Received July 1, 2020, accepted July 11, 2020, date of publication July 16, 2020, date of current version July 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3009783

Towards Medical Data Interoperability Through Collaboration of Healthcare Devices

ABDUL JALEEL[1], TAYYEB MAHMOOD[2], MUHAMMAD AWAIS HASSAN[3], GULSHAN BANO[4], AND SYED KHALDOON KHURSHID[3]

[1]Department of Computer Science, Rachna College of Engineering & Technology (RCET), University of Engineering and Technology, Lahore, Lahore 54890, Pakistan
[2]Department of Electrical Engineering, Rachna College of Engineering & Technology (RCET), University of Engineering and Technology, Lahore, Lahore 54890, Pakistan
[3]Department of Computer Science, University of Engineering and Technology, Lahore, Lahore 54890, Pakistan
[4]Department of Information Technology, University of Sialkot, Sialkot 51040, Pakistan

Corresponding author: Abdul Jaleel (abduljaleel@uet.edu.pk)

ABSTRACT In the era of smart devices and connected neighborhoods, the ubiquitous monitoring and care of patients are possible with the Internet of Medical Things (IoMT). Smart healthcare devices may serve their purpose well when they are able to share patient's data with each other. However, data formats vary widely across vendors, rendering these devices not interoperable. Recent solutions mostly rely on cloud services where a source device uploads the data, and the sink devices download it conforming to their own native formats. However, the quality of service is expected to deteriorate in a cloud processing regime with inherent network delays and traffic congestion, and the real-time data acquisition and manipulation is, therefore, not possible. This article presents MeDIC, a framework of Medical Data Interoperability through Collaboration of healthcare devices. MeDIC improves over a cloud-based IoMT by utilizing translation resources at the network edge, with its probing and translating agents. The probing agents maintain a capability list of MeDIC devices within a local network and enable one MeDIC device to request data conversion from another device when the former is not capable of this conversion by itself. The translating agent of the later then converts the data into the required format and returns it to the former. These novel agents allow IoMT devices to share their redundant computing resources for data translations in order to minimize cloud accesses. Legacy devices are supported through MeDIC-enabled, fog resource managers. We evaluate MeDIC in four use cases with rigorous simulations, which prove that this collaborative framework not only reduces the uplink traffic but also improves the response time, which is critical in real-time medical applications.

INDEX TERMS IoT, IoMT, medical things, fog resource manager, edge resource discovery, continuous patient monitoring, healthcare 4.0, interoperability hub, distributed computing, smart ICU, syntactic interoperability.

I. INTRODUCTION

The continuous advancements in information and communication technologies in the past decade have given rise to the ubiquitous Internet of Things (IoT) [1]. Of its many use cases, the Internet of Medical Things (IoMT) is an evolution in which healthcare devices and systems are connected to communicate and interact with each other [2]. In a smart city, the integration of IoT into medical devices is expected to improve the quality and effectiveness of services provided to patients, particularly those who are having chronic diseases and who require constant care [3]. With the help of Body Area Networks, IoMT allows continuous monitoring of vital physiological functions of otherwise a healthy person so that an illness may get diagnosed, and appropriate actions are taken immediately [4]. This is particularly important in pandemic situations like recent COVID-19 [5], where our most advanced healthcare systems are also under stress, including medical personals, as well as support systems [6], [7]. The need for a remote, autonomous and ubiquitous IoMT framework is greater than ever before [8].

The integration of smart sensors and controllers with the internet has transformed the so-called cyber-physical systems into the Internet of Things and has recently become

The associate editor coordinating the review of this manuscript and approving it for publication was Muhammad Zakarya.

TABLE 1. Comparison of existing researches.

Research work	Normalization Player	IoMT Application	Interoperability Approach
Clarke et al. [25]	Data Source	Personal Health Devices	Zigbee and IEEE 11073-based
Lubamba and Bagula [13]	Data Sink	Cyber Healthcare Cloud	HL7-CDA Standard-based
Reilent et al. [30]	Central Resource	Personal Health Records	HL7 v3 protocol
Alzghoul et al. [19]	Web-Centric Middleware	Electronic Health Records	HL7 CDA
Khalique et al. [31]	Central Resource	Nation-wide Health Repository	HL7-based
Jabbar et al. [24]	Centralized Service	Heterogeneous IoT devices	Semantic Interoperability
Garai and Adamko [27]	Cloud based	Consumer Electronics Interoperability	Syntactic, Semantic
Boutros-Saikali et al. [21]	Cloud based	IoMT Platform	Scriptr.io service
Roehrs et al. [18]	Middleware	Integration of Personal Health Record	Semantic Interoperability
Georgi et al. [32]	Middleware	Data access from Health Sensors	Sementic Interoperability
Ivanov et al. [33]	Middleware	Connectivity Platform for IoMT	OpenICE-Lite
Moustafa et al. [26]	Edge-Cloud	Remote Monitoring	Iotivity Software Stack
Derhamy et al. [1]	Cloud/Fog Server,	Industrial Internet of Things	Multiprotocol Translator
Negash et al. [34]	Cloud and Fog	IoT through Web of Virtual Things	Technical, Syntactic, Semantic
Gia et al. [35]	Fog-based	Health-care IoT System	No Interoperability
Rahman and Hussain [36]	Fog-based	Interoperability in IoT	Semantic
Ahmed et al. [37]	Multilayer, Modular	Interoperability IoT Hub	Syntactic, Semantic, Communication
Zarko et al. [38]	IoT Federation	IoT Framework	Sementic, Organizational
Cintuglu et al. [39]	Hierarchical	Hierarchical Microgrid Control	Multiagent based

the driving force behind the fourth industrial revolution, dubbed as Industry 4.0 [9]. In this paradigm, healthcare services are also witnessing digital transformations through the inductions of smart equipment, for example, Automatic Infusion Systems and real-time Dialysis Dose Monitoring [10], in smart ambulances and hospitals. These healthcare devices are generally interoperable and may be connected to form a scalable IoMT when all the devices are sourced from the same vendor, with recommended network infrastructure, including cloud services. This dependency, however, presents a bottleneck in almost all practical use cases because of a diverse range of medical sensors and instruments. The continuous glucose monitoring (CGM) device from one vendor, for example, measures blood glucose levels at regular intervals [11]. This invaluable data, may not be consumable in an insulin pen (IP) by another vendor [12]. Moreover, due to their life-saving nature, the sophisticated medical equipment comes from specialized vendors, conforming to the veracity of national standards, and often implements proprietary protocols [2], [13]. Similarly, consumer-grade medical gadgets also support various IoT protocols and data formats [14]. Therefore, in a practical IoMT, this diversity creates information silos as the devices are not interoperable, hindering the much sought horizontal integration in Industry 4.0 [9]. To help, some vendors are sourcing interoperability devices, like HealthGO Hubs [15] and ELIOT hubs [16], which support a subset of IoMT protocols, and the medical data which they acquire is available through their proprietary, cloud-based services. These hubs are generally limited to vertical integration of healthcare services, including remote patient monitoring and remote configuration (of healthcare devices) and lack a real-time, horizontal collaboration through direct data transactions among these devices [9]. Moreover, their performance may deteriorate under load due to the centralization of resources, evaluated later in this article.

Owing to its utmost importance, IoT interoperability remains the focus of numerous recent researches [14]. Next sections offers a good review of major efforts in the domain of Electronic Health Records (EHR) [13], [17]–[20], IoMT [2], [21], their security and authentication [17], [22], [23] and their interoperability at various layers and levels, including semantic [18], [24], protocol [1], intrinsic [25] and cloud [13], [26], [27] level. However, less focus has been attributed towards data format conflicts among IoMT devices and their data interoperability. When devices generate data in various formats, their conversion requires computing resources. Most of the previous attempts (summarized in Table 1) relied on a central computing resource, predominantly a cloud. As data conversion lies in the critical path of processing, inherent network dependencies of cloud, including bandwidth limits and latency, are detrimental to the medical use cases, which often require high dependability and real-time latencies [28]. Alternatively, fog and edge computing have been proposed to overcome the above-mentioned challenges [29]. Edge computing brings computing resources closer to the data source and sink devices, which can effectively improve the quality of service (QoS). Fog computing, on the other hand, brings essential cloud services downward, at the user network levels. To achieve data interoperability in real-time within an IoMT, these fog and edge computing devices are needed to be interwoven with the help of a collaborative and distributed framework, to enable the horizontal integration in Industry 4.0 [9].

In this research, we propose a novel IoMT framework that pushes the data interoperability towards the network edge where the medical data originates. First, a pyramid of resource management is implemented within fog devices to provide baseline data interoperability to legacy, vendor-specific devices. Second, an interoperability stack is proposed that helps medical devices in sharing their redundant computing resources, for the translations of data transacting horizontally between these devices. This way, the proposed framework effectively implements the Edge-computing within capable medical devices, with the help of its interoperability stack, integrated with the hierarchy of resource managers. The interoperability stack consists of software agents that remain active in the background, and the framework prioritizes and ensures the availability of a device's computing resources to the native application. Therefore, the cloud is accessed only when data translation cannot be commenced locally. The proposed framework is distributed, scalable, and extendable to the other dimensions of IoMT, such as protocol interoperability, and to other applications of IoT in general.

As a summary, we make the following contributions to the domain of data interoperability within an IoMT.

1) We revisit an end-to-end framework of a cloud-based interoperable IoMT in terms of service layers, including authentication, subscription, and publication.
2) We propose a novel framework of Medical Data Interoprability through Collaboration, or MeDIC. The framework enables device-level resource sharing through its Probe and Translation interfaces in an edge computing paradigm. To the best of our knowledge, this work makes the first contribution towards the horizontal integration of the healthcare devices, by solving the interoperability challenges in a truly distributive manner, thus avoiding information silos in Industry 4.0 paradigm.
3) We simulate four real-life use cases of MeDIC in an open-source simulator, *iFogSim* in which MeDIC enables data sharing between connected medical devices, to compare the response latency and uplink traffic related to data interoperability, with and without MeDIC deployments.

The rest of the article is organized as follows. The next section presents the recent researches on IoMT. This review is followed by a narration of IoMT use cases and the motivation behind this work. Section III presents the proposed MeDIC framework and the architecture of its underlying layers, interfaces and agents. Section IV evaluates the proposed framework based on rigorous simulations of MeDIC use cases, followed by simulation results and discussions. Finally, section V concludes the article.

II. BACKGROUND

In this section, we first discuss the related work and then describe the motivation behind our research.

A. RELATED WORK

Interoperability is "the ability of two or more systems or components to exchange information and to use the information that has been exchanged" [31]. The problem of interoperability in Information Technology has been long existed and has been focused at various levels, for example, at a platform level, networking level, syntactic level and semantic level, and in various domains, for example in industrial and healthcare domains [14]. Especially, many efforts were devoted towards the domain of Electronic Health Records (EHR) [13], [17], [31], [41]. For example, HL7 is an Electronic Health Record System which consists of the standards to maintain syntactic and semantic interoperability [13]. There are also other open source and proprietary/legacy standards, for example, openEHR and MIMIC [18], which are conformed by various healthcare providers and equipment vendors.

To address the problem of interoperability while considering various types of IoT application domains and architectures of the IoT, Negash *et al.* [34] presented a holistic abstract model of IoT with three layers including technical, syntactic and semantic, each with a different level of interoperability requirements. The technical level of interoperability deals with hardware and protocol mappings, whereas syntactic and semantic interoperability methods are used for data formats and protocol mappings. A number of standards and protocols are proposed and conformed by healthcare devices, at these various levels. When vendor-specific devices store data in incompatible formats, even the devices operating with the same protocol are unable to exchange data [42]–[44].

1) DATA NORMALIZATION FOR INTEROPERABILITY

For data interoperability, normalization at the data source ensures syntactic and semantic homogeneity. Clarke *et al.* [25] implemented an end-to-end remote monitoring platform based on the IEEE 11073 standards for personal health devices which interchange data in HL7 format. However, all the stack-holders must conform to the same standard to utilize the benefits of normalization-based solutions. When devices are heterogeneous in their data formats, Lubamba and Bagula [13] demonstrated that instead of converting every record in HL7, a process called normalization, conversion at the data sink results in better resource utilization, hence resulting in lower overheads.

