Vision, Challenges and Future Perspectives of Low Constrained Devices IOT Operating Systems: A Systematic Mapping Review

Sumera Rounaq, Muhammad Iqbal

Abstract **— Now the far-fetched reality has become true with the prominence of IOT (Internet of Things) technology. Various individual devices get connected with each other to establish communication. These devices are built on a microcontroller which is responsible to receive and send information. These devices are very small and appropriate Operating Systems are required on the basis of particular device architecture, scheduling methods, network technologies and programming models. IOT Operating Systems are enormously facilitating low constrained devices to deliver their throughput efficiently in a timely manner. This concept helped a lot in emergence of IOT, which has translated our physical world into a digital cyber world. IOT devices consumes less power, less memory and less energy, therefore they need appropriate Operating Systems to act as interfaces. Low constrained Operating Systems are especially designed to provide support to these low constrained devices. Many researches have been conducted to discuss Operating Systems for these low constrained devices. In this study, capsulization of Internet of Things and its building blocks, architecture of IOT Operating System and network stack architecture of state-of - the art IOT Operating Systems such as Contiki, Tiny OS, Free RTOS, RIOT, Zephyr and Mbed OS is investigated. Moreover this, detailed overview of related work is presented with the comparative analysis of this study with the existing surveys. In addition, open research areas are discussed with recommendations.**

Index Terms **— Internet of Things, Low Constrained Devices, Operating System, RIOT, Zephyr, Mbed OS.**

I. INTRODUCTION

Parallel with the expansion of Internet of Things, network of connected electronic devices is expanding across the globe. To facilitate users, these devices exchange data and information and provide value creation which is going to embrace proximate level of digitization.

Acclivity in wireless technologies has increased demand of IOT devices. These IOT devices interconnected with each other, utilizing embedded sensing and communication. The word '**things**' in IOT is referred to as endpoints (devices and things). IOT is the connection of these endpoints through uniquely identifiable IP addresses. Moreover this, IOT bridges gap between virtual and physical realities by acting as an additional layer of information. In several industries and companies, IOT has created a tangible value. Therefore, by 2050 surge of IOT devices is expected.

Published on December 28, 2020.
Sumera Rounaq, Bahria University Karachi Campus, Pakistan.
(e-mail: sumaira.rounaq@gmail.com)
Muhammad Iqbal, Bahria University Karachi Campus, Pakistan.
(e-mail: miqbal.bukc@bahria.edu.pk)

According to IOT Analytics [2], in 2016, more than 4.7 billion devices were connected to the internet and by 2025 it is estimated that there will be birth of more than 21 billion devices [1]. As per [2], Fig. 1 depicts the expected growth of global number of connected IOT devices by 2025.

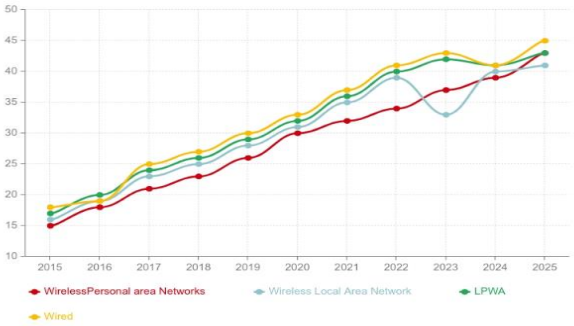

Fig. 1. Growth of global number of connected IOT devices by 2025.

II. CONTRIBUTION AND METHODOLOGY

A. *Contribution to this Survey*

As compare to recent review papers, this survey is summarized as follows:

- This survey presents contribution, structure and selected studies relevant to this study.
- We present discussion on IOT building components and their architecture.
- Compare to previous review in the same context we cover Contiki, Tiny OS, Free RTOS, RIOT, Zephyr, Mbed OS for low constrained devices.
- We review IOT taxonomy along with discussion of key features and characteristics of major IOT OS.
- We also contribute towards Network Stack Architecture and also provide a detailed summary of existing surveys.
- Open issues are also discussed to facilitate future research studies.

B. *Structure of this Survey*

In this paper, we encapsulate details of low-end device IOT Operating Systems in frame of reference of architecture, scheduler, programming model, programming language and network stack architecture. We covered Contiki, Tiny OS, Free RTOS, RIOT, Zephyr and Mbed OS.

The contribution of this paper is structured as follows:

- It presents a detailed review of previous review studies in the section of Related work.

- It discusses network stack architecture with appropriate illustrations.
- It covers overview of IOT building blocks, components and its architecture.

Whole paper is structured as follows:

Section I provides introduction; Section II gives contribution and structure of this survey. Table 1 presents selected studies to define the inclusion criteria of our systematic mapping. Section III provides overview of IOT, building components, IOT taxonomy and IOT OS architectures. Next is Section IV which defines IOT Operating Systems used for this study. Section V focuses on architecture types and features of IOT Operating Systems. Section VI presents Network Stack architecture and Section VII gives comprehensive review of state of art review studies. Comparative analysis of this review with the existing studies is also discussed in this section. Section VIII is dedicated to open research issues followed by Section IX which is comprised of conclusion.

C. Studies selected for this Survey

Following are the studies taken into consideration to complete this survey. Table 1 presents peer review papers along with year of publication and authors.

