique identifier
	Name	String
	Version	Network technology version
Configuration	CommunicationProtocol	MQTT/HTTP/CoAP...
	Functionalities	Deployment features
	Container	Packaging of application
	AccessControl	Permissions and privileges

TABLE X
EXAMPLE PARAMETER VALUES FOR AN ORGANIZATION PROFILE

Parameter	Value
Id	Org001
Name	Azure IoT Edge
Version	Azure IoT Edge v4.x
CommunicationProtocol	MQTT, HTTPS
Functionalities	Data ingestion, Data processing, and Data analytics
Container	Docker
AccessControl	Azure Active Directory

3) Register Layer: This layer holds information about the instances such as devices and controllers that are active in the interoperable environment. Note that registration information includes some fields that are also in the profiles. This is because profiles could list a number of available options for a device or a controller, while only some of the options might actually be used in a given connection to the environment.

a) **Device Register:** Example parameters for device register are shown in Table XI and example parameter values for an example device register are shown in Table XII.

b) **Controller Register:** Controllers are also required to be registered into the environment. Example parameters for controller register are in Table XIII and example parameter values for an example controller register are in Table XIV. In the parameters, source devices and destination devices are two arrays with each pair of kth elements of the two arrays denoting a communicating pair. For each such kth communicating pair, the condition-actions parameter has its kth entry, which is a postfix expression involving ternary,

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

TABLE XI
EXAMPLE PARAMETERS FOR DEVICE REGISTERS

Features	Parameters	Descriptions
Primary	DeviceId	Unique identifier
	DeviceName	String
Init	TransmissionMode	Read, Write, Both, ...
	Connectivity	Wired, Wireless
	DeviceProfile	Device type profile id
	DataProfile	Data profile id
	NetworkProfile	Network profile id
	OrganizationProfile	Organization profile id
Read	ActualValue	Last read raw value
	ConvertedValue	Last read converted value
Write	ActualTask	Task needs to sent
	ConvertedTask	Converted task to sent

TABLE XII
EXAMPLE PARAMETER VALUES FOR A DEVICE REGISTER

Parameter	Value
DeviceId	DHT11001
DeviceName	DHT11 Device
TransmissionMode	Read
Connectivity	Wired
DeviceProfile	Type001
DataProfile	Format001
NetworkProfile	Serial001
OrganizationProfile	N/A
ActualValue	N/A
ConvertedValue	N/A

logical, and arithmetic operators. Also, for the kth value of the device probe interval parameter keeps the time interval when the controller needs to read data from the kth source device. And, for the kth value of the controller probe interval parameter keeps the time interval when the controller needs to read data from the kth message buffer.

TABLE XIII
EXAMPLE PARAMETERS FOR CONTROLLER REGISTERS

Features	Parameters	Descriptions
Primary	ControllerId	Unique identifier
	ControllerName	String
	ConnectedDevices	List of connected devices
	ConnectedControllers	List of connected controllers
Config-uration	Connectivity	Wired, Wireless
	NetworkProfile	Supported networks
	OrganizationProfile	Supported org. platforms
	SourceDevices	List of devices to read data
	ConditionActions	List of conditions & actions
	DestinationDevices	List of devices to write data
	DeviceProbeIntervals	Read data time intervals
	MessageBuffer	Messages for connected controllers
	ControllerProbeIntervals	Read message time intervals

4) Service Layer: The service layer has four types of services: *profiling*, *registration*, *transmission* and *deregistration*.

a) **Profiling:** The first step in establishing an interoperable environment is creating profiles for devices, controllers, data, network technologies, and organizational platforms. These profiles hold profile parameters and their values and are stored in non-volatile memory to ensure they remain accessible and not easily erased or removed.

b) **Registration:** The second step is the registration of the devices and controllers using the device register and the controller register. The registration process ensures unique identification (including authentication) and registration is required before any operation. The registration layer basically handles the dynamic aspects of the interoperable environment and it uses the static information from the profiles. Nevertheless, when a device is disconnected from the environment, its registration information will no longer be retained in random access memory (RAM). Consequently, if the device needs to reconnect to the environment later, it will be treated as a new device. The decision of not keeping device registration information in the volatile memory after disconnection is driven by the fact that the controller devices have limited memory resources. To avoid memory constraints and ensure optimal performance, it is not practical to maintain the registration data for disconnected devices. Also, plugging in and plugging out of devices are not very frequent in edge networks because they are mostly set up in fixed locations and usually for long duration. Nevertheless, the registration process creates device and controller registers with their respective parameters values.