However, most of the current solutions to interoperability of heterogeneous formats operated devices are based on normalization (translation from the source into sink formats) at a central resource. Especially, such efforts are shown fruitful in the domain of personal and electronic health record management, where the data is collected and served through a central repository of the data warehouse. Reilent *et al.* [30] proposed an HL7-based normalization for data interoperability in personal health record systems and wellness telecare systems with a nation-wide health-data repository in Estonia, while Alzghoul *et al.* [19] and Khalique *et al.* [31] proposed similar solutions for Jordan and Pakistan, respectively. Jabbar *et al.* [24] introduced semantic interoperability among data collected from syntactically heterogeneous IoMT devices through a set of annotations that are processed by a centralized service at the cloud. Because such systems are not real-time and normalization is required only during archival, central-resource based interoperability makes sense in these domains. However, in the domain of IoT, especially IoMT, a real-time patient monitoring and healthcare needs a hierarchical solution with a central cloud server offering baseline connectivity with intermediate resources near data origination and consumption which enables fast data access [26]. For interoperability, the literature is thus further subdivided into the cloud-, fog- and edge-based solutions.

2) CLOUD-BASED INTEROPERABILITY

Due to their scalability, universal access, and transparent authentication, cloud-based interoperability was a focus of many studies. For example, Garai and Adamko [27] introduced an Open Telemedicine Interoperability Hub, a cloud-oriented framework that determines a centralized information-flow and interprets, maps, converts, and aggregates the incoming data for syntactic and semantic interoperability. Boutros-Saikali *et al.* [21] presented a cloud-based IoMT platform for the development of healthcare monitoring applications, solving the problems of interoperability, integration and security. The authors extended Scriptr.io, a cloud-based service to implement an interoperability layer that normalizes the data into an internal format for later consumption. A comprehensive review of the existing research works, related to data interoperability in IoMT devices, and their comparison is given in Table 1. The use of centralized and cloud-based approaches brings an advantage of scalability, but these approaches face drawbacks of processing cost and network delays, which are detrimental to real-time patient data sharing among healthcare devices and facilities.

There are some efforts made to shift the integration complexity to the central middleware stack and web services. Roehrs *et al.* [18] applied semantic interoperability through a middleware for the integration of Personal Health Record, whereas Georgi *et al.* [32] proposed a middleware to overcome the heterogeneity of communication protocol implemented by data sensors which smooth the retrieve of data from the sensors. Ivanov *et al.* [33] introduced a middleware, OpenICE-Lite, to provide connectivity platform for IoMT offering real-time communication and coordination of medical devices for accessing and analyzing the medical data.

3) FOG- AND EDGE-BASED INTEROPERABILITY

The Fog/Edge-based approaches work in a distributed fashion and bring the benefits of lesser delays and reduced cloud processing costs. Moustafa *et al.* [26] argued that interoperability through the cloud, despite its technical merits, is less useful in real-time applications and instead introduced an IoT gateway that implements interoperability at the edge of the network, to save bandwidth and to improve response time. Derhamy *et al.* [1] proposed Arrowhead, a dedicated and on-demand protocol-translation service. It is located within local clouds of IoT as a Fog server and reduces the latency related to the translation of supported formats. Negash *et al.* [34] proposed a Web of Virtual Things server, that is deployed as a Fog hub or at the cloud and provides interoperability through normalization for syntactic integration of devices with a REST-based API. The utilization of a web of virtual things for interoperability opens a path for an integrated and scalable IoT. Gia *et al.* [35] presented a low-cost IoMT system that uses fog-layer to operate sensor nodes in an energy-efficient way. Rahman and Hussain [36] presented a fog-based semantic interoperability solution for heterogeneous IoT devices to overcome the bottleneck of longer distances intrinsic in the cloud-based solution of interoperability.

Finally, there are some other approaches towards interoperability that exploit the hierarchical levels of connectivity. Ahmed *et al.* [37] presented a multilayer, modular IoT Hub that offers syntactic, semantic, communication interoperability. Zarko *et al.* [38] proposed an architecture to provide interoperability that was built around the hierarchical stack of application, cloud, smart space, and device domain for providing connectivity between different resources, like sensors, actuators, gateway, and cloud. Cintuglu *et al.* [39] developed real-time test cases for the interoperability of the multi-agent system. Micrograms were installed for standardizing the frameworks of interoperability. This model elaborates on a new approach based on the multi-agent system.

From this review, it is evident that a consensus exists on shifting compute-intensive operations from the cloud level to the fog and edge levels. Inevitably, data translation (or normalization) for interoperability must also be pushed closer to the data source and sink devices in an IoMT because it is the most resource-hungry operation [45]. Moreover, the proposed solutions lack horizontal integration of computing resources, a key enabler of Industry 4.0, and a horizontally collaborative framework of computations in a heterogeneous IoT (IoMT) is yet to be explored. Next, we present how medical use cases also persuade towards this horizontal integration and data sharing of smart medical devices, and how a framework like MeDIC may help in these use cases.

B. MOTIVATION

A medical application (App) may connect a healthcare provider to his patient's IoMT devices, a scenario illustrated in Figure 1(a). Here, the patient's smartphone serves as a gateway to the respective cloud. It supports requesting and sharing of data, but data translations are to be supported by the cloud to resolve heterogeneous data formats; otherwise, the data remains useless. By keeping the problem in perspective, this work focuses on IoMT in four real-life medical use case depicted in Figure 1(b) and elaborated below.

1) INSIDE AN AMBULANCE

A smart ambulance is on its way to the hospital, carrying a patient that requires medical assistance, (lower right of Figure1(b)). Conventionally, a paramedic in the van is communicating with the experts at the hospital and executes standard medical procedures according to the received instructions. When IoMT is deployed in this smart ambulance, the doctor is able to monitor the patient's vital signs through healthcare devices, for example, Glucose Level Monitor (GLM), Heart Beat Rate (HBR) monitor, Oxygen Level/Flow and Blood Pressure Monitor (BPM), etc. These devices connect to a medical cloud through an interoperability hub with remote patient monitoring support because they generate data in diverse formats, including JSON (JavaScript Object Notation), XML (Extensible Markup Language), CSV (Comma-Separated Values) and binary formats, respectively.

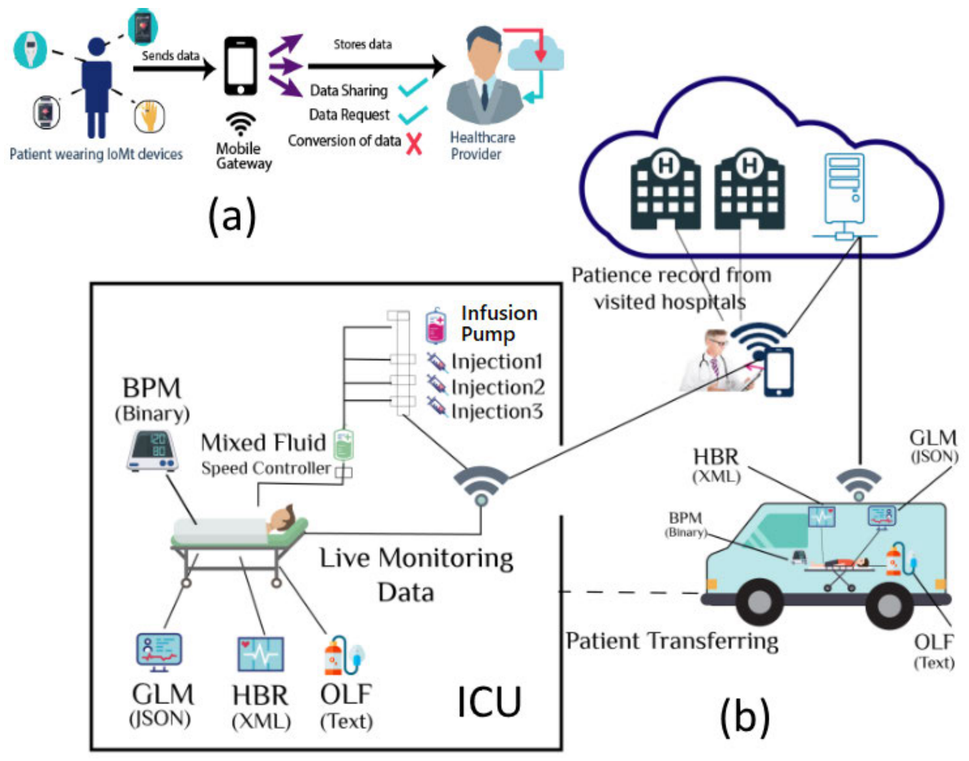


FIGURE 1. **Use cases of an IoMT-enabled smart city, (a) Patient wearing IoMT devices, managed with his smartphone. The data is shared with Healthcare provider through cloud. (b) Connected Ambulance, Hospitals and smart Intensive Care Unit.**

In addition, this ambulance is equipped with devices such as smart infusion controllers or ventilators, rendering remote access possible over the same infrastructure [16]. It enables a doctor to emulate a particular clinical environment that is specifically tailored to the patient's health conditions. However, in a real-time, horizontally-collaborative network, our controllers should be able to directly read from monitors, without cloud trips. When the devices are data-heterogeneous, the hub must also provide prompt data translations in this scenario. Alternatively, the MeDIC framework offloads the hub by putting available computing resources into capable healthcare devices at disposal. Now the treatment plans can be pre-configured into MeDIC controllers (inside the ambulance), and an appropriate plan is activated by the paramedic, earliest in an emergency situation.

2) INSIDE AN INTENSIVE CARE UNIT

A smart intensive care unit (ICU) is depicted in the left part of Figure 1(b). An ICU is different from a smart ambulance in that more capable healthcare devices are installed to address the diverse needs of critical patients [46]. Moreover, the devices are installed in groups (alongside patients' beds) to serve individual patients with diverse healthcare needs. Therefore, it is expected that some very capable devices are busy in one group, but are possibly laying idle in other groups. These devices are able to collaborate and share their idle resources if an appropriate collaborative framework, like MeDIC, is provided within the local area network of the ICU [47]. Translating the required data within the network will not only decrease uplink traffic, but will also enable real-time and autonomous control of medical actuators, including but not limited to infusion controller, ventilators, etc.

More recently, the COVID-19 pandemic has overburdened ICUs in terms of equipment, physical space, and human resources [48]. The equipment and infrastructures can be upgraded at a higher pace, but the front line human resources are scarce and at risk in COVID-19 [49]. Clinical decision support systems were proposed to reduce human interaction and man-made errors [50]. In an edge/fog computing paradigm, a real-time decision making is possible in AI[1]-enabled devices [51]. However, the interoperability among the devices from diverse vendor-base must be ensured at a real-time, using a framework like MeDIC. Similarly, edge computing resources may be added on-demand for decision support systems in large setups. Using MeDIC, compute resources of these edge computers can be efficiently utilized to offload device-level compute resources, whenever necessary.

3) INSIDE A SMART HOSPITAL

In the IoMT paradigm, a smart hospital connects individual smart medical facilities, including ICUs, operation theaters, laboratories, and offices [27], [52]. With MeDIC, all the

[1]Artificial Intelligence

devices in the wide area network of the hospital can share their idle resources. This also includes workstations available to the hospital staff. With the help of a MeDIC App, these workstations become Edge Computers (EC) that perform translations on-demand. Thus, the avalanche data requests originating from the individual facilities' networks can be further filtered out by the collaborating ECs. This may result in tremendous savings in bandwidth and cloud resource usage, along with reductions in translation latencies.

4) IN A SMART CITY

A smart city connects smart hospitals, clinics, kiosks, and ambulances to the patients in a coherent and collaborative way. Here, doctors have access to their patients' medical records irrespective of their geographic location and sampling methods. This access is provided through a medical application (App) on the doctor's smart monitoring device (a mobile phone or a smart pad), hereby referred to as a *doctor's wallet*. Patients are equipped with smart medical sensors that are implantable/wearable, forming a Body Area Network while their data are gatewayed through a patient's smartphone (Figure 1(a)) as a mobile edge computer [28], referred to as a *patient wallet*.

Conventionally, the patient's and doctor's wallets are connected to the cloud services, which archive electronic healthcare records (EHR) and authenticate their access, as shown in the upper right of Figure 1(b). Cloud services can be easily extended to provide data interoperability between IoMT devices, which is the focus of this work. However, such services have computing costs and are attributed to all sorts of issues of a typical cloud, including network latency and congestion, which are critical in medical uses [28]. Using a MeDIC App instead, we argue that the cloud computing load and network traffic related to data translation (to ensure interoperability) can be minimized by using the wallet's own compute resources.

III. PROPOSED FRAMEWORK OF MeDIC

MeDIC stands for Medical Data Interoperability through Collaboration of healthcare devices. The framework is built upon the basic idea of resolving the data format conflicts of IoMT devices within the requesting device itself or with the help of other IoMT devices within the local network.