TABLE 1: SELECTED STUDIES

S. No	Year	Authors	Title
1	2018	Kumar et al. [3]	The Internet of Things: Insights into the building blocks, component interactions, and architecture layers
2	2018	E. Baccelli, et al. [4]	RIOT: an open-source operating system for low-end embedded devices in the IoT
3	2020	Bansal, et al. [5]	IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication
4	2019	Zikria, Yousaf Bin, et al. [6]	Internet of Things (IoT) operating systems management: opportunities, challenges, and solution
5	2018	Musaddiq, Arslan, et al [7]	A survey on resource management in IoT operating systems
6	2017	Sabri, et al. [8]	Comparison of IoT constrained devices operating systems: A survey
7	2018	Javed, Farhana, et al. [9]	Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review
8	2017	Amiri-Kordestani, et al. [10]	A survey on embedded open-source system software for the internet of things.
9	2019	Shammar, et al. [11]	The Internet of Things (IoT): a survey of techniques, operating systems, and trends
10	2016	Chandra, et al. [12]	Operating systems for internet of things: A comparative study.
11	2019	Srinidhi, et al. [13]	Network optimizations in the Internet of Things: A review.
12	2015	Gaur, Pamini, et al. [14]	Operating systems for IoT devices: A critical survey
13	2015	Hahm, Oliver, et al [15]	Operating systems for low-end devices in the internet of things: a survey
14	2019	Silva, Miguel, et al [16]	Operating Systems for Internet of Things Low-End Devices: Analysis and Benchmarking
15	2004	A. Dunkels, et al. [17]	Contiki – a lightweight and flexible operating system for tiny networked sensors
16	2005	P. Levis, et al. [18]	TinyOS: An operating system for sensor networks
17	2006	A. Dunkels et al. [19]	Protothreads: Simplifying event-driven programming of memory-constrained embedded systems
18	2008	T. Alliance, et al. [20]	TinyOS 2.1: Adding threads and memory protection to TinyOS
19	2012	P. Lindgren, et al. [21]	Leveraging TinyOS for integration in process automation and control systems
20	2007	R. Goyette, et al. [22]	An analysis and description of the inner workings of the FreeRtos kernel
21	2009	D. Déharbe, et al. [23]	Formalizing FreeRTOS: First steps," *Formal Methods: Foundations and Applications*
22	2014	J. F. Ferreira, et al. [24]	Automated verification of the FreeRTOS scheduler in hip/sleek
23	2020	Cekerevac, Z, et al. [25]	TOP SEVEN IoT OPERATING SYSTEMS IN MID-2020

III. OVERVIEW OF IOT

When it comes to no human – machine physical contact while solving problems related to science and engineering domain, only one name comes in our mind and that is IOT. IOT is a revolutionary technology that is gaining popularity by leaps and bounds. Due to advancements in network connectivity, real world objects have liberty to establish connectivity between them. Sharing of information is quite feasible and these objects are identified as nodes in IOT framework. Adding more to it, real world objects join their hands with the sensing elements, micro controllers, internet protocols and storage. Integration of the real-world objects make possible all aspects of communication to accomplish real world tasks.

A. IOT Building Components

Things, Gateways, Network Infrastructure (NI) and Cloud Infrastructure mainly participates in the implementation of IOT. Here, the term 'things' is a piece of equipment consists of sensing, actuating, storage or processing capability. Gateway act as an intermediate block between the things and Cloud Infrastructure. To ensure smooth and secure flow, Network Infrastructure (NI) comes into an action to provide control over the information. Imbued with computing proficiencies and information storage, Cloud Infrastructure allows analytical and logical computing abilities. In addition to this, IOT devices must be comprised of a Physical Layer (PHY), an interface and an Internet Protocol (IP) address. Table 2 presents IOT building components, Associated devices and their features [3].

TABLE 2: IOT BUILDING COMPONENTS, ASSOCIATED DEVICES AND THEIR FEATURES

Ref.	IOT building components	IOT Associated Devices	Features
[3]	Things	Sensors and Actuators	Information collection and communication is possible from the objects, without any human intervention.
	Gateways	-	Dataflow becomes secure and manageable. It acts as an intermediate block and establishes strong connection between the things and cloud infrastructure.
	Network Infrastructure (NI)	Routers, Aggregators, Gateways and Repeaters	It provides control of data flow between things and cloud infrastructure
	Cloud Infrastructure (CI)	Virtualized Servers (VS) Data Storage Units (DSU)	It enables advanced computing with analytical and logical proficiencies.

B. IOT Architecture

Architecture is a skeleton that encompasses physical components with underlined principles. Effective IOT Architectures ensures best, fast, and reliable convergence of information technology. Different researchers have been proposed different architectures. Among those three-layer architecture is the basic one which consists of three layers - Perception Layer, Network Layer and Application Layer. Table 3-6 presents layers and their functionalities of three-layer architecture, five-layer architecture, Middleware Architecture and Service Oriented based Architecture.

TABLE 3: THREE LAYER ARCHITECTURE

Layers	Functions
Perception Layer	It is the ground layer and it deals with sensors and actuators.
Network Layer	It deals with the transmission and processing of information.
Application Layer	It ties to facilitate user with the application specific services. It is responsible to define various applications in which deployment of Internet of Things could be done.

TABLE 4: FIVE LAYER ARCHITECTURE

Layers	Functions
Perception Layer	It is the ground layer and it deals with sensors and actuators.
Transport Layer	It takes data from perception layer and transfers it to the processing layer through various mediums such as LAN, 3G, NFC, RFID and Blue tooth.
Processing Layer	It is responsible to take data from transport layer and then it stores, analyze and process huge amount of data. It provides services to subsequent layers and also manage services related to databases, big data and cloud computing.
Middleware Layer	It manages complete system and flow of data.
Application Layer	It is responsible to give interface to the user. It also presents final view of IOT.