TABLE XIV
EXAMPLE PARAMETER VALUES FOR A CONTROLLER REGISTER WHICH IS CONNECTED WITH A SENSOR AND AN ACTUATOR

Parameter	Value
ControllerId	Arduino001
ControllerName	Arduino Uno
ConnectedDevices	Arduino001-DHT11001
	Arduino001-LED001
ConnectedControllers	Arduino002
Connectivity	Wired
NetworkProfile	Serial001
OrganizationProfile	ArduinoMicro001
SourceDevices	(Arduino001-DHT11001)
ConditionActions	(? $>$ Value $30°$) on null
DestinationDevices	(Arduino001-LED001)
DeviceProbeIntervals	(00:00:05)
MessageBuffer	NULL
ControllerProbeIntervals	(00:00:05)

c) **Transmission:** This step offers core services responsible for communication and data exchange among devices and controllers. The data transmission process is divided into methods, with each method responsible for a particular task [40]. A number of methods are crucial for successful data transmission: ReadData, NeedAction, WriteData, ReadMessage, WriteMessage, NeedRouting, and RouteMessage.

Consider a communication scenario shown in Figure 6. This scenario is actually a part of the controller communication network shown in Fig. 3. It shows a statically elected path between two controllers C1 and C7. Nevertheless, assume that device D1's data will be read by controller C1, and then transmitted through C2 and C5 to C7, and finally a command and/or data will be written to device D2 by controller C7. Below we describe the services needed for such as scenario.

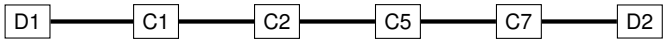


Fig. 6. Static data routing between two devices

1) Controller C1 uses the profile and register parameter values for device D1, communicate with D1, and retrieve data using the ReadData method in Algorithm 1.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

2) After reading the data, controller C1 uses the NeedAction method in Algorithm 1 to check if any action is required against the data read from D1.
3) If an action needs to be taken, then controller C1 checks the destination device which the data is to be sent to using the NeedRouting method in Algorithm 2. In this case, device D2 connected with controller C7 is the destination device. So controller C1 creates a message from the data and the routing information using the WriteMessage method in Algorithm 2, and transmits the message using the RouteMessage method in Algorithm 2. If device D2 was directly connected to controller C1, then no message transmission was needed.
4) Each intermediate controller such as C2 and C5 on the statically elected path from C1 to C7 performs reading of the message using the ReadMessage and rerouting the message using the RouteMessage method.
5) When controller C7 gets the message, it reads device D2's profile and register parameter values along with relevant network and organization profiles. Finally, controller C7 generates commands and/or data and writes to device D2 using the WriteData method in Algorithm 1. Note that if device D2 was directly connected to controller C1, then C1 would have acted like controller C7 particularly for writing data to device D2.

Below we describe the service methods mentioned above.

The ReadData method in Algorithm 1 uses register and profile information for the device, the network, and the organisation. Then, it will connect with the device, reads the data, and applies a formula, if any, to get a processed value. Data is read from a device after a fixed interval specified by the value of DeviceProbeIntervals parameter of the controller register.

The NeedAction method in Algorithm 1 could support edge intelligence and so could use (pretrained) simple AI algorithms, for example, rule-based systems such as decision trees or fuzzy logic systems or simple machine learning algorithms. The NeedAction method reads the corresponding entries in the ConditionActions field from the controller register, evaluates the conditions, and determines the actions to be taken by the destination devices. It returns the action that is needed to be taken by the destination device.

The WriteData method in Algorithm 1 uses register and profile information for the device, the network, and the organisation. Then it will connect with the device and write the data after applying the conversion formula, if any.

The NeedRouting method in Algorithm 2, similar to the NeedAction Method, could support edge intelligence and use simple AI algorithms for decision-making. The NeedRouting method checks if the devices are connected to the same controller. If the source device and the destination device are connected to the same controller, the algorithm returns false. Otherwise, the algorithm returns true.

The WriteMessage method gets the action needs to be taken when a message is required to be transmitted over the network because the destination device is not connected to the controller the source device is connected to. The WriteMessage method then creates a message by combining the action needed to be taken by the destination device with the destination device path.