The MeDIC framework is presented in Figure 2. On the top of the conventional IoT interfaces of authentication, subscribe and publish, MeDIC enables collaboration of IoMT devices through its Probe and Translation interfaces, integrated with a hierarchical resource manager. As shown in the figure, the interfaces provide autonomous and collaborative software agents [53] that expose the services offered by these interfaces. These agents are deployed within the IoMT devices that are categorized based on the following assumptions.

- *Legacy devices* implement conventional interfaces of authentication, subscribe, and publish only.

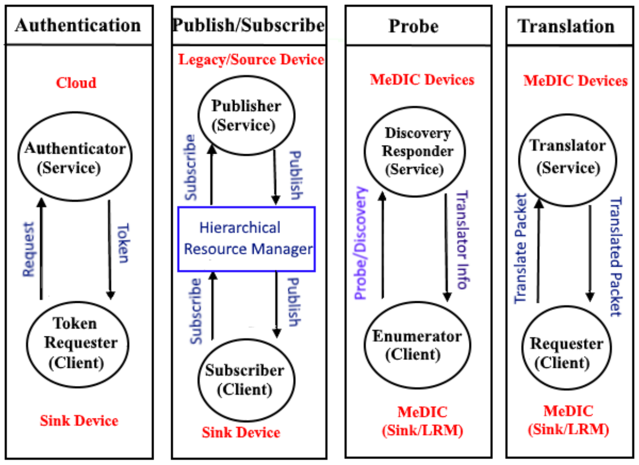


FIGURE 2. The proposed framework for Medical Data Interoperability through the Collaboration of Healthcare Devices (MeDIC).

- *MeDIC devices* implement MeDIC-specific Probe and Translation interfaces that allow resource sharing through data translation services.
- *Source devices* include sensors that generate a patient's medical data. In this work, we assume that source devices are legacy devices and are inherently resource-constrained. Therefore, they are only able to generate data in their vendor-specific formats.
- *Sink devices* include monitor, analytic, and transducer devices that subscribe to the patient's medical data. These also include patient's and doctor's wallets and other devices that possess adequate compute capabilities to process the data. Hence, these devices are the focus of this work. MeDIC sink devices are able to translate medical data from one format to another through their translation interface. For this purpose, their computing resources are enumerated with the help of the Probe interface. In this way, MeDIC devices can be categorized as Edge devices.
- *Resource Manager* represents a hierarchy with cloud resource manager (CRM) at the top, the edge resource manager (ERM) at the intermediate levels, and local resource manager (LRM) at the LAN levels. ERM and LRM cache the related portions of resource tables from CRM. In addition, they provide baseline translation services to legacy devices. This way, ERM/LRM can be categorized as Fog resource managers.

A. AUTHENTICATION

This interface facilitates a sink device to get access rights of a source device. The simplest solution provides an authentication server on the cloud that maintains a list of patient's devices with their access credentials and relevant data permissions, hereby referred to as the resource table (RTAB). A patient deice is identified and indexed with its Uniform Resource Identifier (URI) [54]. Once authenticated, devices can upload their data and download others' data in their native format [55], through the cloud resource manager (explained later). The process of authentication is depicted in Figure 3. The authentication interface provides two agents that are described below.

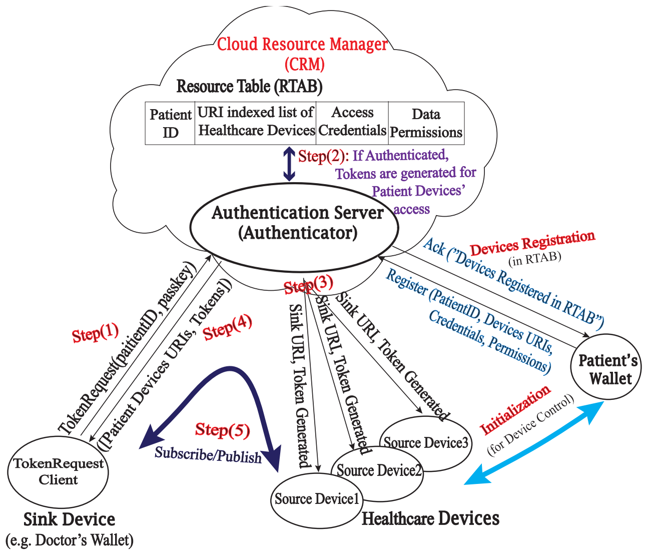

Cloud Resource Manager (CRM)
Resource Table (RTAB)
Patient ID
URI indexed list of Healthcare Devices
Access Credentials
Data Permissions
Step(2): If Authenticated, Tokens are generated for Patient Devices' access
Authentication Server (Authenticator)
Devices Registration (in RTAB)
Ack ("Devices Registered in RTAB")
Register (PatientID, Devices URIs, Credentials, Permissions)
Patient's Wallet
Step(1)
TokenRequest(paitientID, passkey)
([Patient Devices URIs, Tokens])
Step(4)
Step(3)
Sink URI, Token Generated
Sink URI, Token Generated
Sink URI, Token Generated
Step(5)
Subscribe/Publish
Initialization (for Device Control)
TokenRequest Client
Sink Device (e.g. Doctor's Wallet)
Source Device1
Source Device2
Source Device3
Healthcare Devices

FIGURE 3. **The authentication process.**

1) TokenRequestClient

This agent executes in a sink device when it needs to connect with a source device. The agent works as a handshaking client. It gets the access key (shared by the patient to the healthcare provider) of the source device as input and sends a request to the authenticator at the cloud (or in the patient wallet in case of local connectivity) for tokens and the URIs of patient IoMT devices.

2) AUTHENTICATOR

This agent executes within the cloud (or in the patient wallet in case of local connectivity) and implements the response side of handshaking. The input to this agent is the patient public key, and the message type is 'token request'. The authenticator generates access-tokens for all the patient devices indexed in RTAB of authenticator (registered through the patient's wallet). It then sends a token generated specifically for the respective patient device at corresponding URI, which saves it for authenticating data requests. A list of corresponding URIs and related tokens are then sent to the sink device (e.g., a healthcare provider's wallet) that uses them to request healthcare data of the patient. Once allotted, the token can be used as long as the communication is alive.

B. SUBSCRIBE/PUBLISH

This interface provides a Subscriber agent that facilitates a sink device to retrieve the medical data from a source device after the authentication process is successful. This agent executes in sink devices, including the doctor's wallet. The data request is made by using the source device's URI with the authentication token.

When a legacy device makes a data request through its Subscriber agent, the resource manager retrieves the data from the source device's Publisher agent, get it translated into the sink device's supported format if required and then dispatches it back to the Subscriber. This process is illustrated in Figure 4. Alternatively, a Subscriber agent in a MeDIC device is able to make requests of types 'DataRequest', 'DataRequestWithFormat' and 'ForwardRequest'. The resource manager subsequently responds in respective ways, which are further explained in the context of the resource manager and illustrated in Figures 5 and 6. Eventually, the Subscriber receives a properly formatted data and forwards it to the requesting process of the sink device.

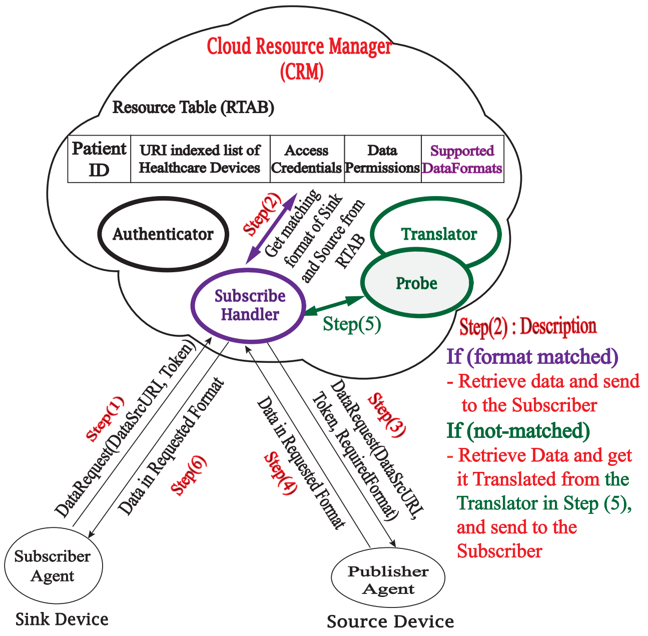

Cloud Resource Manager (CRM)
Resource Table (RTAB)
Patient ID
URI indexed list of Healthcare Devices
Access Credentials
Data Permissions
Supported DataFormats
Authenticator
Translator
Probe
Subscribe Handler
Step(2) Get matching format of Sink and Source from RTAB
Step(5)
Step(2) : Description
If (format matched)
- Retrieve data and send to the Subscriber
If (not-matched)
- Retrieve Data and get it Translated from the Translator in Step (5), and send to the Subscriber
Step(1) DataRequest(DataSrcURI, Token)
Step(6) Data in Requested Format
Step(4) Data in Requested Format
Step(3) DataRequest(DataSrcURI, Token, RequiredFormat)
Subscriber Agent
Sink Device
Publisher Agent
Source Device

FIGURE 4. **The Subscribe-Publish process through a Resource Manager.**

C. RESOURCE MANAGER

MeDIC relies on a hierarchy of resource managers to provide data interoperability. Conventionally, the resource manager is implemented in a cloud service referred to as the Cloud Resource Manager (CRM), as shown in Figure 4. For data interoperability, CRM is entrusted to translate data into the sink's supported format before dispatching. In MeDIC, this is done by extending the RTAB to maintain the list of supported formats (FList) of the participating devices in an IoMT, as shown in the figure.

As explained earlier, the sink device makes a data request through its Subscriber agent. A SubscribeHandler agent in the resource manager responds, initiating an RTAB look-up to get FLists of both source and sink devices. If the intersection of the retrieved FLists is not an empty set, a data format match is identified. In this case, a data request is forwarded to the source device, tagged with the matched format.

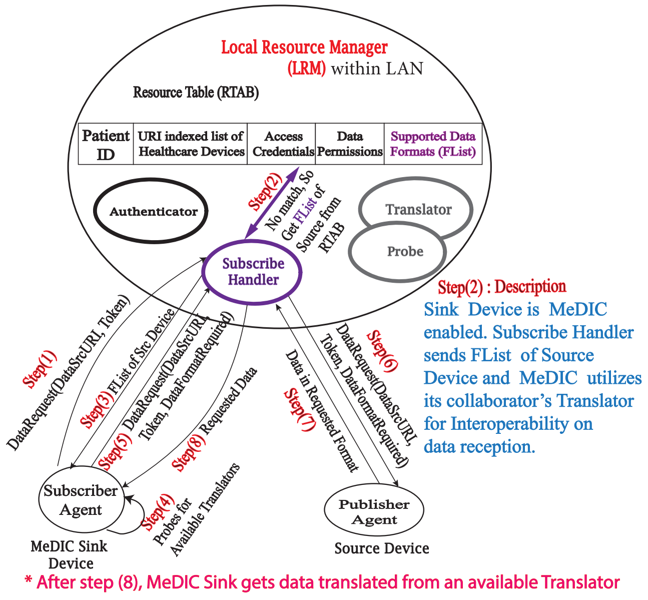

Local Resource Manager (LRM) within LAN
Resource Table (RTAB)
Patient ID | URI indexed list of Healthcare Devices | Access Credentials | Data Permissions | Supported Data Formats (FList)
Authenticator
Translator
Probe
Subscribe Handler
Step(2) No match, So Get FList of Source from RTAB
Step(2) : Description
Sink Device is MeDIC enabled. Subscribe Handler sends FList of Source Device and MeDIC utilizes its collaborator's Translator for Interoperability on data reception.
Step(1) DataRequest(DataSrcURI, Token)
Step(3) FList of Src Device
Step(5) DataRequest(DataSrcURI, Token, DataFormatRequired)
Step(8) Requested Data
Step(6) DataRequest(DataSrcURI, Token, DataFormatRequired)
Step(7) Data in Requested Format
Step(4) Probes for Available Translators
Subscriber Agent
MeDIC Sink Device
Publisher Agent
Source Device
* After step (8), MeDIC Sink gets data translated from an available Translator

FIGURE 5. Local Resource Manager: The architecture and the interface for MeDIC.

