Internet of Things (IoT) Operating Systems Support, Networking Technologies, Applications, and Challenges: A Comparative Review

Farhana Javed, Muhamamd Khalil Afzal, *Senior Member, IEEE*, Muhammad Sharif, and Byung-Seo Kim, *Senior Member, IEEE*

***Abstract*—The Internet of Things (IoT) has become a reality. As the IoT is now becoming a far more common field, the demand for IoT technologies to manage the communication of devices with the rest of the world has increased. The IoT is connecting various individual devices called things and wireless sensor networks is also playing an important role. A thing can be defined as an embedded device based on a micro controller that can transmit and receive information. These devices are extremely low in power, memory, and resources. Therefore, the research community has recognized the importance of IoT device operating systems (OSs). An adequate OS with a kernel, networking, real-time capability, and more can make these devices flexible. This review provides a detailed comparison of the OSs designed for IoT devices on the basis of their architecture, scheduling methods, networking technologies, programming models, power and memory management methods, together with other features required for IoT applications. In addition, various applications, challenges, and case studies in the field of IoT research is discussed.**

***Index Terms*—Internet of Things, operating system, wireless sensor networks, real-time operating system, IPv6 over low power personal area networks, radio frequency identification.**

I. Introduction

THE INTERNET has changed almost every aspect of our lives: how we work, think, educate, and entertain ourselves, and now the time has arrived for the Internet of Things (IoTs). This will connect more devices and impact our lives more than any other aspect of the digital age has done before [1]. IoTs represents the evolution of mobiles, homes and embedded systems. Such an evolution will help to create a smart world for people because the objects around us will have better knowledge of our likes, wants, and needs.

Manuscript received August 7, 2017; revised December 19, 2017 and February 26, 2018; accepted March 12, 2018. Date of publication March 21, 2018; date of current version August 21, 2018. This work was supported in part by the National Research Foundation of Korea (NRF) grant funded by the Korea Government under Grant 2018R1A2B6002399, and in part by Basic Science Research Program through the NRF funded by the Ministry of Education under Grant 2015R1D1A1A01059186. *(Corresponding author: Byung-Seo Kim.)*

F. Javed, M. K. Afzal, and M. Sharif are with the Department of Computer Science, COMSATS Institute of Information Technology, Islamabad 47040, Pakistan (e-mail: farhana.javed1@gmail.com; khalilafzal@ciitwah.edu.pk; muhammadsharifmalik@yahoo.com).

B.-S. Kim is with the Department of Software and Communications Engineering, Hongik University, Sejong 339-701, South Korea (e-mail: snbs@hongik.ac.kr).

Digital Object Identifier 10.1109/COMST.2018.2817685

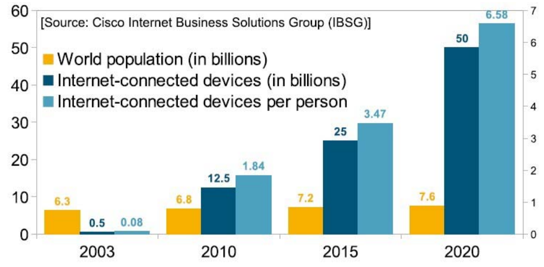

Fig. 1. The Growth of Interconnected Devices by the Year 2020 (adapted from [2]).

Every IoTs device should have a physical layer (PHY), an interface, and an Internet protocol (IP) address. When these devices and systems share data over the cloud and analyze it they can transform our businesses, lives, and world in countless ways. The IoTs involves a growing economy. The IoTs has grown enormously since it was first introduced in the late 1990s [2]–[4] and it is expected to grow still more in the future. It will connect to the world wide Web far more than happens today, with the devices involved ranging from smart phones, automated teller machines (ATMs) to industries, their products, and shipping containers [5]. According to [2] Figure 1 shows the anticipated growth of IoTs devices relative to the population by the year 2020.

A. Motivation: Need for an Adequate Operating System in IoTs

The things which make IoTs intricate involve scheming and construction of the systems itself [6], [7]. To ensure the veracity of the IoTs, a device connected to it should be able to communicate with the rest of the world, and for this purpose wireless technology is the first choice [8]–[10]. The fundamental role of the system (OS) is to hide the minor details of the device. Old-fashioned OSs are restrained, such as Linux and Berkeley software distribution (BSD). OSs for tiny devices should provide services, such as the management of resources, while the processor should also have opportune scheduling policies. The goals to be achieved by an IoTs OS include multitasking, security, and interaction. Currently, the IoTs is bedeviled because of a lack of inter-operability between the many incompatible solutions. Consequently, some profound solutions are required for the IoTs and its OSs [11].