Algorithm 1 Data Related Methods

function ReadData
  Parameter $D \leftarrow$ Source device register
  Get ID of the device from the device register D
  $P \leftarrow$ Device profile
  $N \leftarrow$ Network profile
  $O \leftarrow$ Organization profile
  Read P, N, O and connect with device accordingly
  $V \leftarrow$ Actual data read from the device
  $V \leftarrow$ Formula(V) // converted value
  return the converted value V
end function

function NeedAction
  Parameter $C \leftarrow$ Controller register
  Parameter $D \leftarrow$ Source device register
  Parameter $V \leftarrow$ Value to be checked
  Read ID of the device from the device register D
  Read ConditionActions F from the controller register C for the ID
  Check the value V with condition in F
  if Any action A is needed to be taken **then**
    return Action A
  end if
end function

function WriteData
  Parameter $D \leftarrow$ Destination device register
  Parameter $A \leftarrow$ Action to be sent
  Get ID of the device from the device register D
  $P \leftarrow$ Device profile
  $N \leftarrow$ Network profile
  $O \leftarrow$ Organization profile
  Read P, N, O and connect with device accordingly
  $V \leftarrow$ Formula(A) // conversion
  Send Data (Command+Value) to destination device
  return If data sent successfully
end function

The routing of message is done by using static routing information stored in the fields SourceDevices and DestinationDevices in the controller registers. TABLE XV shows the two fields in the registers of controllers C1, C2, C5, and C7 for the scenario in Fig. 6. For C1 the source device is C1-D1 and destination device is C1-C2-C5-C7-D2. Using this information, C1 will send the message to C2. For C2 the source device is C2-C1-D1 and the destination device is C2-C5-C7-D2. So C2 sends the message to C5. For C5 the source device is C5-C2-C1-D1 and the destination device is C5-C7-D2. C5 thus sends the message to C7. Finally, for C7 the source device is C7-C5-C2-C1-D1 and the destination device is C7-D2. Since the destination device is connected to the controller C7 itself, no further message routing is needed.

Message is sent from one controller to another using the RouteMessage method in Algorithm 2. Controller C1's

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

Algorithm 2 Message Related Methods

function NeedRouting( )
  Parameter $C \leftarrow$ Controller register
  Parameter $D \leftarrow$ Source device register
  Get ID of the device from the device register D
  Get the corresponding destination device path P from DestinationDevices parameter of the controller register C for the ID
  return true if the destination device is connected to another controller else **false**
end function

function WriteMessage( )
  Parameter $C \leftarrow$ Controller register
  Parameter $D \leftarrow$ Source device register
  Parameter $A \leftarrow$ Action to be taken by destination device
  Get ID of the device from the device register D
  Get the corresponding destination device path P from DestinationDevices parameter of the controller register C for the ID
  Create a message M combining Action A and corresponding destination device path P
  return M the message created
end function

function RouteMessage( )
  Parameter $M \leftarrow$ Message that needs to be routed
  Read next controller ID from the message M
  if Controller ID does not match with own ID **then**
    Write message M to the next controller
  end if
end function

function ReadMessage( )
  Parameter $C \leftarrow$ Controller register
  Check for any incoming message in MessageBuffer
  If data is available, read the incoming message
  return M the message read
end function

TABLE XV
SOURCE AND DESTINATION MAPPING FOR DATA ROUTING

Controller	Source Device	Destination Device
C1	C1-D1	C1-C2-C5-C7-D2
C2	C2-C1-D1	C2-C5-C7-D2
C5	C5-C2-C1-D1	C5-C7-D2
C7	C7-C5-C2-C1-D1	C7-D2

RouteMessage method writes a message into the MessageBuffer parameter of controller C2. Next, C2 reads the message from this parameter using its ReadMessage method and writes message into the MessageBuffer parameter of controller C5 using the RouteMessage method. The MessageBuffer parameter of a controller register is designed as an array with a dedicated sections for each connected sender controller. This arrangement helps avoid data conflicts, where simultaneous writing by multiple controllers could result in message overwrites.

The ReadMessage method reads the message from the parameter MessageBuffer after each interval given in the parameter ControllerProbeIntervals of controller register.

We have used edge intelligence for the methods NeedAction and NeedRouting to optimize local decision-making closer to the data source. This approach enhances efficiency by processing data locally, reducing the necessity for continuous data transmission to a centralized system.

d) **Deregistration:** If a device is no longer needed in the interoperable environment, it is deregistered. The deregistration process involves removing the device register information and any relevant data of that device from the controller register.

5) Control Layer: A ControllerProgram and scheduler methods such as InitSchedule, UpdateSchedule, DataReadingScheduled, and MessageReadingScheduled are needed to perform continuous probing of the devices and other controllers to get data and messages and thus initiate subsequent tasks. The scheduler methods are shown in Algorithm 3.