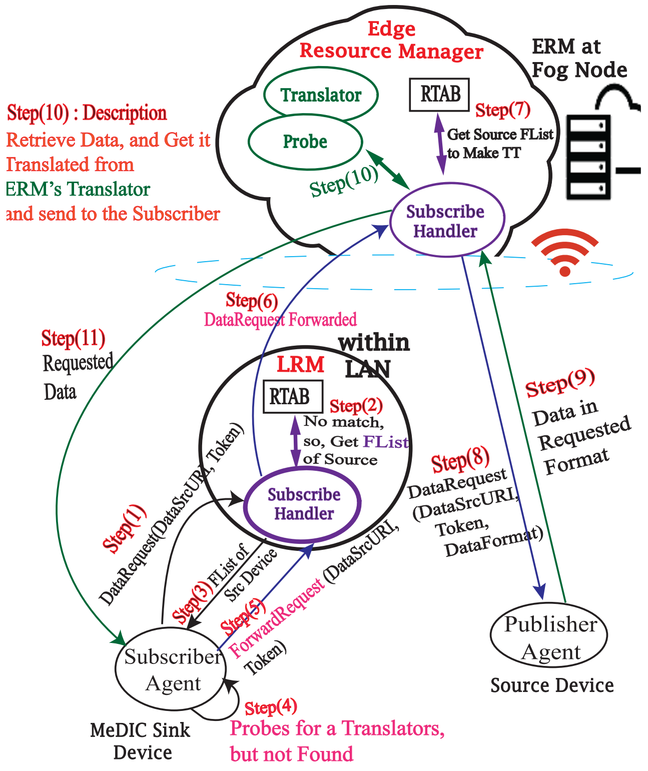

Edge Resource Manager
ERM at Fog Node
Translator
Probe
RTAB
Step(7) Get Source FList to Make TT
Step(10)
Subscribe Handler
Step(10) : Description
Retrieve Data, and Get it Translated from ERM's Translator and send to the Subscriber
Step(6) DataRequest Forwarded
Step(11) Requested Data
LRM within LAN
RTAB
Step(2) No match, so, Get FList of Source
Subscribe Handler
Step(1) DataRequest(DataSrcURI, Token)
Step(3) FList of Src Device
Step(5) ForwardRequest (DataSrcURI, Token)
Step(8) DataRequest (DataSrcURI, Token, DataFormat)
Step(9) Data in Requested Format
Subscriber Agent
MeDIC Sink Device
Step(4) Probes for a Translators, but not Found
Publisher Agent
Source Device

FIGURE 6. Edge Resource Manager (ERM) architecture and interface.

The retrieved data is then sent back to the sink device. Alternatively, CRM gets it translated through its own Probe and Translation agents into a format that is supported by the sink device ($DataFormat \in FList_{sink}$), before dispatch.

Evaluated later in this article, the above mentioned CRM, albeit its scalability, is expected to compromise the key feature of a real-time IoMT, i.e., the response time. To deter, MeDIC provides a cache of CRM as a proxy to the cloud, at the edge of our IoMT network. For example, in a hospital WAN, the resource manager is provided as a fog service, referred to as Edge Resource Manager (ERM) or within an Edge Computer. Alternatively, it is implemented within a low-cost hardware shell (an interoperability hub [15]), referred to as Local Resource Manager (LRM), within a LAN of small medical facilities, including ambulances and kiosks.

The purpose of the LRM is to provide baseline data interoperability of legacy devices within small networks, thus, saving the cloud trips. For this, the RTAB in LRM imports the relevant entries of healthcare devices present in the network from CRM. Besides, the routing appliance is configured to forward all IoMT requests to the respective LRM. The advantages include an efficient RTAB look-up because this local table is much smaller than one in CRM. Moreover, a data request can be locally served when the source device is also present in the same network because LRM caches URIs and access tokens. Finally, like CRM, the proposed LRM also houses MeDIC's Probe and Translation agents, which translate data from the source device's supported format to the legacy sink device's supported format, like interoperability Hubs in earlier works [38], [39]. However, the LRM can run out of computing resources when participating devices have diverse data formats and when sink devices are subscribing at a higher frequency, or subscribing to many source devices.

Here comes the fundamental advantage of MeDIC that avoids LRM resource bottlenecks, by allowing horizontal collaboration of participating healthcare devices to translate between data formats with the help of their own compute resources. To ascertain baseline interoperability, the LRM's translation resources are reserved for legacy devices, and their data requests are handled within the LRM in a fashion similar to CRM, as illustrated in Figure 4. To distinguish, MeDIC devices access LRM on a separate channel (using a specified network port). When a MeDIC sink makes a DataRequest, the SubscribeHandler in LRM responds with the FList of the source device, in a case when the intersection of sink and source FLists results in an empty set, as depicted in Figure 5. Once in the sink, the data formats present in this FList are then probed for a collaborator MeDIC device. On a hit, the MeDIC Subscriber initiates a subsequent DataRequestWthFormat, tagged with the matched data format. SubscribeHandler is now able to retrieve the requested data and sends it back to the Subscriber. From this point, the sink device assumes the responsibility to translate received data through the collaborating MeDIC device, thus offloading the LRM, in addition to avoiding a cloud trip.

When a MeDIC device is unable to find a collaborator, its Subscriber agent initiates a ForwardRequest. The LRM simply forwards it as a data request towards the ERM/CRM. In this way, both computing resources and network resources are optimally managed in the proposed framework. This process is depicted in Figure 6.

It is noteworthy that the RTAB entries in the LRM or ERM are populated from CRM after proper authentication, as described earlier in this section. Therefore, only authenticated MeDIC devices are entrusted with data translation, and data security issues are taken care of, centrally in the cloud.

D. PROBE

Compute resource sharing is enabled in MeDIC through its Probe and Translation agents. Particularly, the Probe agents maintain a Translation Resource Table (TRT) that enlists the capabilities and the state of available MeDIC devices, as illustrated in Figure 7. TRT is a hash table that is keyed with a Translation Tuple (TT).[2] Each entry in TRT maintains a priority queue containing the URIs of MeDIC devices that support that particular TT.[3] The queues are ordered with respect to the Load Factors of available MeDIC devices, having a device with the lowest Load Factor on the top. The Load Factor reflects the available computational resources in a MeDIC device and will be elaborated in the context of Translation, later in this section.

To help populating and maintaining its TRT, the Probe interface provides two agents, namely, the Enumerator and the Discovery Responder, which are detailed below.

1) ENUMERATOR

An Enumerator populates and maintains the device's TRT and is illustrated in Figure 7. The Enumerator makes a discovery broadcast, which is responded by all the MeDIC devices in the local network. A list of supported formats (the FList) and a Load Factor of the responding device is received as a response, which is used to enqueue the collaborating MeDIC device.

As MeDIC devices are able to translate between the formats listed in FList, the Translation tuples are simply the permutations of all 2-element subsets of the FList.[4] Each TT is then used either to make a new queue in TRT when one does not already exist or to enqueue the MeDIC device with respect to its Load Factor. The enumeration process is shown in Algorithm 1.

The network congestion due to probing is minimized in MeDIC by allowing the discovery broadcast in only two cases. First, when a MeDIC device is powered on, and later when a TT is missed in TRT, as explained later. As a result, a MeDIC node always gets the current network state at power-on, and gets listed in TRTs of other devices subsequently.

2) DISCOVERY RESPONDER

This agent implements the server-side of the Probe interface in MeDIC devices. Its job is to simply forward the natively supported FList with the current Load Factor, in response to a discovery broadcast, which is not a resource-intensive function. The discovery response generated by the Discovery Responder carries a list of supported formats (the FList), and the current Load Factor of the device, which is used by the Enumerator to enqueue the collaborating MeDIC device.

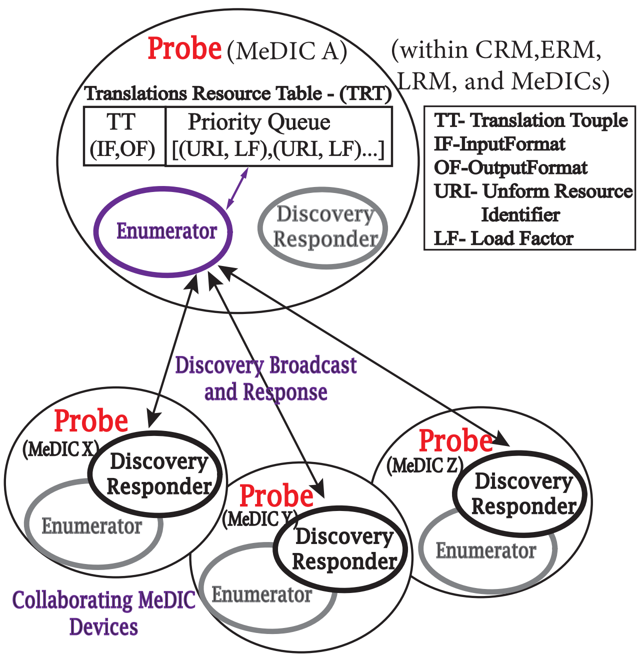

Probe (MeDIC A)
(within CRM,ERM, LRM, and MeDICs)
Translations Resource Table - (TRT)
TT (IF,OF)
Priority Queue [(URI, LF),(URI, LF)...]
TT- Translation Touple
IF-InputFormat
OF-OutputFormat
URI- Unform Resource Identifier
LF- Load Factor
Enumerator
Discovery Responder
Discovery Broadcast and Response
Probe (MeDIC X)
Probe (MeDIC Y)
Probe (MeDIC Z)
Collaborating MeDIC Devices

FIGURE 7. Enumerating MeDIC devices with Probe's Enumerator and Discovery Responder agents.

Algorithm 1 The Enumeration Algorithm

```
1:  procedure ENUMERATE($M$, $Flist$, $LF$)
2:      Inputs:
        M is the MeDIC device to be enrolled
        FList is a list of supported formats
        LF is the Load Factor
3:      Initialize:
        $S_T \leftarrow \{\}$
4:
5:      for $TT \in S_T$ do
6:          $Q \leftarrow TRT[TT]$                ▷ TRT is Hash table
7:          if $Q \neq NULL$ then
8:              $enqueue(Q, M, LF)$
9:          else
10:             $TRT[TT] \leftarrow newQueue(Head = M)$
11:         end if
12:     end for
13: end procedure
```

3) TRTWalk

When a MeDIC Subscriber agent receives FList of the source device in the response of initial DataRequest (Figure 5), it

[2] An ordered pair of input and output data formats, e.g. (JSON, XML) and (XML, JSON) are two different Translation Tuples.

[3] The supported TTs of a MeDIC device belong to the permutation set of its FList.

[4] For example, when the Flist consists of JSON, XML and CSV, the 6 TTs are $(JSON, XML)$, $(JSON, CSV)$, (XML, CSV), $(XML, JSON)$, $(CSV, JSON)$, (CSV, XML).

Algorithm 2 The TRTWalk Algorithm

```
 1: function TRTWALK($S_o$, $S_i$)
 2:     Inputs:
        $S_o = \{IF \mid \text{IF is a Source Data Format}\}$
        $S_i = \{OF \mid \text{OF is a Sink Data Format}\}$
Require: $S_o \cap S_i \equiv \{\}$                ▷ No format match
 3:     Initialize:
        $S_T \leftarrow S_o \times S_i$
        $S_T = \{(IF, OF) \mid IF \in S_o \wedge OF \in S_i\}$
 4:
 5:     for $TT \in S_T$ do
 6:         $Q \leftarrow TRT[TT]$                 ▷ TRT is Hash table
 7:         $Data \leftarrow$ QUEUEWALK($TT$, $Q$)
 8:         if $Data \neq NULL$ then
 9:             return $Data$
10:         end if
11:     end for
12:     $DiscoveryBroadcast()$                 ▷ Translator not found
13:     $Data \leftarrow ForwardRequest()$
14:     return $Data$
15: end function
```

Algorithm 3 The QueueWalk Algorithm

```
 1: function QUEUEWALK($TT$, $Q$)
 2:     Inputs:
        $TT = (IF, OF)$
        Q is the priority queue of MeDIC devices,
        with respect to LF
 3:     Initialize:
        $S_M \leftarrow \{\}$                  ▷ An empty stack
 4:
 5:     while $M \leftarrow dequeue(Q)$ do
 6:         $LF \leftarrow probeRequest(M)$
 7:         if $LF < Threshold$ then
 8:             break
 9:         else
10:             $S_M \leftarrow \{(M, LF)\} \cup S_M$
11:         end if
12:     end while
13:     for $(M, LF) \in S_M$ do
14:         enqueue(Q, M, LF)
15:     end for
16:     if $LF < Threshold$ then
17:         $D_I \leftarrow DataRequestWithFormat(IF(TT))$
18:         $(D_O, LF) \leftarrow TransRequest(M, D_I, TT)$
19:         enqueue(Q, M, LF)
20:         return $D_O$
21:     end if
22:     return NULL
23: end function
```

probes for a translation collaborator with a process called TRTWalk that is listed in Algorithm 2. In TRTWalk, a set S_T is formed by multiplying the received set of source formats with the set of native formats. Each entry in this set is a Translation tuple, for which a QueueWalk is iteratively performed.