TABLE 5: MIDDLE WARE ARCHITECTURE

Layers	Functions
Perception, Access, and Edge Layer	These layers deal with actuators and sensors.
Backbone Network Layer	It presents virtualized plane which consist of cloud and servers.
Middleware Layer	It is responsible to take data from transport layer and then it stores, analyze and process huge amount of data. It provides services to subsequent layers and also manage services related to databases, big data and cloud computing.
Coordination and Application Layer	These layers present application plane which provides interface to the user.

TABLE 6: SERVICE ORIENTED BASED ARCHITECTURE

Layers	Functions
Enterprise Layer	It manages to communicate with business processes sub-layer for Application Integration. It also communicates with Service Discovery, Service Election and Service Orchestration Layer.
Service Discovery, Selection and Orchestration Layer	These layers communicate with IOT Services, Cloud of Things and Cloud.

It maintains services and interrelationships between services. Each service is responsible to initiate messages from a process or service.

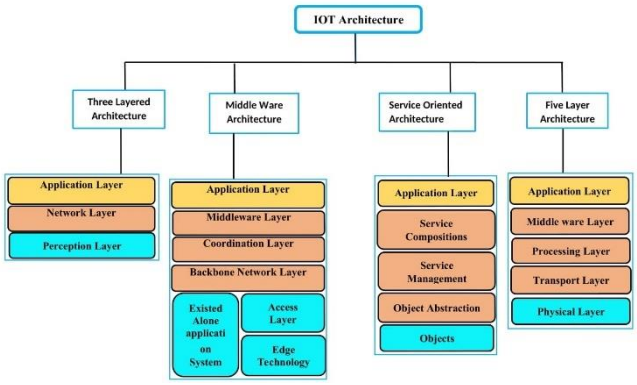

Fig. 2. IOT Architectures of Three Layer, Middle ware, Service Oriented and Five-layer Architectures.

C. IOT Taxonomy

IOT taxonomy is a way in which devices are composed to deliver their specific functionalities. In IOT taxonomy, we discuss those layers which are always supposed to be a part of IOT ecosystem. We begin our discussion with perception layer, which is composed of sensors and actuators, where sensors are responsible to collect data while actuators perform actions on that data. These sensors and actuators are further categorized as low-end, middle end and high-end devices. Sensors, actuators, and motes come under the category of low constrained IOT devices. Constrained technologies are deployed on data preprocessing layer. At this layer, there are some security features which filter the data before processing it further to the middleware.

IOT taxonomy defines IOT ecosystem and IOT ecosystem is divided in to six elements. Starting with IOT devices which exist in all layers of IOT architecture can be divided into open source and proprietary IOT devices. IOT devices have limited capability in terms of memory, power, and storage. IOT gateways mediate between sensing networks and high end IOT devices. Gateways are

responsible for collection of data from different sensors and then send data for high level processing.

Due to low constrained nature, IOT devices require efficient communication protocol to establish a network of devices. IOT devices cannot connect directly to the internet through IP stack because IP stack requires more power and here low power technologies like WSN, Bluetooth, Zigbee and WIFI came into existence. Fig. 3 presents elements of IOT.

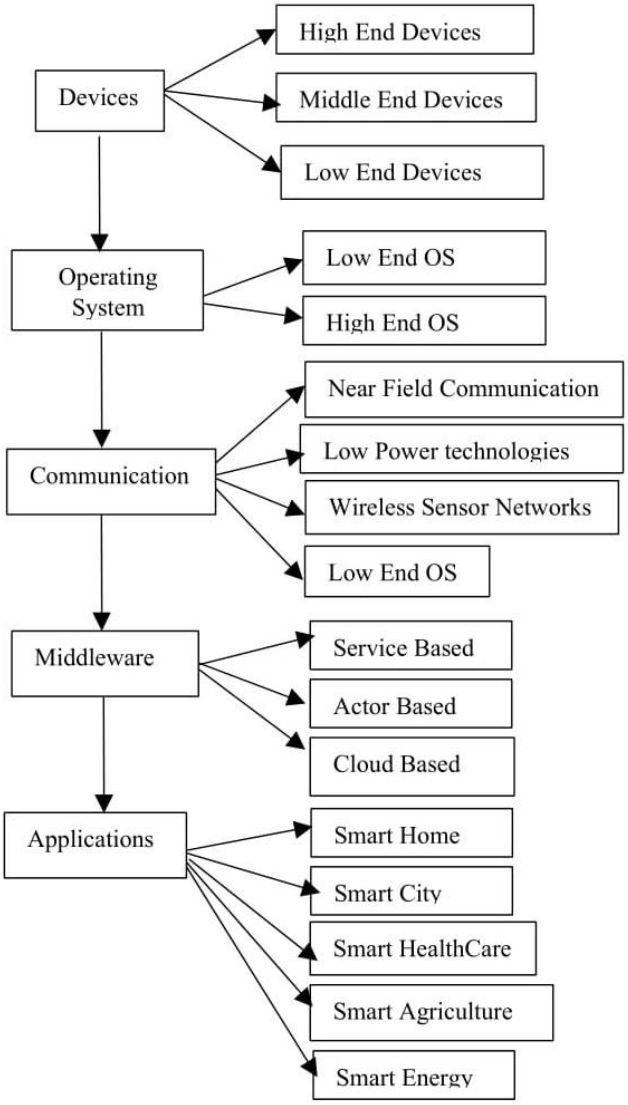

Fig. 3. IOT Elements.