Algorithm 4 shows the program continuously run by the controller. Based on the values in the DeviceProbeIntervals parameter, the controller initialises a schedule for reading data from the source devices. The controller also initialises another schedule for reading messages from controller message buffer based on the values in the ControllerProbeIntervals parameter. When a source device's turn comes as per the DataReadingScheduled method, the program reads data using the ReadData method. Then, the program executes the NeedAction method. If an action is needed, using the NeedRouting method, it further checks whether the destination device is connected to the same controller or a different one. If the destination device is connected to the same controller as the source, the message doesn't need routing. So the controller executes WriteData method to send the data to the destination device. On the other hand, if the destination device is connected to a different controller, the source controller creates a message using the WriteMessage method and routes the message using the RouteMessage method through another controller. Each controller checks its MessageBuffer parameter in the register for new messages at intervals set in the ControllerProbeIntervals parameter and reads the message accordingly using the ReadMessage method . If a message has a destination device connected to the controller, the data in the message is sent to the device using WriteData method. If the message is to go to a destination device connected to another controller, the message is again routed by the controller using the RouteMessage method.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

Algorithm 3 Scheduler Related Methods

```
function InitSchedule( )
    Parameter C ← Controller register
    Read value of DeviceProbeIntervals parameter
    Read value of ControllerProbeIntervals parameter
    Set time interval for scheduled methods
    Initialize necessary variables
    Initialize scheduler
    Schedule methods to run after the specified time interval
end function

function DataReadingScheduled( )
    Parameter C ← Controller register
    Read source device list from SourceDevices parameter
    for each device do
        if the device is scheduled for now then
            return D the source device register
        end if
    end for
end function

function MessageReadingScheduled( )
    Parameter C ← Controller register
    Read controller list from ConnectedControllers
         parameter of controller C
    for each controller do
        if the controller is scheduled for now then
            return C′ the controller register
        end if
    end for
end function

function UpdateSchedule( )
    if it is time to update then
        Update the schedule
    end if
end function
```

Algorithm 4 ControllerProgram

```
C: the controller executiong this function
InitSchedule( )
while true do
    D_s ←DataReadingScheduled(C)
    V ← ReadData(D_s)
    A ← NeedAction(C, D_s, V)
    D_d ← Destination device register for D_s
    if NeedRouting(C, D_s) then
        M ←WriteMessage(C, D_s, A)
        RouteMessage(M)
    else
        WriteData(D_d, A)
    end if
    C′ ←MessageReadingScheduled(C)
    M ←ReadMessage(C′)
    if NeedRouting(C, D_s) then
        RouteMessage(M)
    else
        WriteData(D_d, A)
    end if
    UpdateSchedule( )
end while
```

D. Mapping of Interoperability Levels

The network level interoperability is achieved with the network profile parameters. The syntactic level interoperability is achieved with data profile parameters. The semantic level interoperability is achieved with device profile parameters. The organizational level interoperability is achieved with organization profile parameters. These mappings between the proposed interoperability levels and the proposed solution layers are listed in Table XVI.

TABLE XVI
MAPPING OF INTEROPERABILITY LEVELS WITH THE COMPONENTS OF THE PROPOSED SOLUTION

Interoperability level	Profile
Organizational	Organization Profile
Semantic	Device Profile
Syntactic	Data Profile
Network	Network Profile

V. EVALUATION

In TABLE XVII, we compare our work with existing work on IoT interoperability. We see that most of the work is based on cloud based architectures. The predominant challenge with adopting a cloud-based architecture to address IoT interoperability lies in the potential for latency in responses, the susceptibility to network or connection problems, and the substantial costs associated with setting up intricate cloud infrastructures. Consequently, achieving seamless IoT interoperability using a cloud-based approach can prove to be complex to realize and incorporate effectively. This situation has led to its limited implementation and adoption due to the aforementioned constraints, resulting in its reduced popularity and limited widespread usage.

As we can see from TABLE XVII, four works [14], [15], [39], [47] support edge devices. Among these, INTER-IoT [39] uses a hybrid cloud-edge structure and is complex and difficult to use for simple IoT systems. So, we have compared our work with the three others works, although two [14], [15] of those depend also on cloud. To present a comparative study of our work with these three edge based solutions we have used the following criteria:

1) **Efficiency:** The capability of an IoT solution to deliver optimal performance while minimizing the consumption of resources (e.g., CPU, memory, and disk space), as well as enhancing speed and efficiency. It is crucial to note that the efficient utilization of time and resources is crucial in this context, particularly since IoT devices are susceptible to power constraints. Therefore, it is essential to prioritize energy efficiency, which can be accomplished by smartly regulating transmission intervals or maximizing the use of available technologies.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