In a QueueWalk (listed in Algorithm 3), the enqueued MeDIC devices are dequeued, iteratively. A dequeued entry, M, is then probed with a probeRequest which expects an updated Load Factor in response. If the Load Factor is above a threshold, the Queue Walk continues to probe the next available MeDIC device. In case when no appropriate collaborator is found, TRTWalk is repeated for the next TT, as listed in Algorithm 2. Eventually, a discovery broadcast is made within the local network to enumerate available MeDIC devices, simultaneously with a ForwardRequest when the set S_T is exhausted. The set S_M acts as a stack in QueueWalk, as dequeued MeDIC devices are temporarily pushed onto it during probing, and are later enqueued back at the end of a QueueWalk.

Alternatively, when a MeDIC device with an appropriated Load Factor is found, a DataRequestWithFormat is made. The retrieved data is then translated through the Translation interface, as illustrated in Figure 8. Queuing of the available MeDIC devices with respect to Load Factors is MeDIC's arbitration policy when more than one collaborators are available. Because a Load Factor reflects the current CPU load on a MeDIC device, the device with the lowest CPU load is selected for translation. In this way, a device that is initially at the top of the queue will slowly sink towards its bottom, effectively relieving it from further Translation requests. Thus, the goal is to efficiently enable collaboration, without overburdening one particular device. Similarly, the devices which become idle will automatically rise towards the top, and will eventually serve Translation requests. Moreover, some medical use cases would require assistance from an additional Edge resource to aid translations, as will be demonstrated in the next Section. Our proposed queuing policy will automatically put this Edge Computer on the top of the related queues, relieving overburdened MeDIC devices from serving Translation requests.

E. TRANSLATION

The Translation interface provides the mechanism through which MeDIC enables a capable device to perform on-demand data translation, by utilizing its redundant compute resources. It maintains a single attribute, a Load Factor which represents currently utilized resources, including CPU time and its memory occupation. The Translation interface provides two agents, a Requester and a Translator.

As illustrated in Figure 8 and explained above, when a MeDIC device A finds a collaborator with a workable Load Factor, its Subscriber agent fetches data with a DataRequestWithFormat request. The Requester agent now tags this data with the corresponding TT and sends it to the selected collaborator as a TranslationRequest. Subsequently, the Translator agent within the collaborator (MeDIC device X) receives this properly formatted request, and it immediately initiates the data translation.

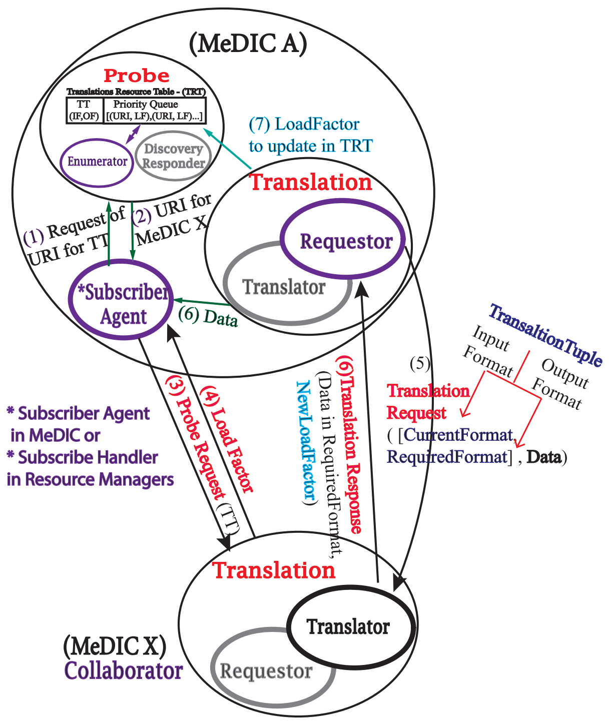


FIGURE 8. The Translation agents: the architecture and the interface.

1) TRANSLATION PROCESS

The translation of input data format into another format is a computation-intensive process [45]. It requires parsing, access, modification, and serialization of the input data. The parsing step is the most resource-intensive and thus poses the performance bottleneck for a translator. It involves character conversion into the target format, followed by a lexical analysis, syntactical analysis, semantic analysis, and an intermediate data tree generator.

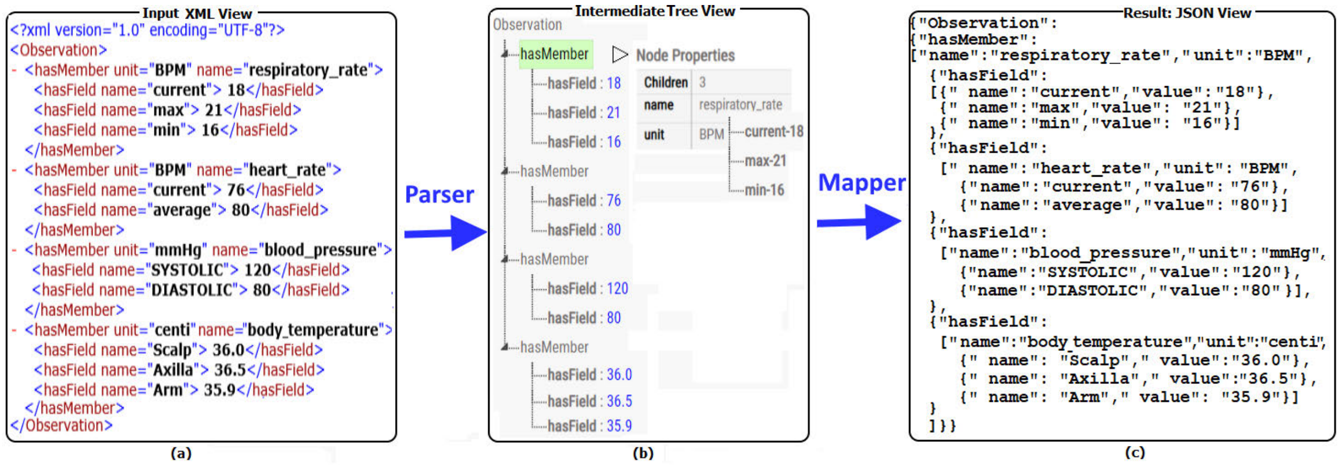
The translation process is elaborated in Figure 9 with the help of an illustrative example. It presents an input data snippet in XML format. To process this XML data, firstly, a Lexical Analyzer uses its pre-defined Token Set to convert the source XML listing into tokens by separating tags, double quotes, equal sign, data values, etc. Then the Parser applies the defined grammar rules for syntax analysis. In syntactic analysis, the Parser checks whether the sequence of tags is OK, the tags are properly closed, double quotes are properly formatted, and values are described in proper data types. If the validation test is passed, the meanings of XML tags are determined by a Semantic Analyser. It identifies the tags, attributes, and data for proper units and checks whether the data types are correctly mentioned. Finally, the Parser reads the data from XML Tags, into an intermediate Tree Data Representation (TDR) for later access, as shown in the figure. The main tag in XML becomes the root of this tree and the sub-tags are placed as children nodes, as shown in part (b) of Figure 9.

No data modification is required in this work. Therefore, a Mapper function traverses this TDR, recursively in pre-order (Root, Leftmost child — Rightmost child) and prints its data into the output stream, formatted in the desired output format (JSON in this example), a process called serialization. The output of the Mapper function is shown in part (c).

In a nutshell, a Parser function converts the received data into TDR, which is then converted into the required format by the Mapper function. At the completion of data translation, the Translator dispatches the converted data to the Requester along with the device's updated Load Factor, which is then used to enqueue the MeDIC device in the TRT of requesting device, as given in Algorithm 3 (Line 19). In this way, probes keep their TRTs in an updated state during the whole process.

2) LOAD FACTOR

The Load Factor is a representation of computing resources that are currently occupied within a MeDIC device. A realistic estimation of Load Factors is key to the effective-


Input XML View

```
<?xml version="1.0" encoding="UTF-8"?>
<Observation>
- <hasMember unit="BPM" name="respiratory_rate">
    <hasField name="current"> 18</hasField>
    <hasField name="max"> 21</hasField>
    <hasField name="min"> 16</hasField>
  </hasMember>
- <hasMember unit="BPM" name="heart_rate">
    <hasField name="current"> 76</hasField>
    <hasField name="average"> 80</hasField>
  </hasMember>
- <hasMember unit="mmHg" name="blood_pressure">
    <hasField name="SYSTOLIC"> 120</hasField>
    <hasField name="DIASTOLIC"> 80</hasField>
  </hasMember>
- <hasMember unit="centi" name="body_temperature">
    <hasField name="Scalp"> 36.0</hasField>
    <hasField name="Axilla"> 36.5</hasField>
    <hasField name="Arm"> 35.9</hasField>
  </hasMember>
</Observation>
```

(a)

Intermediate Tree View

```
Observation
  hasMember
    hasField : 18
    hasField : 21
    hasField : 16
  hasMember
    hasField : 76
    hasField : 80
  hasMember
    hasField : 120
    hasField : 80
  hasMember
    hasField : 36.0
    hasField : 36.5
    hasField : 35.9
```

Node Properties

Children	3	
name	respiratory_rate	
unit	BPM	current-18
		max-21
		min-16

(b)

Result: JSON View

```
{"Observation":
{"hasMember":
["name":"respiratory_rate","unit":"BPM",
  {"hasField":
  [{" name":"current","value":"18"},
   {" name":"max","value": "21"},
  },{" name":"min","value": "16"}]
  {"hasField":
   [" name":"heart_rate","unit": "BPM",
     {"name":"current","value": "76"},
     {"name":"average","value": "80"}]
  },
  {"hasField":
   ["name":"blood_pressure","unit":"mmHg",
     {"name":"SYSTOLIC","value":"120"},
     {"name":"DIASTOLIC","value":"80" }],
  },
  {"hasField":
   ["name":"body temperature","unit":"centi",
     {" name": "Scalp"," value":"36.0"},
     {" name": "Axilla"," value":"36.5"},
     {" name": "Arm"," value": "35.9"}]
  }
  ]}}
```

(c)

FIGURE 9. The illustrated translation process: (a) A data snippet from a source device supporting XML format, (b) Input data parsed into internal TDR, (c) TDR mapped into output (JSON) format.

ness of MeDIC, which is essentially a heterogeneous distributed system. A number of algorithms are developed over time to ascertain load balancing among heterogeneous nodes and rely on various algorithms to estimate their utilization factors [56]–[58]. However, in this work, we employ a simple queuing scheme for load balancing, as explained in the context of TRTWalk and a simple model to estimate Load Factor, as described below.

Suppose D is a MeDIC device with the computational power of N_D Million Instructions Per Second (MIPS). Its Load Factor LF_D is given as,

$$LF_D = LF_{ID} + \frac{L_{TQ} \times C_{Avg}}{N_D} \tag{1}$$

where LF_ID represents intrinsic load on device D, L_{TQ} is length of Translation Requests in queue and C_{Avg} is the average cost of translation algorithms in Million Instructions. In this way, the Load Factor reflects the expected time of completion of its task queue and can be utilized to arbitrate among MeDIC devices for translation services. On a Probe request, $LF_D + \frac{C_{Avg}}{N_D}$ is returned as a Load Factor to reflect the response time of a subsequent Translation request. On the completion of a translation, the current length of the queue is decremented by 1, and the new LF_D is conveyed back to the Requester in order to update its TRT accordingly, as given in Algorithm 3.

As depicted in Figure 8, when a Subscriber Agent in a MeDIC device or a Subscribe Handler in a resource manager (LRM or ERM) sends a Probe Request to a collaborating MeDIC device, the later shares its Load Factor which is calculated by using the model in (1). Also, the collaborator reports its new Load Factor when it returns the data in the required format after the translation.

IV. EVALUATION

The MeDIC is evaluated by extending an open-source IoT simulator, *iFogSim* [59], which enables the quantification of various performance metrics of cloud-, edge- or fog-based IoT frameworks. The evaluation simulates a hypothetical smart city which deploys MeDIC, as shown previously in Figure 2. The IoMT and infrastructure devices are configured with the parameters listed in Table 2, while the configuration tree is illustrated in Figure 10.