Unlike personal laptops, the IoTs demands a large number of nodes. These nodes will be connected to gateways. Furthermore, the devices will be connected to a remote cloud platform. The functions of sensor nodes and their form factors and target applications vary considerably and create a context for the suitability of any specific operating system for IoTs implementations. Therefore, the key requirements for the IoTs OS can be a small memory footprint, real-time, energy efficiency, hardware agnostic operations, security, networking, protocols support, data storage, reliable communication, and end device management. The multiple characteristics required of an OS can be organized into various categories. The first key group of characteristics is that the architecture of the kernel must be either monolithic, micro-kernel, or layered. Second, is that there must be a scheduling policy for the tasks assigned to the processor. Third, and crucial, is that the programming model that can be either multi-threaded or event based. Also, the programming model will help to determine the programming language for the OS. Moreover, the reprogramming of the devices is significant because these devices can be placed far away, i.e., in a forest or on a highway, where they are not easily reached. Therefore, dynamic reprogramming is required [11]–[13]. Data passes through each system via a node to a node and these nodes are usually low power devices. The structure of the IoTs is made up of various component including sensors, power management devices, amplifiers, micro-controllers, processors, integrated circuits (ICs), low power radio frequencies (RF) and more. Various companies are working toward building micro-controller units (MCUs) and require processors that have wireless fidelity (WiFi) and fiber to the home (FTTH) facilities.

Sensors collect information from the PHY environment and convert it into a form readable by humans. Two major parts of the IoTs network are the digital back-end and the front-end. The front-end is the most obvious domain, which provides efficient and innovative ideas for users, whereas the back-end is the device-to-Internet cloud communication. IoTs sensors are used with the following inputs: ambient light, optical, temperature, inertia, humidity, gesture, proximity, touch, and fingerprints. Micro-electro-mechanical systems (MEMS) [14] have sensors for all these various purposes: ZigBee [15], [16], radio frequency identification (RIFD) [9], [17] and IP version 6 (IPv6) [18], [19] for communication. Some of the best-known hardware electronic platforms for IoTs include Arduino, Raspberry Pi [20], [21], The BeagleBone Black [22], The Intel Galileo [23] and The pcDuino [24].

The Internet engineering task force (IETF) defines these constraints in various terms such as cost and physical constraints, the latter including power, memory processing resources and processing cycles, code space, and energy and network optimization [25], [26]. IETF categorizes devices into three major classes based on their data size and code size. Class 0 devices are extremely constrained and therefore, accurate consideration of the software is required, and it should be hardware specific. Classes 1 and 2 are moderately constrained, and therefore, the new ideas that emerge are generally focused on these device classes. Details of mostly used devices including their random access memory (RAM), read-only memory (ROM), central processing unit (CPU), size in kilobyte (KB) or megabyte (MB) network interface, class, and OS are given in Table I. Whereas, Table II defines abbreviations generally used in this review.

TABLE I
MAJOR HARDWARE PLATFORM OF IoTs

Name	**Hardware Feature**	**Network Interface**	**Class**	**OS**
Aurdriono [25]	RAM:64MB, ROM:16MB, CPU:Atheros AR9331	IEEE 802.3/802.11	Class 2	-
Raspberry [18]	RAM:512MB, CPU:Broadcom, BCM2835, ARM11	IEEE 802.3/802.11	Class 2	-
Intel	RAM:256MB, CPU: SoC X, Intel, Quark X1000	IEEE 802.3	Class 2	-
Zolertia Z1 [24]	RAM:8KB, ROM:92KB	IEEE 802.15.4	Class 1	Contiki

B. Contributions of This Survey

The outline of the contributions of this paper relative to the recent literature in the field can be summarized as:

- Compared to other survey papers in the field, this survey provides a deeper summary of the most relevant key features of an OS for IoTs and its applications.
- This review addresses the IoT's constraints in term of its OS with respect to their hardware platforms, design choices, networking and communication technologies.
- We present collected information about the solutions for IoT's application, understanding how technologies are used in the IoTs for developed OS.
- We provide an overview IoT's communication and networking technologies moreover, we review OS according to their implemented technologies for communication.
- We review IoT's solutions to identify developments, production requirements, shortcomings, and innovation prospects.
- We provide an overview of some of the key IoTs challenges presented in the recent literature and provide a summary of related research work.
- We also present detailed case studies to illustrate the different OS delivering desired IoTs services.