TABLE XVII
COMPARISON OF OUR WORK WITH EXISTING WORKS

Authors & year	Cloud Based	Edge Based	Domain/Protocol Based	Syntactic/Device	Semantic	Network	Organization
Dave, M. (2020) [41]	✓		✓		✓	✓	✓
Famá, F. (2022) [16]	✓		✓	×	×	✓	✓
Jabbar, S. (2022) [42]	✓		✓	✓	✓	✓	✓
Mavrogiorgou, A. (2019) [43]	✓		✓	✓	✓	×	×
Zarko, I. (2017) [44]	✓			×	✓	×	✓
Van Der Veer, H. (2008) [45]	✓			×	×	✓	×
Bröring, A. (2017) [46]	✓			✓	✓	✓	✓
Palau, C. (2021) [39]	✓	✓	✓	✓	✓	✓	
Han, K. (2020) [15]	✓	✓	✓	✓	✓	✓	
Jazayeri, M. (2012) [14]	✓	✓			✓	✓	✓
Aloi, G. (2017) [47]		✓				✓	✓
This Work		✓		✓	✓	✓	✓

TABLE XVIII
COMPARISON WITH EXISTING EDGE-BASED SOLUTIONS

Authors & year	Efficiency	Mobility	Flexibility	Advantages	Limitations
Jazayeri, M. (2012) [14]	×	✓	×		-supports read data only
Aloi, G. (2017) [47]	×	×	×		-battery needs to be charged frequently -phone needs at one place
Han, K. (2020) [15]	✓	×	✓		-cannot be setup easily as needs cloud configuration
This Work	✓	✓	✓	-complete edge-based solution -easy setup -operational without server setup	-suitable for small network and a limited number of devices

Consequently, there is a requirement for solutions that can minimize the energy consumption.

2) **Mobility:** The ability to quickly and efficiently provide users access to information related to IoT devices.
3) **Flexibility:** Ease of incorporating, altering, or eliminating features of the solution.

We compare our device profile approach with EdgeX Foundry [15]. While both use profiles, there are key differences. EdgeX Foundry's profiles mainly outline device attributes, while ours cover devices, controllers, data, network technologies, and organizational platforms. We use profiles for connection and communication with devices, whereas EdgeX Foundry uses functions for various networks without relying on profiles. EdgeX Foundry lacks mobility because of its cloud based architecture. Another solution TinySOS [14] is not efficient and flexible. Devices themselves must send data to a central database in a specific format. TinySOS only receives data from sensors and lacks writing data to an actuator capability. Both EdgeX Foundry and TinySOS use cloud-based architecture for interoperability, but our model achieves it without server setups.

The work presented in [47] is not efficient because while using the solution, the smartphone battery charge drops significantly if the apps is continuously running. For an interoperable solution it is required for the solution to be running all the time. Moreover, that solution is not effective because in order to keep the system running the smartphone needs to be nearby to maintain connectivity of the IoT devices. The solution is mostly feasible if a dedicated smartphone is used only for this interoperability gateway. Furthermore, the solution is not flexible because, before each time the apps will be installed on a new smartphone, it has to be changed according to the devices that will be connected with the smartphone. The comparison of these solutions with ours is presented in Table XVIII.

Our model is suitable for small network having a small number of devices and for simple IoT systems. Although, the network could be extended with additional controllers, our model is not designed to deal with large networks having large numbers of devices and a substantial volume data transfer. Also, our solution suits scenarios where device registrations and removals are infrequent. We avoid server setup and cloud data transmission for decision-making and thus use resource constrained controllers. Due to our use of resource-constrained controllers, device information is not retained after deregistration. If a device wants to register again, it is treated as a new registration. We ensure efficiency and mobility by using microcontrollers which are small and low powered. Also, our model uses profiles and interoperability levels to achieve flexibility.

For evaluation, let us consider a use case scenario depicted in Fig. 7. When a person gets into the house, our solution can be designed to detect motion using a motion sensor, which is accessible via 'Bluetooth'. The motion detection could automatically turn on the light, which is accessible via 'WiFi' working on 'Contiki'. When the motion sensor detects motion or presence of someone, it will also send a signal to check the temperature from a temperature sensor, which is accessible via

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

'Bluetooth' working on 'TinyOS' and will automatically turn on the ac, which is accessible via 'WiFi' with user's preferred temperature set beforehand. The user can also install a camera and program to turn on the TV, which is accessible via 'WiFi' working on 'Android', if the motion sensor detects any motion or the camera detects any presence of human. Again, if the room is found to be empty for a specified period, the lights or the climate control can be automatically turned off to save energy. The user can also program the solution to monitor their swimming pool conditions in real time.