At the top of the MeDIC hierarchy of the smart city is a cloud at Level 0, hosting a CRM. It serves directly to smart hospitals in the smart city, its ambulances, kiosks, and the smart medical apps in individuals' wallets, each hosting an ERM or LRM at Level 1 or 2, respectively. Finally, both legacy and MeDIC devices constitute the leaf nodes of our configuration tree at Level 3 and are hierarchically served with LRM and ERM at respective levels, as shown in the figure while their simulated parameters are tabulated in Table 2, including available bandwidth (in bits per second), compute power (in Million Instructions Per Second) and physical memory size (in bytes).

TABLE 2. Device configurations in *iFogSim* simulations.

Device	Level	Bandwidth	MIPS	RAM
CRM	0	100 Gbps	1000K	8TB
ERM/ (Edge Computer)	1	10 Gbps	4000	8GB
LRM (Kiosk)	2	4 Mbps	1000	256MB
LRM (ICU)	2	1 Gbps	2000	1GB
MeDIC device	3	1 Mbps	200	64MB
Legacy device	3	100 kbps	50	1MB
Wallet (MeDIC App)	1,2,3	1 Mbps	2000	8GB

A. THE EVALUATION MODEL

In a real-time system, the response time is always critical [60]. In the framework of MeDIC, we model the response time (T_r) as the time elapsed since a Subscriber agent originates a request to the reception of data in the required format. This response time comprises of the accumulated round-trip times of all the network transactions (T_n), in addition to the processing time for data translation (T_p), when required. Mathematically,

$$T_r = T_n + T_p \tag{2}$$

Let $RM = \{LRM, ERM, CRM\}$ be a set of network levels of MeDIC framework, as illustrated in Figure 10, with latencies $T = \{T_l, T_e, T_c\}$, respectively. T_p depends on the compute capability of a particular node (tabulated as MIPS rating in Table 2) and its Load Factor, governed by (1). The network flight time depends on T_n depends on a number of factors, for example, the network level that serves a subscribe request (link latencies listed in Table 3), data packet size and the network capabilities of the nodes (listed as Bandwidth in Table 2).

1) CRM RESPONSE TIME

The worst-case response time is the time to get translation through the cloud services as,

$$T_r = 4L_c + T_p \tag{3}$$

2) LRM RESPONSE TIME

And the best case happens when a request is served through LRM. In this case,

$$T_r = 4L_l + T_p \tag{4}$$

When a transaction does not need translation, $T_p = 0$.

3) ERM RESPONSE TIME

When a request is served at ERM level, its response time is is given by,

$$T_r = 4L_e + T_p \tag{5}$$

4) MeDIC RESPONSE TIME

As shown in Figure 8, the response time is the time spent on an initial Data request, followed by n Probe requests in TRTwalk until a translator is found, followed by a Data

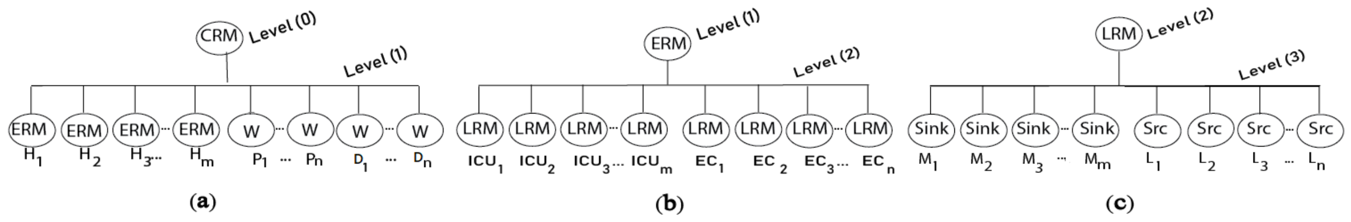


FIGURE 10. Network configurations for *iFogSim* simulations, (a) Smart City, (b) Smart Hospital, (c) Smart Ambulance/ICU. The legends used are CRM, ERM and LRM for Cloud, Edge, and Local resource managers, respectively. W is a Wallet device with MeDIC App, H stands for hospital, P is for Patient, D is for Doctor, EC stands for Edge computer, M is for MeDIC devices, and L is for Legacy devices.

request with the format and finally, a Translate request. Mathematically,

$$\begin{aligned} T_r &= T(FList) + T(Probe) + T(Data) \\ &\quad + T(Translate) + T_p \\ &= 2T_l + 2nT_l + 4T_l + 2T_l + T_p \\ &= 8T_l + 2nT_l + T_p \end{aligned} \tag{6}$$

5) AVERAGE RESPONSE TIME

The origination of the data request in IoMT is a random process. The average response time in MeDIC depends on factors such that source and sink data formats, load factors, and the number of iterations in the TRTWalk algorithm, which are all probabilistic in nature and are often dependant on each other. Therefore, instead of trying to develop complex closed-form solutions for average response time and bandwidth consumption, we rely on rigorous Monte-Carlo simulations, which are described next in this section.

B. EXPERIMENTAL SETUP

A rigorous Monte-Carlo simulation methodology is adopted in this work. We generate the iFogSim configurations dynamically, varying key simulation parameters, including the numbers of legacy/MeDIC devices within an LRM and the numbers of data requests. We further assume what data sinks generate data requests on an average of 1 request per second, peaking at 10 requests per second, with a Gaussian distribution. Moreover, the number of translation requests handled by the resource managers and the MeDIC devices are in proportion to their MIPS ratings. The translation time and CPU occupancy are also simulated, governed by the model in (1). The results obtained through these Monte-Carlo simulations are compiled and then presented as an average and a peak response, normalized with respect to those pertaining to our baseline (minimum) configurations as listed in Table 3.

1) USE CASE 1: SMART AMBULANCE AND KIOSKS

Up to ten medical devices were simulated in this scenario, including a varying number of (legacy) data sources and a relatively smaller number of (legacy and MeDIC) data sinks. The participating sink devices were restricted to less than ‘four’ to keep this simulation realistic. The simulation results are plotted in Figure 11.

TABLE 3. Network configurations for simulated use cases. Latency data has been sourced from [36].

Scenario	Configuration	Value
Ambulance	Resource Manager	LRM (Kiosk)
	Network Size (Min.)	(2) 1 sink, 1 source
	Network Size (Max.)	(10) 3 sinks, 7 sources
	Within LAN Latency	1 millisecond
	Cloud Latency	100 milliseconds
ICU	Resource Manager	LRM (ICU)
	Network Size (Min.)	(50) 6 sinks, 44 sources
	Network Size (Max.)	(250) 30 sinks, 220 sources
	Within LAN Latency	1 millisecond
	Cloud Latency	100 milliseconds
Hospital	Resource Manager	ERM
	Network Size (Min.)	5 LRMs (ICU)
	Network Size (Max.)	25 LRMs (ICU)
	Within WAN Latency	30 milliseconds
	Cloud Latency	100 milliseconds
City Cloud	Resource Manager	CRM
	Network Size (ERMs)	50 - 250 ERMs
	Network Size (Apps)	25000 active MeDIC App accounts

Figure 11(a) shows the average response time of data requests originating from the Subscriber Agents within the data sinks, with error bars corresponding to peak frequencies of data requests. Results show that the LRM is sufficient to handle the translation requests within a small network. In this case, even peak demands are met with LRM, evident from smaller error bars when the total number of devices is up to 4. Beyond that, peak demands manifest proportionally increasing response times because the data request rate is modeled to proportionally increase with the number of source devices in the network. MeDIC devices come to play their role in these situations, by collaborating through their probe and translation interfaces (Figure 8) and consequently reducing the response time in Figure 11(a) and the uplink traffic in Figure 11(b). The up-link traffic increases because Translation requests are being forwarded upwards to the cloud, which consequently increases the response time of such data requests. Because the response time is a critical factor in medical networks due to stringent real-time requirements, the MeDIC framework greatly improves this metric by utilizing local computing resources more efficiently, especially under load.

2) USE CASE 2: A SMART ICU

In this case study, we simulate a sufficiently large, MeDIC-enabled ICU [61]. Our *iFogSim* model consists of

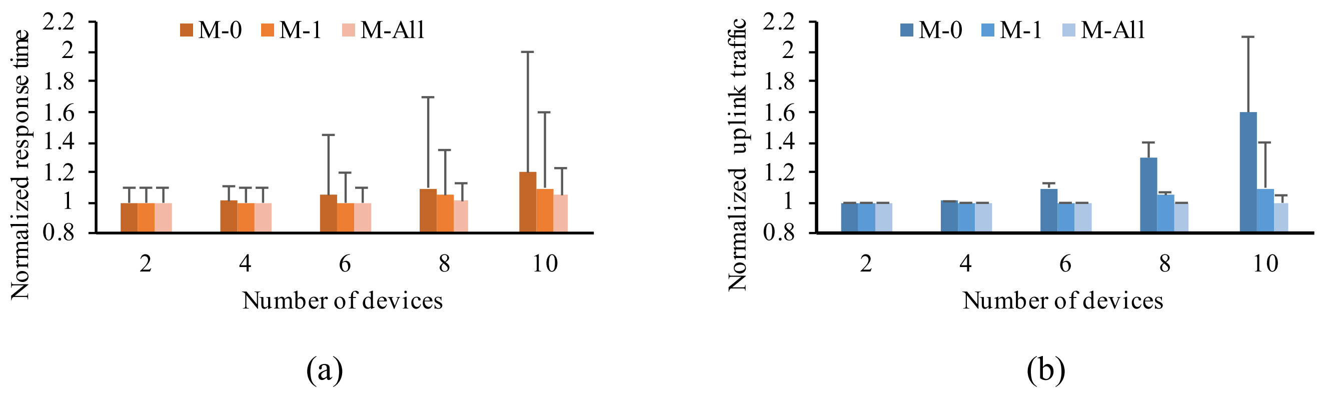


FIGURE 11. Response time and uplink traffic from LRM of a connected ambulance, with configurations, 1) no MeDIC sink (M-0), 2) one MeDIC sink (M-1), and 3) all MeDIC sinks (M-ALL). The Y-axis represents ratios, as simulation results are averaged and are further normalized with respect to Baseline case (of 2 devices). The solid bars represent average numbers, while the error bars represent peak numbers.

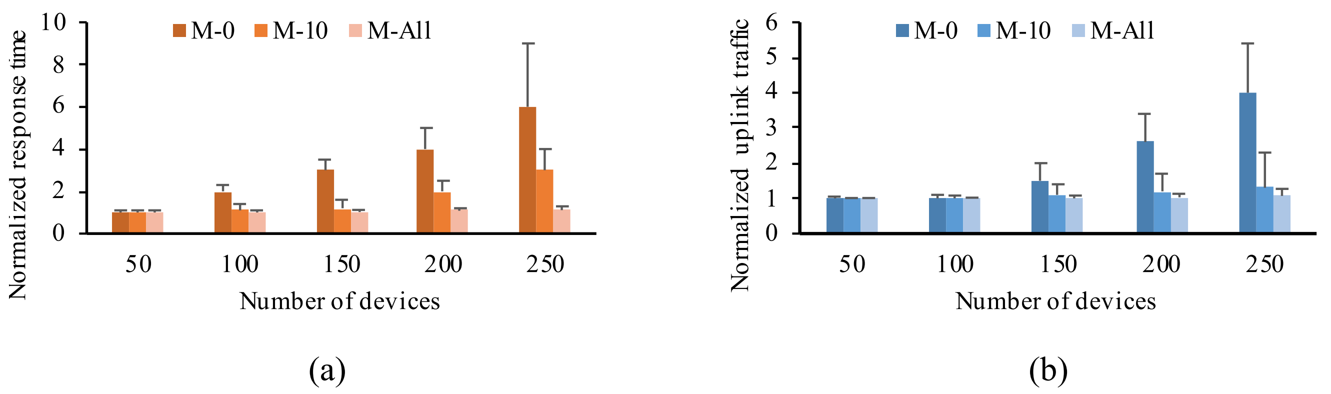


FIGURE 12. Response time and uplink traffic results from the LRM of a connected ICU with configurations, 1) no MeDIC sink (M-0), 2) ten MeDIC sink (M-10) and 3) all MeDIC sinks (M-ALL). The Y-axis represents ratios, as simulation results are averaged and are further normalized with respect to Baseline case (of 50 devices). The solid bars represent the normal occupancy while the error bars show an over-burdened situation amid a disaster.

an LRM, 10s of data sinks (including ventilators, drip controllers), and more than 100 legacy data sources (monitor devices) to record vital parameters of admitted patients. Two simulation scenarios are provided and are detailed below.