IV. Major IOT Operating Systems

To make IOT application efficient, reliable, and scalable, an Operating system plays a vital role. IOT OS can be categorized as High-End and Low-End Operating Systems. High End OS operate on devices with single board systems for example: Raspberry pi and on the other hand Low End IOT OS acts as an interface for small board with constrained resources for example Arduino. High End and Low-End OS are further classified into Linux based and non-Linux based category. Fig. 4 presents categorization of IOT Operating System.

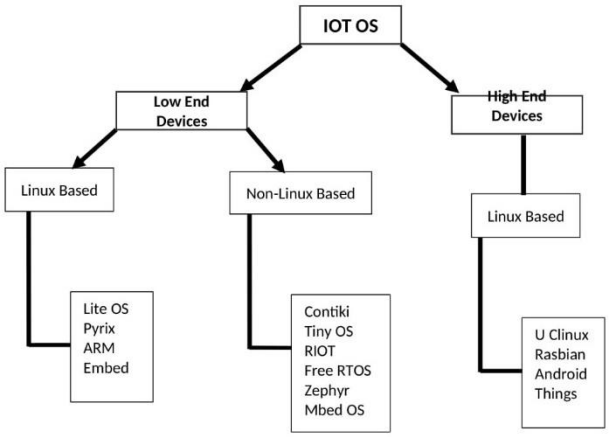

Fig. 4. Categorization of IOT Operating Systems.

A. *Contiki*

It is best suitable for low constrained devices. It was created by Adam Dunkels in 2002 and it was released under license of BSD as an open-source software. It supports light weight preemptive scheduling, and it is considered as most appropriate OS for low constrained devices due to its TCP/IP stack. It is developed with a modular architecture and it is written in C language. Hardware devices that are low constrained in terms of memory, power and communication bandwidth mostly go for Contiki OS. Moreover this, Contiki OS works on three network stack protocols, the ulP TCP/IP stack, which provides IPV4 networking, the ulPV6 stack, which provides IPV6 networking, and the Rime stack, which is comprised of light weight networking protocols especially designed for low power wireless networks.

B. *Tiny OS*

It is mainly aim for low power devices operate in wireless sensor networks (WSNs), ubiquitous computing, building automation and smart meters. It is written in nesC, which is a dialect of C programming language. It is an open-source software, and it was released under the license of BSD. Tiny OS provides interface to common abstractions like packet communication, routers, sensors, actuators, and storage.

C. *RIOT*

With a focus on low constrained devices, RIOT can be considered as a good choice. RIOT is developed by FU Berlin. It has micro kernel architecture, and it is written in C and C++. Its programming model is of Hybrid nature and it supports 6LoWPAN and real time scheduling.

D. *RIOT*

It is based on microkernel architecture and it is a small operating system for low constrained IOT devices. It is developed by Intel subsidiary wind driver. Its programming model supports multithreading, preemptive and non-preemptive scheduling. It has been developed in C and C++ programming language. It supports Blue tooth Low Energy (BLE) 5.0 because it provides network stack support with multiple protocols.

E. Mbed OS

It is based on monolithic kernel and it provides support for preemptive scheduling. It has written in C and C++ programming language. It is developed by ARM with the focus on low- constrained devices. It supports multithreading, 6LoWPAN, BLE, WiFi, Near Field Communication (NFC) and Radio Frequency Identification (RFID). Recently, it is considered as an epicenter to IOT research and development due to its multifaceted features.

F. Free RTOS

It is based on microkernel architecture and it provides support for preemptive priority based and cooperative scheduler. Its programming model features multithreading, mutexes and semaphores. It is written in C language and it also possesses set of assembly functions. It is built in tickles idle implementation. It uses Idle task hook which allows power saving and due to this nature, it can be considered for low powered IOT devices.

V. Architecture Types And Features Of IOT Operating Systems

Categorization of Architecture of IOT Operating System is defined as follows:

TABLE 7: Architecture Types

Architecture type	Features
Monolithic	• All processes run in kernel space. • Its faster in execution. • Easy to code this type of architecture. • Difficult to modify.
Microkernel	• Core Operating System services like scheduling, inter process communication and synchronization resides in kernel address space. • User services reside in user address space while OS core services reside in kernel address space. • Due to plugin availability, it provides flexibility.
Vm architecture	• Modification is easy. • Provide high level of portability. • Extensibility is high. • It is slow in execution.
Modular architecture	• Provides support for adding and replacing of components dynamically. • Each module presents separate functionality.
Layered architecture	• Easy to operate and handle. • It is designed for specific requirement. • Not flexible in nature.

A. Key features of IOT OS

Table 8 presents Key features of major IOT Operating Systems.