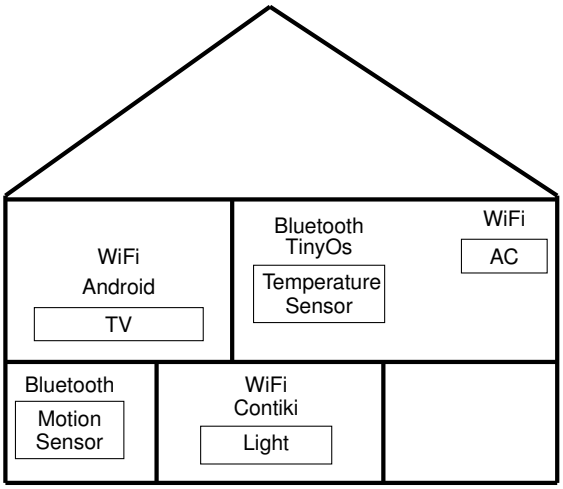


Fig. 7. Use case scenario

To use our solution in the above use case, the user can buy any cheap sensors and actuators, and design the system. We need a controller and register the devices to our solution. From TABLE XVII we see that three other solutions [14], [15], [39] need to have cloud server setup in order to implement this use case. However, it is not viable to setup a cloud based architecture for this simple scenario. In order to implement another solution [47], a dedicated smart phone is to be used and it consumes too much energy and is to be charged frequently (see TABLE XVIII). So these four solutions for edge devices are not viable for this use case scenario.

VI. Conclusion

The Internet of Things (IoT) is a global network of devices that are connected to the internet and can communicate with each other. IoT devices are used in a variety of domains, including home automation, smart cities, supply chains, healthcare, agriculture, and environmental monitoring. However, IoT faces interoperability challenges due to the diverse communication protocols used by different devices. In this paper, we have presented a hierarchical interoperability model and a layer-based solution for seamless communication among resource-constrained IoT edge devices. Unlike existing cloud-based solutions, our proposed solution empowers local data processing and decision-making at the edge level.The proposed solution integrates IoT edge devices that have the potential to facilitate existing and future IoT developments. Comparing with existing edge-based solutions of interoperability, our proposed solution is better in terms of *efficiency*, *mobility*, and *flexibility*. In future, we plan to implement our solution considering some general use cases.

References

[1] K. Ashton *et al.*, "That 'internet of things' thing," *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[2] I.-T. S. Group *et al.*, "New itu standards define the internet of things and provide the blueprints for its development," 2012.

[3] A. Rejeb, K. Rejeb, S. Simske, H. Treiblmaier, and S. Zailani, "The big picture on the internet of things and the smart city: a review of what we know and what we need to know," *Internet of Things*, vol. 19, p. 100565, 2022.

[4] M. M. Dhanvijay and S. C. Patil, "Internet of things: A survey of enabling technologies in healthcare and its applications," *Computer Networks*, vol. 153, pp. 113–131, 2019.

[5] B. B. Sinha and R. Dhanalakshmi, "Recent advancements and challenges of internet of things in smart agriculture: A survey," *Future Generation Computer Systems*, vol. 126, pp. 169–184, 2022.

[6] B. Rawlins, J. Trevathan, and A. Sattar, "Embedded fog models for remote aquatic environmental monitoring," *Internet of Things*, vol. 20, p. 100621, 2022.

[7] H. yliopisto. Department of Computer Science, F. Eliassen, and J. Veijalainen, *A functional approach to information system interoperability*, 1988.

[8] A. Čolaković and M. Hadžialić, "Internet of things (IoT): A review of enabling technologies, challenges, and open research issues," *Computer networks*, vol. 144, pp. 17–39, 2018.

[9] S. Sinche, D. Raposo, N. Armando, A. Rodrigues, F. Boavida, V. Pereira, and J. S. Silva, "A survey of IoT management protocols and frameworks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 2, pp. 1168–1190, 2019.

[10] E. Lee, Y.-D. Seo, S.-R. Oh, and Y.-G. Kim, "A survey on standards for interoperability and security in the internet of things," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 1020–1047, 2021.

[11] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in internet of things: Taxonomies and open challenges," *Mobile networks and applications*, vol. 24, pp. 796–809, 2019.

[12] C. Srinivasan, B. Rajesh, P. Saikalyan, K. Premsagar, and E. S. Yadav, "A review on the different types of internet of things (IoT)," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 11, no. 1, pp. 154–158, 2019.

[13] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, T. Jiang, J. Crowcroft, and P. Hui, "Edge intelligence: Empowering intelligence to the edge of network," *Proceedings of the IEEE*, vol. 109, no. 11, pp. 1778–1837, 2021.

[14] M. A. Jazayeri, C.-Y. Huang, and S. H. Liang, "Tinysos: Design and implementation of interoperable and tiny web service for the internet of things," in *Proceedings of the First ACM SIGSPATIAL Workshop on Sensor Web Enablement*, 2012, pp. 39–46.