C. Review of Related Survey

Our present survey on IoT's OSs in the context of its hardware support, communication and networking technologies and applications is different from previous magazine articles and surveys as we comprehensively cover the area of OS for IoTs. There is an extensive literature of prior surveys that focus on OS for IoT's low-end devices. Also, some recent surveys discuss OS in the context of features. However, to the best of our knowledge, there is no prior detailed survey that comprehensively covers the discussion of OS in the context of its applications and communication technologies. There is an extensive survey literature on OS for IoTs [25]. It covers

TABLE II
ABBREVIATIONS

Symbol	Description
ADRS	Adaptive Double Ring Scheduling
6Lowpan	IPv6 over Low power Wireless Personal Area Networks
ADRS	Adaptive Double-Ring Scheduling
API	Application Program Interface
BSD	Berkeley Software Distribution
CoAP	Constrained Application Protocol
CS	Contention Slots
DIP	Density Inference Protocol
DM	Deadline Monotonic
DNL	Dynamic Network Layer
DTLS	Datagram Transport Layer Security
EATT	Energy-aware target tracking
EDF	Earliest-Deadline-First
ELF	Executable linkable format
FIFO	First In First Out
HAL	Hardware Adaptation Layer
HPL	High power listening
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IoTs	Internet of Things
LEACH	Low-energy adaptive clustering hierarchy
LLF	Least Laxity First
LoC	Line of Code
MAC	Medium Access Control
MLD	Multicast Listener Discovery
MLL	Mac and Data Link Layer
MOAP	Multi-hop Over air reprogramming
NSL	Networking supporting Layer
OS	Operating System
PDA	Personal Digital Assistant
PHY	Physical
POSIX	Portable Operating System Interface
PRR	Priority Round Robin
QOS	Quality of Service
RF	Radio Frequency
RFID	Radio Frequency IDentification
RM	Rate Monotonic
RPL	Routing Protocol for low power
RR	Round Robin
RTOS	Real-time Operating System
RTS	Real-Time Scheduling
SPIN	Sensor Protocol for Information via Negotiation
SRM	Scalable Reliable Multicast Protocol
SS	Scheduled slots
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UDP	User Datagram Protocol
VDI	Virtual Desktop Infrastructure
WSNs	Wireless Sensor Networks

low-end devices, scheduling policies and networking stack. Architecture, scheduling and real-time features of OS have been discussed in [26]. Energy efficiency and protocols have been discussed in [27]. Some recent survey articles [28], [29] give some insight on key features.

Table III summarizes the contributions to this survey as drawn from recent literature in the field. All the reviews mentioned in Table III, focus on an introduction and overview of the features. We feel that there is a need for more than an introduction. We need to pay more attention to the features and requirements that can make a particular OS a better choice for the IoTs environment. Therefore, many challenging issues still need to be addressed.

D. Structure of Survey

The main objective of this comparison is to give the reader the opportunity to understand what has been done (in relation to the kernel, networking technologies, protocols, algorithms, efficiency methods) in the field of IoTs for all the OSs designed and what still remains to be addressed in each OS.

The remainder of this survey paper is organized as follows. In Section II, we introduce and briefly describe various OS for the IoTs paradigm, which are available from the literature. IoT's requirement for kernel design and architecture choice for OS is presented in Section III. Section IV gives a detail discussion of real time capabilities required for IoTs and therefore, multiple scheduling algorithm which are proposed and adapted by OS to provide help for real time tasks are reviewed. Section V takes a close look at all the programming models of OS and their programming language and how an OS can provide ease to developer while developing application for IoTs. In Section VI, we review, communication protocols and methods for reprogramming devices of IoTs. Networking stack of IoTs, its technologies and protocols are presented in Section VII. Then, we discuss power management and memory management of OS required for IoTs applications and devices in Sections VIII and IX. In Section X, we review a selected number of simulators for the OS and why they hold an importance for development. In Section XI, ten major applications of IoTs their brief description, and OS role in their development is reviewed. Consequently, we present limitation and challenges of IoTs based on evaluation results, in Section XII and before closing in Section XIII we have some honorable mention of OS for IoTs. Finally, we present the conclusion at the end of this review.