TABLE 8: Key Features of IOT OS

Operating System	Architecture	Scheduler	Programming Model	Programming Language	IOT Devices	OS Type	Ram (KB)	Rom (KB)
TINYOS	Monolithic	Non-preemptive FIFO	Event-driven concurrency	NesC	Low	Non-Linux	10	4–8
CONTIKI	Modular	Preemptive FIFO	Multithreading and event driven	C	Low	Non-Linux	2	40
RIOT	Microkernel	Preemptive priority based	Hybrid	C	Low	Non-Linux	1.5	5
FREERTOS	Microkernel	Preemptive priority based and cooperative scheduler	Multiple threads, mutexes, semaphore	C and assembly functions	Low	Non-Linux	10	12
ZEPHYR	Monolithic	non-preemptive and preemptive scheduling	Multithreading	C	Low	Non-Linux	2 to 8	50
MBED OS	Monolithic	preemptive	Multithreading	C and C++	Low	Non-Linux	4	16

VI. Network Stack Architecture Of IOT OS

A. Zephyr

Native Network stack consists of layers which provide specific support according to their own functionalities. These layers include:

NETWORK APPLICATION:

- This layer communicates with the provided application-level protocols for example CoAP, LWM2M, MQTT.
- This layer may access BSD Socket AI for network connection.
- This layer is responsible of data transmission and managing connections.
- It can also use network management API for the configuration of network and setting network link options.
- This layer sets IP address by using network interface API.

NETWORK PROTOCOLS:

- Provide implementation of application-level network protocols like CoAP, LWM2M and MQTT etc.
- It also provides support for core network protocols like IPV6, IPV4, UDP, TCP etc.

NETWORK INTERFACE ABSTRACTION:

It provides support that is common for all network like setting network interface down.

L2 NETWORK TECHNOLOGIES:

It implements API that is responsible for data communication to and from a device. These include Ethernet, Bluetooth and CANBUS etc.

NETWORK DEVICE DRIVERS:

Transmission of data packets over the net is taken care by low level device drivers.

B. Contiki

NETWORK STACKS:

It is comprised of three network stacks.

1.IPV6

2. IPV4

3. Rime

NETWORK LAYERS:

There are four layers.

1.Network layer

2. MAC (Medium Access Control) layer

3. RDC (Radio Duty Cycling) Layer

4. Radio Layer

NETWORK LAYER:

It is comprised of upper IPV6 layer and the lower adaptation layer.

MAC layer:

It is the simplest layer in IOT/IP stack. In case of any traffic, it helps in avoid collisions. It monitors the medium before sending and holds its operations when someone else is sending. This layer depends on RDC layer.

RDC (Radio Duty Cycling):

It provides facility of energy saving by keeping its radio transceiver off.

Contiki supports three cycling mechanisms: Contiki MAC, X-MAC, LPP (Low-Power Probing). These mechanisms are based on the principles of low power consumption and better power efficiency.

RADIO LAYER:

It is the ground layer in the Contiki Net Stack. Here interrupt handlers are responsible to fetch data. The input data is read into the buffer and then polling process sent the data to upper layers.

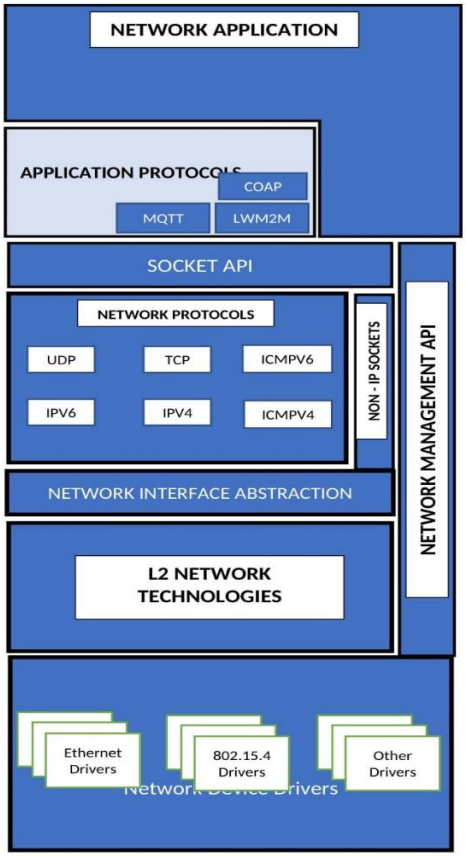

Fig. 5. Network Stack of Zephyr OS.

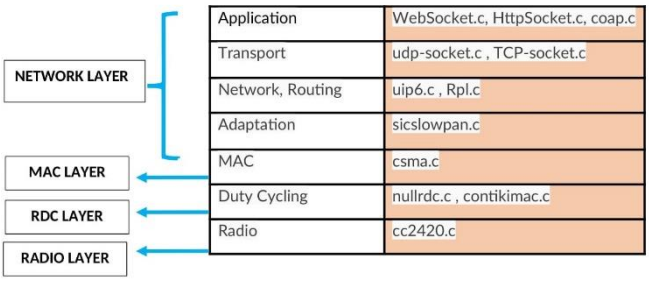

Fig. 6. Network Stack of Contiki OS.

C. Mbed OS

Network Stack of Mbed Os is divided in to three layers:

Transport Layer

Network Layer

Data link Layer

All IP connectivity methods shares Socket API. Socket API provides portability among various connectivity methods. Socket API relates to the transport layer and it supports TCP and UDP protocols. Network Stack layer supports IPV4 and IPV6. Data link relates to Network driver and it supports Ethernet, WiFi, Cellular and IEEE 802.15.4 drivers.

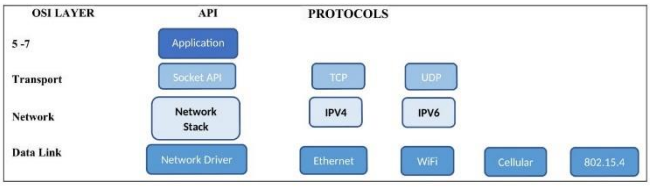

Fig. 7. Network Stack of Mbed OS.