[15] K. Han, Y. Duan, R. Jin, Z. Ma, H. Rong, and X. Cai, "Open framework of gateway monitoring system for internet of things in edge computing," in *2020 IEEE 39th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2020, pp. 1–5.

[16] F. Famá, J. N. Faria, and D. Portugal, "An IoT-based interoperable architecture for wireless biomonitoring of patients with sensor patches," *Internet of Things*, vol. 19, p. 100547, 2022.

[17] C. Bonsignori, C. Puliafito, A. Virdis, E. Mingozzi, and G. Iannaccone, "Integrating mobile IoT devices into the arrowhead framework using web of things," in *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2022, pp. 1–6.

[18] C. Pereira, A. Pinto, A. Aguiar, P. Rocha, F. Santiago, and J. Sousa, "IoT interoperability for actuating applications through standardised m2m communications," in *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. IEEE, 2016, pp. 1–6.

[19] L. Sciullo, F. Montori, A. Trotta, M. Di Felice, and T. S. Cinotti, "Discovering web things as services within the arrowhead framework," in *2020 IEEE Conference on Industrial Cyberphysical Systems (ICPS)*, vol. 1. IEEE, 2020, pp. 571–576.

[20] T. Clark and R. Jones, "Organisational interoperability maturity model for c2," in *Proceedings of the 1999 Command and Control Research and Technology Symposium*, vol. 29. Citeseer, 1999.

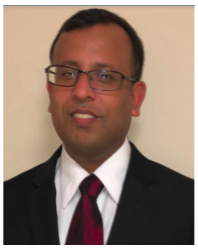
[21] C. A. W. Group *et al.*, "Levels of information systems interoperability," *US DoD, http://www. c3i. osd. mil/org/cio/i3*, 1998.

Authorized licensed use limited to: Griffith University. Downloaded on Januay 02,2024 at 00:42:14 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in IEEE Internet of Things Journal. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/JIOT.2023.3340298