II. Major OSs for IoTs: An Overview

In previous years, remarkable ideas regarding IoTs applications have been proposed. For IoTs applications, various software and OSs have been presented. Some major OSs of the IoTs are discussed below:

A. Contiki

Contiki [30], [31] was an early development of WSNs, later with a few improvements now it is used on a platform of IoTs and it has a well-known reputation. Since its publication in 2004, it has come far because of its clever features. Contiki has had various versions and is licensed under Berkeley software distribution (BSD). Recently, the Contiki has been developed with a modular architecture style. Preemptive multi-threading scheduling is used and is written in C language. Contiki implements Rime as its networking stack and aims for high efficiency in power and memory management. Its support for IPv6 over low power wireless personal area networks (6LoWPAN) [32] has made Contiki famous as well. In some applications, Contiki has already been declared a winner, and at this point in time it is the most frequently used OS for IoTs.

TABLE III
COMPARATIVE OVERVIEW OF EXISTING REVIEWS

Discussed Features	**This Review**	**Hahm, (2015) [25]**	**Dong, (2010) [26]**	**Farooq, (2011) [27]**	**Chandra, (2016) [28]**	**Gaur, (2015) [29]**
Supported Devices	✓	✓	×	×	×	×
Architectures	✓	×	✓	✓	✓	✓
Scheduling Policies	✓	✓	✓	✓	✓	✓
Real-time Support	✓	×	✓	✓	✓	×
Scheduling Algorithms	✓	×	×	×	×	×
Programming Models	✓	×	×	×	×	✓
Programming Languages	✓	×	✓	×	×	✓
Network Stack	✓	✓	✓	×	✓	✓
Protocols	✓	×	×	✓	×	✓
Reprogramming Techniques	✓	×	✓	×	×	×
Memory Management	✓	×	×	✓	×	×
Energy Efficiency	✓	×	×	×	×	×
Simulators	✓	×	×	×	✓	×
Licenses	✓	✓	×	×	×	×
New Version	✓	✓	×	×	×	×
Documentation	✓	✓	✓	×	×	×
Shell	✓	×	×	×	×	×
Database	✓	×	×	×	×	×
Testing	✓	✓	×	×	×	×
Debugging	✓	✓	✓	×	×	×
Open/Close Standard	✓	✓	×	×	×	×
Code Development	✓	✓	×	×	×	×
Application	✓	×	×	×	×	×
Challenges	✓	×	×	×	×	×

B. TinyOS

TinyOS is called the *defacto* OS for WSNs. However, due to its extraordinary features, such as support for various devices and its programming ease, it can also be used for IoTs. TinyOS supports monolithic architecture and an event-driven programming model. It has various scheduling techniques and multiple algorithms. Moreover, TinyOS has its own programming language, NesC, originally derived from the C language. Through NesC, it has been able to improve its efficiency and management techniques for both power and memory. It also has its own active message mechanism for communication over a network. Contiki and TinyOS are together the dominant players in the field [33].

C. RIOT

In contrast to TinyOS and Contiki, RIOT [34], [35] is directly aimed at the IoTs and has done its best to prove itself in this area. RIOT has multiple features, such as its support for 6LoWPAN and real-time scheduling, which make it suitable for IoTs. RIOT has come into the spotlight in recent years because it came out in 2013 and was licensed under a less general public license (LGPLv2). RIOT has micro-kernel architecture and supports real-time scheduling because of its multi-threading model and its networking support. It has been developed in C and C++ programming languages [36].

D. Nano-RK

Nano-RK [37], [38] is a fully preemptive reservation-based real-time operating system (RTOS) from Carnegie Mellon University with multi-hop networking support for use in WSNs. Nano-RK is also famous for its unique method of resource reservation. Nano-RK came under the spotlight around 2005. Since then, it has been improved and has stayed on top because of its efficient methods of energy and memory management. Nano-RK is much like TinyOS because of its monolithic architecture and event-driven system. However, it supports real-time applications and is written in C, making it easier for developers to use.

E. LiteOS

LiteOS was developed in 2008 and since then has been fighting for popularity in this area because it has a unique modular architecture and kernel; it uses a programming language known as LiteC++. It implements both event and threads in its programming model, however, the scheduling of tasks is “run to completion”. It is an open source OS for IoTs and can be considered for various applications in the IoTs field [39].

F. MantisOS

MantisOS [40] is another OS released in 2005 and licensed under BSD [41]. MantisOS has a different layered architecture and implements threads as its programming model system. It is written in C language. It has a method called “comm” for networking support in IoTs. It implements resource sharing as semaphores.