D. RIOT

It is very flexible as any network can be integrated in its network architecture. It provides two interfaces: Application programming interface and the device driver API netdev [26]. Network stack architecture of RIOT is composed of six layers. Network layer is loosely coupled with the hardware layer. Communication between Network Layer and hardware layer is established by means of netdev API. Application layer communicates with the network stack by means of Sock API. Separate thread is assigned to each device driver. Driver layer provides implementation of radio devices.

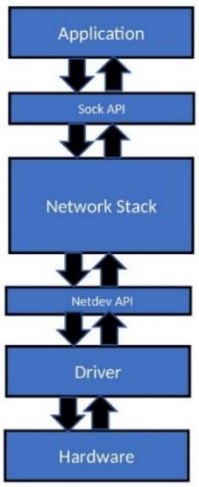

Fig. 8. Network Stack of RIOT OS.

E. Tiny OS

Prime component of Tiny OS in communication is Active message. Active message is an extensively used protocol in parallel computing where each message consists of an identifier which is user handler name. The handler function is responsible to invoke on the target node using user level handler name to pass the active message. This mechanism give rise to event driven communication between nodes.

F. Free OS

It relies on third party tools to implement MAC layer. For example, IoT LAB. In addition to this, TCP and UDP operate as transport protocols. Free RTOS UDP Protocol is mainly aim for low constrained devices as it provides socket with compact code size. On the other hand, Free RTOS TCP is based on open-source stack which provides Ethernet based stack. TCP/IP protocol is best suitable for low constrained devices because it is based on IwIP. IwIP is mostly used in embedded systems where less resource usage is in focus. Network stack implementation of IwIP is mainly aim for systems which have 10kb of RAM and 40kb of ROM.

VII. Related Work

By the passage of time Operating Systems for low constrained devices have emerged as predominant Operating Systems. This section discusses literature of previous studies and presents comparative analysis of this study with the existing surveys. Table 9 presents comparative analysis of this study with the existing surveys.

VIII. Open Research Issues And recommendations

A. Small Memory Footprint

To enhance the functionalities of low constrained devices more research is required in the area of small memory footprint. IOT devices operate on minimum RAM and ROM which contains few kilobytes. Hence, the core characteristic of low constrained IOT devices is to minimize the code size to ensure minimum memory utilization.

B. Less Energy Consumption

Sustenance in battery life of IOT devices is a key factor to reduce energy consumption. Designing of more efficient network protocols is highly required to ensure prolong battery life. In addition to this, more effort is required to manage hardware features in such an efficient manner which could result in less power consumption.

C. Reliability of IOT Devices

Another research direction led us to the reliability factor of IOT devices. While designing IOT complex architectures, OS reliability should be ensured by focusing on micro kernel architecture, memory protection units and static code analysis etc.

D. Scheduling Model

Sometimes scheduling model hit processor's performance by affecting system's energy and efficiency. Constraints during scheduling, affect processor's performance and due to this reason IOT Operating Systems are not able to perform up to the mark.

E. Runtime Behavior of RTOS

Extensive research is required to deal with complex run time behavior of RTOS task. In a complex IOT system, proper task priorities and processor shared timings are required. Hence, while implementing RTOS special care should be taken because in case of RTOS predicting real time behavior is very difficult. Tasks may get failed or may result in delay during execution and this low point of RTOS Operating System cannot be tolerated while implementing time critical applications.

TABLE 9: Comparative Analysis of This Study with the Existing Surveys

Discussed Aspects	This Study						Musaddiq, Arslan, et al. [7]						Bansal, et al. [5]						Javed, Farhana, et al. [9]					
	T	**C**	**R**	**F**	**Z**	**M**	**T**	**C**	**R**	**F**	**Z**	**M**	**T**	**C**	**R**	**F**	**Z**	**M**	**T**	**C**	**R**	**F**	**Z**	**M**
IOT OVERVIEW	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓			✓	✓	✓			
ARCHITECTURE	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓			✓	✓	✓			
TAXONOMY	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓			✓	✓	✓			
KEY CHARACTERISTICS AND FEATURES	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓			✓	✓	✓			
IOT DEVICES	✓	✓	✓	✓	✓	✓							✓	✓	✓	✓			✓	✓	✓			
PROGRAMMING MODEL	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓	✓			✓	✓	✓			
SCHEDULING MODEL	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓	✓			✓	✓	✓			
MEMORY	✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓	✓			✓	✓	✓			
NETWORK STACK ARCHITECTURES	✓	✓	✓	✓	✓	✓	✓	✓		✓														

C: Contiki, T: TinyOS, F: FreeRTOS, R:RIOT, Z:Zephyr, M:MbedOS.