In the first scenario, ICU is dealing with a moderate number of patients with diverse needs. We model an ICU having 10 beds with 70% occupancy on average. However, idle devices are on standby and are collaborating through the MeDIC framework in this simulation. The response time and uplink bandwidth requirements of the simulated ICU are plotted in Figure 12. It is shown that when new patients are admitted, and consequently, more devices become active, and network congestion increases. A group of devices serving an individual patient may or may not have a MeDIC device. In case when the group only consists of legacy devices, translation is provided by an LRM. As shown in Table 2, the ICU LRM is sufficiently resourced to fulfill a moderate number of translation requests. Thus, the average response time does not degrade much with a monotonically increasing number of devices. However, peak demand aggravates the response time, especially in M-0, an ICU without MeDIC devices. M-10 results in a much flattened response here, due to the engagement of idle compute resources available within the network.

In addition to the above-mentioned Monte-Carlo simulations, we have also simulated a disastrous scenario in which ICU is suddenly occupied with patients. It generates traffic with an overwhelming number of data requests and eventually over-burdening our LRM with translation requests. As shown in the figure, both M-0 and M-10 cases suffer in this scenario, but M-All, a case where all data sinks are MeDIC, does not incur a significant penalty. This much improvement is achieved in part due to abundant local compute resources, and in part, due to the reduced number of cloud trips over the bandwidth-limited uplink, evident in Figure 12(b).

Finally, we simulated a pandemic situation (like COVID-19) where temporary but fully-equipped and large ICUs are deployed. A massively large number of patients admitted here show similar medical conditions and may have to share medical devices like ventilators. Shared devices are expected to generate more than usual data requests, and therefore over-burden themselves and the LRM. We cover this situation in our *iFogSim* simulations by modeling a 200 beds facility. Here, the data requests are multiplied by a factor of 4 as compared to the above worst case. The simulation results are summarized in Table 4. LRM and MeDIC infrastructure is clearly over-whelmed in this case. To help, we have simulated a case where an edge computer is inducted to provide data translation through a MeDIC App. As evident in the results, this device takes care of additional data traffic locally within the network and thus allows a minimal impact on the response time. In this way, MeDIC ensures the quality of the service, even amidst disasters.

TABLE 4. Results of a simulated pandemic situation, M-0 represents no MeDIC device, M-All is when all sinks are MeDIC, and M-EC represents an augmentation with an Edge computer. The values represent ratios, as simulation results are averaged and are further normalized with respect to the baseline case (of 50 devices).

Configuration	M-0	M-ALL	M-EC
Normalized Response Time	30.7	10.7	1.1
Normalized Uplink Traffic	19.6	5.2	1.3

3) USE CASE 3: A SMART HOSPITAL

A smart hospital was modeled in *iFogSim* with an Edge resource manager (ERM) on the top and up to 25 LRMs representing individual ICU facilities. Moreover, up to 10 workstations equipped with MeDIC App were also incorporated as Edge computers (EC). The configuration of an ERM/EC is given in Table 2. The results from *iFogSim* simulations are plotted in Figure 13.

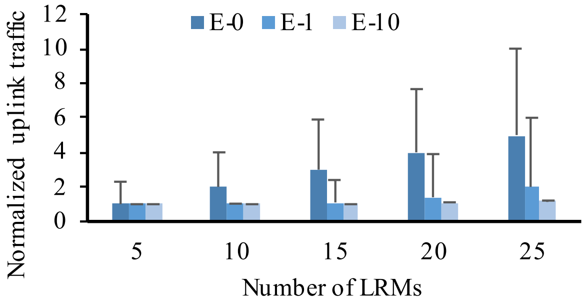


FIGURE 13. Uplink traffic towards Cloud from the ERM of a connected hospital with zero, one, and 10 Edge computers (E-0, E-1, and E-10, respectively). The Y-axis represents ratios, as simulation results are averaged and are further normalized with respect to Baseline case (of 5 devices).

The ERM is deployed to filter the translation requests from LRMs. Because ERM also performs translations with its probe and translator interfaces, it suffices for small hospitals, and no EC is required in this case, simulated as E-0.

However, in large setups, ERM must be augmented with additional computing resources, as in the case of ICUs amid disasters. This is also evident from E-1 and E-10 configurations in *iFogSim* simulations, where one and the ECs, respectively, are collaborating through the MeDIC framework, eventually reducing the cloud-bound translation traffic.

4) USE CASE 4: A SMART CITY

The *iFogSim* model of a smart city was included from 50 to 250 hospitals. For each experiment, the participating hospitals were randomly sized according to the configuration in Table 3. The number of translation requests is plotted in Figure 14, once without MeDIC and then with MeDIC deployment. The tremendous reduction in the number of requests can be seen in the figure with MeDIC deployment. As the required compute resources at the cloud increase proportionally to the number of translation requests, MeDIC also cuts down cloud processing costs, in addition to huge bandwidth savings and thus leads to a sustainable Internet of Medical things.

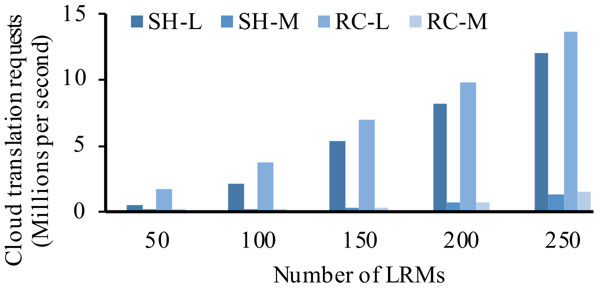


FIGURE 14. Translation requests redirected to the Cloud in Millions per second, for a smart hospital with legacy IoMT (SH-L), a smart hospital with a MeDIC framework (SH-M), remote care with legacy IoMT (RC-L) and remote care with a MeDIC framework (RC-M).

Finally, we present a case of remote medical care where a doctor's wallet is able to access the patient's medical devices through a patient's wallet. In Figure 14, we also plot the simulation results of this scenario with and without MeDIC. Exactly 2000 doctor's wallets and 25000 patient's wallets were simulated with parameters given in Table 2. Although the access is provided through a CRM, the proposed MeDIC framework avoids cloud compute resources because translation is done in house within a data sink. Thus, the plot shows a reduced number of translation requests when the MeDIC app is deployed in wallets. However, bandwidth will still be required to complete these transactions.

5) COMPARISON WITH RELATED WORK

From Table 1, the previously proposed solutions can be classified as cloud, middleware, fog, and edge-based interoperability. It has been established that a hierarchy of Fog/edge devices outperforms cloud-based solutions in their response time and bandwidth as well as cloud utilization [26], and as a result, many Interoperability Hubs are already proposed and commercialized. Here, we make a comparison of these schemes with our hierarchy of CRM, ERM and LRM, and MeDIC devices. Shown in Figures 11 and 12, the simulations were performed for three configurations, $M-0$, $M-x$ and $M-All$. Here $M-0$ represents a configuration without the collaborative framework of MeDIC and is similar to schemes proposed in [37]–[39]. It is evident from the results that as the number of devices in an IoMT grows, the resource-bottlenecks manifest themselves in aggravated response time and uplink traffic. Consequently, the spikes in data requests will be amplified to a point where the real-time performance of IoMT will be compromised.

With the inclusion of the MeDIC framework, the computing resources scale up with the growing number of MeDIC devices. More translation resources result in better probe operation, eventually reducing n in (6) and the overall response time, albeit an increased number of data requests. Therefore, the LRMs are offloaded to provide baseline interoperability to legacy devices, and ERMs filter the requests bound to the CRM, thereby reducing overall uplink bandwidth and cloud utilization, shown in Figures 13 and 14, respectively. Without MeDIC, the previously proposed solutions would work optimally with bounded requirements. However, more Edge Computers would be required to supplement Fog resources under load (as can be deduced from Table 4), while device resources would be under-utilized. Therefore, MeDIC enables optimal resource utilization by developing a horizontally integrated framework in an effort to extend the domain of the Internet of Medical things towards the fourth industrial revolution (Industry 4.0).

V. CONCLUSION AND FUTURE WORK

We have proposed MeDIC as a framework in which medical devices collaborate to translate otherwise incompatible data formats. The MeDIC framework provides services, including registration, subscribing, probing, translation, and publishing. The experiments were performed to analyze the effectiveness of MeDIC in terms of data response time and uplink traffic. The results show that the response time of both average and peak data requests is significantly improved. In the future, this work will be extended to the protocol and semantics compatibility.

REFERENCES

[1] H. Derhamy, J. Eliasson, and J. Delsing, "IoT Interoperability—On-demand and low latency transparent multiprotocol translator," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1754–1763, Oct. 2017.

[2] H. Habibzadeh, K. Dinesh, O. Rajabi Shishvan, A. Boggio-Dandry, G. Sharma, and T. Soyata, "A survey of healthcare Internet of Things (HIoT): A clinical perspective," *IEEE Internet Things J.*, vol. 7, no. 1, pp. 53–71, Jan. 2020.

[3] P. Puello Marrugo, E. Martinez Franco, and J. C. Rodriguez Ribon, "Systematic review of platforms used for remote monitoring of vital signs in patients with hypertension, asthma and/or chronic obstructive pulmonary disease," *IEEE Access*, vol. 7, pp. 158710–158719, 2019.

[4] M. Salayma, A. Al-Dubai, I. Romdhani, and Y. Nasser, "Wireless body area network (WBAN)," *ACM Comput. Surveys*, vol. 50, no. 1, pp. 1–38, Apr. 2017, doi: 10.1145/3041956.

[5] H. A. Rothan and S. N. Byrareddy, "The epidemiology and pathogenesis of coronavirus disease (COVID-19) outbreak," *J. Autoimmunity*, vol. 109, May 2020, Art. no. 102433, doi: 10.1016/j.jaut.2020.102433.

[6] E. J. Emanuel, G. Persad, R. Upshur, B. Thome, M. Parker, A. Glickman, C. Zhang, C. Boyle, M. Smith, and J. P. Phillips, "Fair allocation of scarce medical resources in the time of Covid-19," *New England J. Med.*, vol. 382, no. 21, pp. 2049–2055, May 2020, doi: 10.1056/NEJMsb2005114.

[7] D. S. W. Ting, L. Carin, V. Dzau, and T. Y. Wong, "Digital technology and Covid-19," *Nature Med.*, vol. 26, no. 4, pp. 459–461, Mar. 2020.

[8] T. Yang, M. Gentile, C.-F. Shen, and C.-M. Cheng, "Combining point-of-care diagnostics and Internet of medical things (IoMT) to combat the COVID-19 pandemic," *Diagnostics*, vol. 10, no. 4, p. 224, Apr. 2020.

[9] L. M. Camarinha-Matos, R, Fornasiero, and H. Afsarmanesh, "Collaborative networks as a core enabler of industry 4.0," in *Collaboration in a Data-Rich World* (IFIP Advances in Information and Communication Technology), vol. 506. Cham, Switzerland: Springer, 2017. [Online]. Available: https://link.springer.com/chapter/10.1007/978-3-319-65151-4_1

[10] *B. Braun Medical Inc*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.bbraunusa.com/en.html

[11] *International | Dexcom*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.dexcom.com/global

[12] *Home—Companion Medical*. Accessed: Apr. 30, 2020. [Online]. Available: https://www.companionmedical.com/

[13] C. Lubamba and A. Bagula, "Cyber-healthcare cloud computing interoperability using the HL7-CDA standard," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2017, pp. 105–110.

[14] M. Noura, M. Atiquzzaman, and M. Gaedke, "Interoperability in Internet of Things: Taxonomies and open challenges," *Mobile Netw. Appl.*, vol. 24, no. 3, pp. 796–809, Jun. 2019.

[15] *Healthgo, Home Health Hub for Remote Patient Monitoring*. Accessed: Apr. 29, 2020. [Online]. Available: https://www.edevice.com/products/healthgo

[16] *Medisante Putting IoT to Work for Caregivers*. Accessed: Apr. 30, 2020. [Online]. Available: https://medisante.ch/hub

[17] A. Azaria, A. Ekblaw, T. Vieira, and A. Lippman, "MedRec: Using blockchain for medical data access and permission management," in *Proc. 2nd Int. Conf. Open Big Data (OBD)*, Aug. 2016, pp. 25–30.

[18] A. Roehrs, C. A. da Costa, R. da Rosa Righi, S. J. Rigo, and M. H. Wichman, "Toward a model for personal health record interoperability," *IEEE J. Biomed. Health Informat.*, vol. 23, no. 2, pp. 867–873, Mar. 2019.