G. SOS OS

SOS is an OS providing features like reprogramming, and it implements various protocols for this purpose. SOS was developed in 2005. SOS uses a message passing technique between modules for interaction. It has a modular architecture and its event-based programming model is written in C language. It is also a dynamic OS which means run-time changes can be made in an application [12], [42].

TABLE IV
ARCHITECTURE FOR IOTS

Architecture Design for IoTs	**Features**						**Operating Systems**							
	Memory footprint	Development Cost	Simplicity	Reliability	Flexibility	Portability	Contiki [61] [62]	TinyOS [63] [64]	RIOT [65] [66]	Nano-RK [67]–[69]	LiteOS [70] [71]	MantisOS [72]–[74]	SOS [75] [76]	RETOS [77] [78]
Monolithic	max	max	min	min	min	-	-	✓	-	✓	–	–	–	–
Microkernel	max	min	min	max	max	–	–	–	–		–	–	–	–
Modular/Hybrid	max	max	–	min	max	–	✓	–	✓	–	✓	–	✓	✓
Exokernel	–	–	–	–	max	–	–	–	–	–	–	–	–	–
Layered	max	–	max	max	min	–	–	–	–	–	–	✓	–	–
VirtualMachine	–	–	–	–	max	max	–	–	–	–	–	–	–	–

H. Resilient, Expandable, and Threaded OS (RETOS)

RETOS [43], [44] was developed in 2007 and it is an open source OS. It has a modular architecture and implements multi-threading as its programming model. It supports real-time to some extent with its portable operating system interface (POSIX) threads [45]. It has a network stack which can be either static or dynamic. As a collective, with dynamic operating, RETOS is getting a lot of attention due to its networking stack structure.

I. Other Renowned OS

There are widely known OS which include OpenWSN because it implements 6iTCH [46] as its network stack and gets a lot of attention in IoTs for its features. However, Android [47] and Brillo [48], despite the critical acclaim, they fail to keep pace. Some other open source OS are: nuttX [49], mbedOS [50], L4 microkernel family [51], uClinux [52], ChibiOS/RT [53], ERIKA Enterprise [54], MansOS [55], and NanoQplus [56]. Furthermore, some closed OSs which support the IoTs are: QNX [57], VxWorks [58], PikeOS [59], and embOS [60].

III. ARCHITECTURE TYPES AND FEATURES OF OS FOR THE IOTS

In the IoTs, the OS of a device determines its structure. The architecture of an OS has enormous influence on the size of the kernel as it provides services to the application programs. The architecture of the OS can be categorized as either monolithic, micro-kernel, virtual machine, or layered architecture. Monolithic architecture is a combination of necessary OS components and applications. Services are implemented separately and each service has an interface for another service. The monolithic approach results in an underprivileged design choice for the OS. Micro-kernel architecture provides minimum functionality in the kernel. The application and OS are built as a set of interacting modules. Therefore, the kernel size is reduced. Another type of OS architecture is the virtual architecture, which works on the principle that a virtual machine is exported to user programs, which resemble hardware. Table IV shows a comparison of all available architectures and their major features. IoTs require an OS to have a small kernel size, a small memory footprint, reliability, and ease of extension of the kernel to boost flexibility [26].

A. Architecture of Contiki

Contiki uses a modular architecture. It is a combination of both the thread and event model. A modular approach is the design choice for a multiple embedded OS due to its small kernel size and the context switching of a traditional WSNs application. Modular architecture provides minimum functionality inside the kernel which results in a reduction in kernel size. This OS provides functionality via user-level servers as a memory, file, and time server. This gives the advantage that the whole system does not crash if one server fails. At the kernel level, Contiki is primarily an event-driven OS. However, it supports multi-threading to provide optional threading facilities for each process. This provides optional application-level libraries for applications where they need multi-threading. This architecture provides better reliability, ease of extension, and customization of its user-level servers as a memory, file, and time server. Contiki's kernel consists of a lightweight event scheduler that dispatches events to running processes and occasionally calls the processors as polling handlers. Every process execution is prompted either by events dispatched by the kernel to the processor or by a polling mechanism [79], [80]. These polling mechanisms are used to avoid a race situation and the kernel does not trigger an event handler once it has been scheduled, which is the reason any scheduled event will run to its completion. However, preemption can be attained by an event handler using an internal mechanism. Two kinds of events are supported by the kernel: asynchronous events are more like a deferred procedure call that are acquired by the kernel and are dispatched later to the target process after some delay. Synchronous events are similar to asynchronous however, they are dispatched immediately to the target process, causing it to be rescheduled. The control returns to the posting process after the task processing is completed. This can be seen as an inter-process procedure call. Moreover, the kernel of Contiki provides a procedure polling mechanism and this is evident as a high priority event that is scheduled between the asynchronous events. After a poll is scheduled, all programs that implement a poll handler are called in order of their priority. The Contiki's kernel uses a single shared stack for all its executions. Asynchronous events reduce stack space requirements as the stack is rewound between each invocation of event handlers. Processes which use the polling are operated near the hardware to get a status update of the hardware devices [61], [81]–[84].