TABLE 10: CONTRIBUTIONS AND FUTURE DIRECTIONS OF EXISTING SURVEYS

Ref.	Discussed IOT Operating Systems	Discussed aspects	Future directions
[6]	TinyOS, Contiki, RIOT, Zephyr, MbedOS and Brillo	In this paper, overview of different IoT Operating systems, supported hardware, and future research directions are presented. Moreover this, this study provides overview of the previous literature papers in Special Issue on IoT OS management: opportunities, challenges, and solution. Finally, this study concluded the whole survey.	• Future work should be focused on efficient techniques to acquire the acute motes of synchronization. • RDC (Radio Duty Cycling) can be another direction of research along with motes synchronization to work on. • More work is needed to achieve accuracy in execution of critical tasks of the IOT motes to enhance real time capabilities. • Critical systems such as health care, smart home, smart city is the flavor of time and their security is still a question mark.
[7]	Contiki, TinyOS, and FreeRTOS,	Different aspects of resource management including process management, memory management, energy management, communication management, and file management are investigated, and their advantages and disadvantages are presented.	• Efficient mechanism is required to utilize the minimum memory. • Reliability of IOT devices requires more research. • Real time operating systems and execution of tasks without delay in real time is a great challenge. • Scheduling model constraints, limitations in network buffer management and programming model could be considered as future research direction. • More operating systems need to be explored for future research such as Zephyr, RIOT, Mbed OS.
[8]	FreeRTOS, Mbed, Contiki, TinyOS and RIOT	This study is mainly focused on comparative analysis of the most recent IOT operating systems for low constrained IOT devices. This study discussed architecture, scheduling, real-time capabilities, programming model, memory footprint, network connectivity, hardware support and energy efficiency.	• IOT operating Systems are still deficient in context of security. • Reduction in Operating System services is required in light Kernel architecture and major focus should be drawn towards small memory footprint. • Execution of tasks and real time compatibility should be equated in order to achieve accuracy.
[9]	Contiki, TinyOS, RIOT, Nano-RK, LiteOS, MantisOS,ROS OS, RETOS	This study addressed IOT operating Systems constraints with respect to their architecture, programming model, schedular algorithms, networking and communication protocols. This study presented requirements and shortcomings in development. In addition to this, it also contributed towards summary of related work and detailed case studies are also illustrated.	• IOT operating Systems are still deficient in context of security. • Reliable Communication, Bandwidth, Interoperability need to be addressed in detail for future challenges. • Small memory foot print needs to be addressed to facilitate low constrained IOT devices.
[10]	Android Things, Mbed OS, Contiki OS, RIOT OS, Zephyr	This study identified key parameters that needs to be focused on while selecting open-source project. Number of selected studies have been discussed in this survey to explore open-source system software projects and frameworks.	• Security and Privacy are still open to research. • Significant research is required to stable communication protocols. • Security concerns need to be monitored such as bug fixes or updates and hardware constraints.
[11]	Tiny OS, Contiki OS, FreeRTOS, and RIOT.	This study contributes in the dimensions of over view and evolution of IOT. Architectures of major IOT Operating systems are discussed. In addition to this, challenges of IOT Operating Systems and open research issues are discussed extensively.	• Small memory foot print is a big challenge to address while proposing new architecture for IOT. • Challenges in real time operating systems defines new direction for research. • Execution of tasks and real time compatibility should be equated in order to achieve accuracy.
[12]	Contiki TinyOS mbed OS Real Time Operating System (RTOS)	A comprehensive overview of IOT OS was discussed in this paper. Memory, Programmability Support, Network Protocols, Architecture, Schedular, Modularity Support is discussed in view of major OS.	• Scheduling model constraints, limitations in network buffer management and programming model could be considered as future research direction. • IOT operating Systems are still deficient in context of security.
[14]	Contiki, RIOT, TinyOS, LiteOS, FreeRTOS, Mantis OS, Nano-RK, SOS, NutOS, uC/OS-III, uClinux	This explanatory paper presents content on prevalent IOT operating systems. This study did comparative analysis and illustrate findings for future studies.	N/A
[5]	TinyOS, Contiki, RIOT, LiteOS, FreeRTOS, Mynewt, uClinux, Raspbian, Android thing	This study serves to elucidate taxonomy of IOT ecosystem. Many technical aspects are illustrated such as architectures, devices, communication protocols and network stack architectures.	• IOT ecosystem is established with heterogeneous devices that work together and allow inter device communication. More research is required for heterogeneity among these devices. • Security and Privacy are still open to research. • Scheduling model constraints, limitations in network buffer management and programming model could be considered as future research direction.
[15]	Contiki, RIOT, FreeRTOS .TinyOS, OpenWSN nuttX , eCos uClinux ,ChibiOS/RT CoOS, nanoRK,Nut/OS	Contribution of this study lies in exegetic discussions on specific requirements that should be fulfilled by an OS to cater low constrained devices. Several tradeoffs have been discussed regarding the constrains of IOT and hardware platforms.	N/A

IX. CONCLUSION

This paper provides an overview of IOT and its building components. This paper gives insights into architecture, programming model, schedular and network stack architecture of different IOT Operating Systems. The contributions are multifaceted. Firstly, we discussed contribution to this survey followed by the structure of this survey. Peer review papers are presented along with year of publication and authors. Secondly, an overview of IOT is presented along with IOT building components. IOT architecture is explained along with layers and its functionalities. In this section we discussed IOT elements and major IOT Operating Systems for low constrained devices with their key characteristics. Network Stack architecture of IOT OS is also given in this section. After this, we discussed related work and compare our work with the previous surveys. Finally, critical research areas are unfolded to facilitate future research studies in this domain.

ACKNOWLEDGMENT

The author is grateful to her instructor Dr. Muhammad Iqbal, Assistant Professor, Department of Computer Science, Bahria University Karachi, Pakistan for his immense support throughout this study.

REFERENCES

[1] NortonLifeLock. (2019, August 28). https://us.norton.com/internetsecurity-iot-5-predictions-for-the-future-of-iot.html. Retrieved from NortonLifeLock: https://us.norton.com/

[2] Lueth, K. L. (2018, August 8). Retrieved from IOT Analytics: https://iot-analytics.com/state-of-the-iot-update-q1-q2-2018-number-of-iot-devices-now-7b/

[3] Kumar, N. M., & Mallick, P. K. (2018). The Internet of Things: Insights into the building blocks, component interactions, and architecture layers. Procedia computer science, 132, 109-117.