[19] M. M. AlZghoul, M. A. Al-Taee, and A. M. Al-Taee, "Towards nationwide electronic health record system in jordan," in *Proc. 13th Int. Multi-Conf. Syst., Signals Devices (SSD)*, Mar. 2016, pp. 650–655.

[20] J. H. Brenas, M. S. Al-Manir, C. J. O. Baker, and A. Shaban-Nejad, "A malaria analytics framework to support evolution and interoperability of global health surveillance systems," *IEEE Access*, vol. 5, pp. 21605–21619, 2017.

[21] N. Boutros-Saikali, K. Saikali, and R. A. Naoum, "An IoMT platform to simplify the development of healthcare monitoring applications," in *Proc. 3rd Int. Conf. Electr. Biomed. Eng., Clean Energy Green Comput. (EBECEGC)*, Apr. 2018, pp. 6–11.

[22] G. Pulkkis, J. Karlsson, M. Westerlund, and J. Tana, "Secure and reliable Internet of Things systems for healthcare," in *Proc. IEEE 5th Int. Conf. Future Internet Things Cloud (FiCloud)*, Aug. 2017, pp. 169–176.

[23] G. S. Tamizharasi, H. P. Sultanah, and B. Balamurugan, "IoT-based E-health system security: A vision archictecture elements and future directions," in *Proc. Int. Conf. Electron., Commun. Aerosp. Technol. (ICECA)*, vol. 2, Apr. 2017, pp. 655–661.

[24] S. Jabbar, F. Ullah, S. Khalid, M. Khan, and K. Han, "Semantic interoperability in heterogeneous IoT infrastructure for healthcare," *Wireless Commun. Mobile Comput.*, vol. 2017, pp. 1–10, 2017.

[25] M. Clarke, J. de Folter, V. Verma, and H. Gokalp, "Interoperable end-to-end remote patient monitoring platform based on IEEE 11073 PHD and ZigBee health care profile," *IEEE Trans. Biomed. Eng.*, vol. 65, no. 5, pp. 1014–1025, May 2018.

[26] H. Moustafa, E. M. Schooler, G. Shen, and S. Kamath, "Remote monitoring and medical devices control in eHealth," in *Proc. IEEE 12th Int. Conf. Wireless Mobile Comput., Netw. Commun. (WiMob)*, Oct. 2016, pp. 1–8.

[27] A. Garai and A. Adamko, "Comprehensive healthcare interoperability framework integrating telemedicine consumer electronics with cloud architecture," in *Proc. IEEE 15th Int. Symp. Appl. Mach. Intell. Informat. (SAMI)*, Jan. 2017, pp. 000411–000416.

[28] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, "Fog computing in healthcare–A review and discussion," *IEEE Access*, vol. 5, pp. 9206–9222, 2017.

[29] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for Internet of Things: A primer," *Digit. Commun. Netw.*, vol. 4, no. 2, pp. 77–86, Apr. 2018.

[30] E. Reilent, I. Løøbas, A. Kuusik, and P. Ross, "Improving the data compatibility of PHR and telecare solutions," in *Proc. 5th Eur. Conf. Int. Fed. Med. Biol. Eng.*, Á. Jobbágy, Ed. Berlin, Germany: Springer, 2012, pp. 925–928.

[31] F. Khalique, S. A. Khan, and I. Nosheen, "A framework for public health monitoring, analytics and research," *IEEE Access*, vol. 7, pp. 101309–101326, 2019.

[32] N. Georgi, A. Corvol, and R. Le Bouquin Jeannes, "Middleware architecture for health sensors interoperability," *IEEE Access*, vol. 6, pp. 26283–26291, 2018.

[33] R. Ivanov, H. Nguyen, J. Weimer, O. Sokolsky, and I. Lee, "OpenICE-lite: Towards a connectivity platform for the Internet of medical things," in *Proc. IEEE 21st Int. Symp. Real-Time Distrib. Comput. (ISORC)*, May 2018, pp. 103–106.

[34] B. Negash, T. Westerlund, and H. Tenhunen, "Towards an interoperable Internet of Things through a Web of virtual things at the fog layer," *Future Gener. Comput. Syst.*, vol. 91, pp. 96–107, Feb. 2019.

[35] T. Nguyen Gia, M. Jiang, V. K. Sarker, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, "Low-cost fog-assisted health-care IoT system with energy-efficient sensor nodes," in *Proc. 13th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2017, pp. 1765–1770.

[36] H. Rahman and M. I. Hussain, "Fog-based semantic model for supporting interoperability in IoT," *IET Commun.*, vol. 13, no. 11, pp. 1651–1661, Jul. 2019.

[37] A. Ahmed, M. Kleiner, and L. Roucoules, "Model-based interoperability IoT hub for the supervision of smart gas distribution networks," *IEEE Syst. J.*, vol. 13, no. 2, pp. 1526–1533, Jun. 2019.

[38] 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 *Proc. Global Internet Things Summit (GIoTS)*, Jun. 2017, pp. 1–6.

[39] M. H. Cintuglu, T. Youssef, and O. A. Mohammed, "Development and application of a real-time testbed for multiagent system interoperability: A case study on hierarchical microgrid control," *IEEE Trans. Smart Grid*, vol. 9, no. 3, pp. 1759–1768, May 2018.

[40] *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Standard 610.12-1990, 1990, pp. 1–84.

[41] H. Kaur, M. A. Alam, R. Jameel, A. K. Mourya, and V. Chang, "A proposed solution and future direction for blockchain-based heterogeneous medicare data in cloud environment," *J. Med. Syst.*, vol. 42, no. 8, p. 156, Aug. 2018.

[42] Q. Life, "Compatibility of device data with hospital information systems (white paper)," Qualcomm Life Inc., San Diego, CA, USA, Tech. Rep. MKT-210 DCN 2018-206 Rev 3.0, Nov. 2018. [Online]. Available: https://capsuletech.com/wp-content/uploads/2019/01/compatability-of-device-data-with-hospital-information-systems.pdf

[43] Kathy Tong. (Jun. 2012). *Ehr Compatibility and Connectivity: Two Obstacles to Patient Care*. Accessed: May 11, 2019. [Online]. Available: https://ehrintelligence.com/news/ehr-compatibility-and-connectivity-two-obstacles-to-patient-care

[44] R. Quinn. (Feb. 2015). *Compatibility Issues Make Physicians Use of Electronic Health Records Systems Tougher—The Rheumatologist*. Accessed: Mar. 11, 2019. [Online]. Available: https://www.the-rheumatologist.org/article/compatibility-issues-make-physicians-use-of-electronic-health-records-systems-tougher/

[45] T. Lam, J. J. Ding, and J.-C. Liu, "XML document parsing: Operational and performance characteristics," *Computer*, vol. 41, no. 9, pp. 30–37, Sep. 2008.

[46] A. S. R. M. Ahouandjinou, K. Assogba, and C. Motamed, "Smart and pervasive ICU based-IoT for improving intensive health care," in *Proc. Int. Conf. Bio-Eng. Smart Technol. (BioSMART)*, Dec. 2016, pp. 1–4.

[47] N. A. Halpern, "Innovative designs for the smart ICU," *Chest*, vol. 145, no. 3, pp. 646–658, Mar. 2014.

[48] T. Kain and R. Fowler, "Preparing intensive care for the next pandemic influenza," *Crit. Care*, vol. 23, no. 1, Oct. 2019.

[49] J. E. Hollander and B. G. Carr, "Virtually perfect? Telemedicine for Covid-19," *New England J. Med.*, vol. 382, no. 18, pp. 1679–1681, Apr. 2020, doi: 10.1056/nejmp2003539.

[50] M. Frize, C. M. Ennett, M. Stevenson, and H. C. Trigg, "Clinical decision support systems for intensive care units: Using artificial neural networks," *Med. Eng. Phys.*, vol. 23, no. 3, pp. 217–225, Apr. 2001, doi: 10.1016/S1350-4533(01)00041-8.

[51] X. Liu, R. H. Deng, Y. Yang, H. N. Tran, and S. Zhong, "Hybrid privacy-preserving clinical decision support system in fog–cloud computing," *Future Gener. Comput. Syst.*, vol. 78, pp. 825–837, Jan. 2018.

[52] H. Zhang, J. Li, B. Wen, Y. Xun, and J. Liu, "Connecting intelligent things in smart hospitals using NB-IoT," *IEEE Internet Things J.*, vol. 5, no. 3, pp. 1550–1560, Jun. 2018.

[53] H. S. Nwana, "Software agents: An overview," *Knowl. Eng. Rev.*, vol. 11, no. 3, pp. 205–244, Sep. 1996.

[54] T. Clark, *Uniform Resource Identifier (URI)*. New York, NY, USA: Springer, 2013, pp. 2319–2320.

[55] N. Kahani, K. Elgazzar, and J. R. Cordy, "Authentication and access control in e-health systems in the cloud," in *Proc. IEEE 2nd Int. Conf. Big Data Secur. Cloud (BigDataSecurity) Int. Conf. High Perform. Smart Comput. (HPSC), IEEE Int. Conf. Intell. Data Secur. (IDS)*, Apr. 2016, pp. 13–23.

[56] A. Jaleel, S. Arshad, and M. Shoaib, "A secure, scalable and elastic autonomic computing systems paradigm: Supporting dynamic adaptation of self-* services from an autonomic cloud," *Symmetry*, vol. 10, no. 5, p. 141, May 2018, doi: 10.3390/sym10050141.

[57] M. Alam, R. A. Haidri, and M. Shahid, "Enhanced load balancing strategy with migration cost on heterogeneous distributed systems," in *Proc. 3rd Int. Conf. Contemp. Comput. Informat. (IC3I)*, Oct. 2018, pp. 273–278.

[58] J. M. Shah, K. Kotecha, S. Pandya, D. B. Choksi, and N. Joshi, "Load balancing in cloud computing: Methodological survey on different types of algorithm," in *Proc. Int. Conf. Trends Electron. Informat. (ICEI)*, May 2017, pp. 100–107.

[59] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "IFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, edge and fog computing environments," *Softw., Pract. Exper.*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017.

[60] A. Jaleel, S. Arshad, M. Shoaib, and M. Awais, "Design quality metrics to determine the suitability and cost-effect of Self-* capabilities for autonomic computing systems," *IEEE Access*, vol. 7, pp. 139759–139772, 2019.

[61] S. O. Koh, P. H. Park, M. H. Kong, C. Y. Jeung, W. M. Lim, and Y. L. Kim, "Number of beds and types of intensive care unit (ICU) in university and non-university hospitals in korea," *Korean J. Crit. Care Med.*, vol. 13, no. 2, pp. 212–217, 1998.

ABDUL JALEEL received the B.S., M.S., and Ph.D. degrees in computer science and engineering from the University of Engineering and Technology, Lahore, Lahore, Pakistan, in 2006, 2010, and, 2019, respectively. He is currently working as an Assistant Professor with the Rachna College of University of Engineering and Technology, Lahore, Pakistan. His research interests include the development of self-managing software applications, Health-IoT, autonomic computing, and software quality measurement metrics.

TAYYEB MAHMOOD received the Ph.D. degree in information and communication engineering from the Korea Advanced Institute of Science and Technology, Daejeon, South Korea. He is currently working as an Assistant Professor with the Department of Electrical Engineering, University of Engineering and Technology, Lahore, Lahore, Pakistan. His research interests include low-power computing, the Internet of Things, and embedded systems.

MUHAMMAD AWAIS HASSAN received the B.S. degree (Hons.) in computer science from Punjab University, and the M.S. and Ph.D. degrees in computer science from the University of Engineering and Technology, Lahore, Lahore, Pakistan. He is currently working as an Assistant Professor with the Department of Computer Science and Engineering, University of Engineering and Technology, Lahore. His research interests include artificial intelligence, reinforcement learning, and adaptive eLearning systems.

GULSHAN BANO received the M.S. degree in computer science from the University of Engineering and Technology, Lahore, in 2018. Since October 2018, she has been working as a Lecturer with the Department of Information Technology, University of Sialkot, Pakistan. Her specialization is in software engineering and machine learning. She supervises the researches on machine learning, data sciences, and software engineering. Her current research interests include machine learning and artificial intelligence for social media usage and gratification on youth behaviors.

SYED KHALDOON KHURSHID received the Ph.D. degree in computer science, in 2014. He is currently working as an Assistant Professor with the Department of Computer Science and Engineering, University of Engineering and Technology (UET), Lahore, Lahore, Pakistan. His current research interests are in smart correlation patterns among devices and empathy in artificial intelligence.