[22] A. Tolk and J. A. Muguira, "The levels of conceptual interoperability model," in *Proceedings of the 2003 fall simulation interoperability workshop*, vol. 7. Citeseer, 2003, pp. 1–11.
[23] C. Turnitsa, "Extending the levels of conceptual interoperability model," in *Proceedings IEEE summer computer simulation conference, IEEE CS Press*, 2005.
[24] L. S. Winters, M. M. Gorman, and A. Tolk, "Next generation data interoperability: It's all about the metadata," in *IEEE Fall Simulation Interoperability Workshop*, 2006.
[25] A. Gluhak, O. Vermesan, R. Bahr, F. Clari, T. Maria, T. Delgado, A. Hoeer, F. Bösenberg, M. Senigalliesi, and V. Barchett, "Bdeliverable d03. 01 report on IoT platform activities-unify-IoT," 2016.
[26] E. S. Pramukantoro and H. Anwari, "An event-based middleware for syntactical interoperability in internet of things," *International Journal of electrical and computer engineering*, vol. 8, no. 5, p. 3784, 2018.
[27] E. S. Pramukantoro, F. A. Bakhtiar, B. Aji, and R. Pratama, "Middleware for network interoperability in IoT," in *2018 5th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*. IEEE, 2018, pp. 499–502.
[28] J. Hoebeke, E. De Poorter, S. Bouckaert, I. Moerman, and P. Demeester, "Managed ecosystems of networked objects," *Wireless Personal Communications*, vol. 58, no. 1, pp. 125–143, 2011.
[29] I. Ishaq, J. Hoebeke, I. Moerman, and P. Demeester, "Internet of things virtual networks: Bringing network virtualization to resource-constrained devices," in *2012 IEEE International Conference on Green Computing and Communications*. IEEE, 2012, pp. 293–300.
[30] J. Rodríguez-Molina, J.-F. Martínez, P. Castillejo, and L. López, "Combining wireless sensor networks and semantic middleware for an internet of things-based sportsman/woman monitoring application," *Sensors*, vol. 13, no. 2, pp. 1787–1835, 2013.
[31] S. Cox, "Observations and measurements-xml implementation. version 2.0." 2011.
[32] A. Na and M. Priest, "Sensor observation service. version 1.0." 2007.
[33] M. Botts and A. Robin, "Opengis® sensor model language (sensorml) implementation specification. version 1.0. 0." 2007.
[34] I. Simonis and P. C. Dibner, "Opengis sensor planning service implementation specification," *Implementation specification OGC*, pp. 1–21, 2007.
[35] S. Liang and T. Khalafbeigi, "Ogc sensorthings api part 2–tasking core, version 1.0." 2019.
[36] T. O'Reilly *et al.*, "Ogc® puck protocol standard version 1.4," *Open Geospatial Consortium, OGC Encoding Standard OGC*, 2010.
[37] H. Rahman and M. I. Hussain, "Fog-based semantic model for supporting interoperability in IoT," *IET Communications*, vol. 13, no. 11, pp. 1651–1661, 2019.
[38] B. Negash, T. Westerlund, and H. Tenhunen, "Towards an interoperable internet of things through a web of virtual things at the fog layer," *Future Generation Computer Systems*, vol. 91, pp. 96–107, 2019.
[39] C. E. Palau, G. Fortino, M. Montesinos, P. Giménez, G. Markarian, V. Castay, F. Fuart, W. Pawłowski, M. Mortara, A. Bassi *et al.*, "Introduction to interoperability for heterogeneous iot platforms," *Interoperability of Heterogeneous IoT Platforms: A Layered Approach*, pp. 1–26, 2021.
[40] L. Sun, Y. Li, and R. A. Memon, "An open IoT framework based on microservices architecture," *China Communications*, vol. 14, no. 2, pp. 154–162, 2017.
[41] M. Dave, M. Patel, J. Doshi, and H. Arolkar, "Ponte message broker bridge configuration using mqtt and coap protocol for interoperability of IoT," in *Computing Science, Communication and Security: First International Conference, COMS2 2020, Gujarat, India, March 26–27, 2020, Revised Selected Papers 1*. Springer, 2020, pp. 184–195.
[42] S. Jabbar, F. Ullah, S. Khalid, M. Khan, and K. Han, "Semantic interoperability in heterogeneous IoT infrastructure for healthcare," *Wireless Communications and Mobile Computing*, vol. 2017, 2017.
[43] A. Mavrogiorgou, A. Kiourtis, K. Perakis, S. Pitsios, and D. Kyriazis, "IoT in healthcare: achieving interoperability of high-quality data acquired by IoT medical devices," *Sensors*, vol. 19, no. 9, p. 1978, 2019.
[44] I. P. Zarko, S. Soursos, I. Gojmerac, E. G. Ostermann, G. Insolvibile, M. Plociennik, P. Reichl, and G. Bianchi, "Towards an IoT framework for semantic and organizational interoperability," in *2017 Global Internet of Things Summit (GIoTS)*. IEEE, 2017, pp. 1–6.
[45] H. Van Der Veer and A. Wiles, "Achieving technical interoperability," *European telecommunications standards institute*, 2008.
[46] A. Bröring, S. Schmid, C.-K. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente, "Enabling IoT ecosystems through platform interoperability," *IEEE software*, vol. 34, no. 1, pp. 54–61, 2017.
[47] G. Aloi, G. Caliciuri, G. Fortino, R. Gravina, P. Pace, W. Russo, and C. Savaglio, "Enabling IoT interoperability through opportunistic smartphone-based mobile gateways," *Journal of Network and Computer Applications*, vol. 81, pp. 74–84, 2017.

TANZIMA AZAD is currently pursuing her Ph.D. degree with the School of Information and Communication Technology, Griffith University, Brisbane, Australia. She received her MSc degree in Computer Science and Engineering from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2018 and BSc degree in Computer Science and Engineering from Military Institute of Science and Technology (MIST), Dhaka, Bangladesh in 2010. Her research interests include internet of things, machine learning, spatio temporal data, and crowdsourced data.



M. A. HAKIM NEWTON received the B.Sc.Eng. and M.Sc.Eng. degrees from the Bangladesh University of Engineering and Technology (BUET), and the Ph.D. degree from Strathclyde University, U.K. He is a Lecturer with the School of Information and Physical Sciences, The University of Newcastle, Australia. He is also an Adjunct Senior Research Fellow with the Institute for Integrated and Intelligent Systems (IIIS), Griffith University, Australia. He was a Research Engineer with the National ICT Australia (NICTA). His research interests include artificial intelligence, intelligent search, machine learning, and bioinformatics.

JARROD TREVATHAN is an Associate Professor with the School of Information and Communication Technology, Griffith University, Australia. He is an expert in affordable environmental monitoring technologies and ecommerce security and fraud algorithms. He has won significant community awards/grants and prestigious Australian Research Council (ARC) funding particularly for water resource monitoring and disaster management.

ABDUL SATTAR is a Professor with the School of Information and Communication Technology, Griffith University, Australia. He was the Founding Director of the Institute for Integrated and Intelligent Systems, at Griffith University. He was also the Education Director with the Queensland Research Laboratory (QRL), National ICT Australia (NICTA). He won a number of ARC discovery grants and international awards for his work in artificial intelligence.