[4] Baccelli, E., Gündoğan, C., Hahm, O., Kietzmann, P., Lenders, M. S., Petersen, H., ... & Wählisch, M. (2018). RIOT: An open-source operating system for low-end embedded devices in the IoT. IEEE Internet of Things Journal, 5(6), 4428-4440.

[5] Bansal, S., & Kumar, D. (2020). IoT Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication. International Journal of Wireless Information Networks, 1-25.

[6] Zikria, Y. B., Kim, S. W., Hahm, O., Afzal, M. K., & Aalsalem, M. Y. (2019). Internet of Things (IoT) operating systems management: opportunities, challenges, and solution.

[7] Musaddiq, A., Zikria, Y. B., Hahm, O., Yu, H., Bashir, A. K., & Kim, S. W. (2018). A survey on resource management in IoT operating systems. IEEE Access, 6, 8459-8482.

[8] Sabri, C., Kriaa, L., & Azzouz, S. L. (2017, October). Comparison of IoT constrained devices operating systems: A survey. In 2017 IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA) (pp. 369-375). IEEE.

[9] Javed, F., Afzal, M. K., Sharif, M., & Kim, B. S. (2018). Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review. IEEE Communications Surveys & Tutorials, 20(3), 2062-2100.

[10] Amiri-Kordestani, M., & Bourdoucen, H. (2017). A survey on embedded open source system software for the internet of things. In Free and Open Source Software Conference (Vol. 2017).

[11] Shammar, E. A., & Zahary, A. T. (2019). The Internet of Things (IoT): a survey of techniques, operating systems, and trends. Library Hi Tech.

[12] Chandra, T. B., Verma, P., & Dwivedi, A. K. (2016, March). Operating systems for internet of things: A comparative study. In Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies (pp. 1-6).

[13] Srinidhi, N. N., Kumar, S. D., & Venugopal, K. R. (2019). Network optimizations in the Internet of Things: A review. Engineering Science and Technology, an International Journal, 22(1), 1-21.

[14] Gaur, P., & Tahiliani, M. P. (2015, May). Operating systems for IoT devices: A critical survey. In Proceedings of the 2015 IEEE Region 10 Symposium (pp. 33-36).

[15] Hahm, O., Baccelli, E., Petersen, H., & Tsiftes, N. (2015). Operating systems for low-end devices in the internet of things: a survey. IEEE Internet of Things Journal, 3(5), 720-734.

[16] Silva, M., Cerdeira, D., Pinto, S., & Gomes, T. (2019). Operating Systems for Internet of Things Low-End Devices: Analysis and Benchmarking. IEEE Internet of Things Journal, 6(6), 10375-10383.

[17] Dunkels, A., Gronvall, B., & Voigt, T. (2004, November). Contiki-a lightweight and flexible operating system for tiny networked sensors. In 29th annual IEEE international conference on local computer networks (pp. 455-462). IEEE.

[18] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., ... & Culler, D. (2005). TinyOS: An operating system for sensor networks. In Ambient intelligence (pp. 115-148). Springer, Berlin, Heidelberg.

[19] Dunkels, A., Schmidt, O., Voigt, T., & Protothreads, M. A. (2006). Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems In Proceedings of the Forth International Conference on Embedded Networked Sensor Systems.

[20] Alliance, T. (2008, November). TinyOS 2.1 adding threads and memory protection to TinyOS. In Proceedings of the 6th ACM conference on Embedded network sensor systems (pp. 413-414).

[21] Lindgren, P., Mäkitaavola, H., Eriksson, J., & Eliasson, J. (2012, October). Leveraging TinyOS for integration in process automation and control systems. In IECON 2012-38th Annual Conference on IEEE Industrial Electronics Society (pp. 5779-5785). IEEE.

[22] Goyette, R. (2007). An analysis and description of the inner workings of the freertos kernel. Carleton University, 5.

[23] Déharbe, D., Galvão, S., & Moreira, A. M. (2009). Formalizing FreeRTOS: First Steps, Formal Methods: Foundations and Applications: 12th Brazilian Symposium on Formal Methods, SBMF 2009 Gramado, Brazil, August 19-21, 2009 Revised Selected Papers.

[24] Ferreira, J. F., Gherghina, C., He, G., Qin, S., & Chin, W. N. (2014). Automated verification of the FreeRTOS scheduler in Hip/Sleek. International Journal on Software Tools for Technology Transfer, 16(4), 381-397.

[25] Cekerevac, Z., Dvorak, Z., & Pecnik, T. TOP SEVEN IoT OPERATING SYSTEMS IN MID-2020.

Sumera Rounaq received BS degree in Software Engineering from UBIT, Karachi University, Pakistan. Currently, she is pursuing MS in Computer Science from Bahria University, Karachi Campus, Pakistan. Her research interests include IOT Operating Systems in Low end devices, Detection of patterns using Machine Learning, Deep Learning, Natural Language Processing and extraction and analysis of Data using Data Science.

Dr. Muhammad Iqbal is a Senior Assistant Professor and Cluster Head of Department of Computer Science, Bahria University, Karachi Campus, Pakistan. He received Ph.D. degree from South West Jiao tong University, Chengdu, China (SWJTU). He has 14 international peer-reviewed publications on his credit.