B. Architecture of TinyOS

In 2000, at the University of California Berkeley, TinyOS was first conceived and developed. At first, TinyOS was limited to the research domain but in late 2000 system architecture for IoTs was proposed [85]. Services are implemented separately and each service has an interface for another service.

The monolithic approach results in TinyOS being an open source OS designed for IoTs and it is the most widely used OS in the IoTs. The OS is flexible, portable, and component-based, which makes it appropriate for supporting multiple applications. TinyOS requires a low memory footprint of 400 bytes to support the current application. TinyOS is in its third generation, which has involved several iterations of hardware, programming tools, and radio stacks. Various companies use TinyOS with their products [86]. An application which is independent of component implementation is used to connect the components using wiring specifications. Components include an individual computational unit that is depicted into multiple interfaces. Components are categorized under following abstractions; commands, events, and tasks. To perform an operation such as initiating sensor reading, the command abstraction initializes a request message by a component. A command is used in this fashion to execute the services. In contrast, the output to an event signals completion of services and displays. Tasks provide a mechanism for communication of components. All hardware resources in TinyOS are abstracted as a component, therefore, providing multiple components to the application developer which include an abstraction for sensors, networking for single hop, ad-hoc routing, power management, timers, and non-volatile storage. As a result, it provides advantages to the developer who wants to develop an application by writing components and connecting them with TinyOS, as the components can aid in providing the implementation for the required operation or service [63].

In TinyOS, components use interfaces. The basic principle of interfaces is how components interact directly with each other. Interface is specified by an interface type and generally demonstrates some services. An interface is bidirectional and consists of both command and events, therefore, commands are implemented by interface provider while the events are implemented by the interface's user. There is not any severance between kernel space and user space because TinyOS provides single shared stack.

C. Architecture of RIOT

The IoTs has billions of interconnecting devices, for instance, household appliances, low-end devices, and automobiles using WSNs or power line communication. These heterogeneous devices have limited constraints in term of memory, power consumption, communication to meet the requirements of real-time systems, reliability, efficiency, and communication stack, and these are the indispensable parts of the IoTs. These services are similar to those provided with traditional commodities. Cloud computing provides virtual desktop interface (VDI) [87] for convenient computing which incorporates devices and tools which are used to monitor, store and analyze. In this manner, a broader vision is required that goes beyond embedding intelligence into our environment by using smart phones and portables and evolves into the connection of our existing everyday objects [88].

RIOT has micro-kernel architecture and through this RIOT is trying to use as little memory as it can. Hence, the size of the kernel is minimized. The size is just 1.5KB random access memory (RAM) and 5KB read-only memory (ROM). Its architecture is adapted from fire kernel as explained below [89].

- Fire kernel: Fire kernel provides primary features such as scheduling, process communication, and synchronization. RIOT adds support for C++ which enables efficient libraries, for instance, Wise-lib algorithm framework which includes an algorithm for clustering, time sync, localization, and security and has a transmission control protocol (TCP) network stack. It provides multi-threading with standard application program interfaces (API). Minimal footprint helps in easy debugging and less kernel failure possibilities. Failure in device drivers or file system will not harm the whole system. The approach of a micro-kernel certainly provides an actual stable system, since a device failure will not crash the whole system. The kernel increases the stability [90]–[92].

It implements multi-threading and aims to be effective in relation to energy, memory, modulator, and in term of APIs. These are all key features required for a tiny device. Therefore, RIOT is developer friendly, highly reliable, and robust against bugs. POSIX compliance is already partly available and full POSIX compliance is planned for the near future.

D. Architecture of Nano-RK

As it has been established, IoT is a network of distributed yet contagious devices. Many OSs have been suggested and have been practically implemented. In this attempt Nano-RK is presented which is a fixed, energy-aware, preemptive, reservation-based RTOS for IoTs. It was proposed by Carnegie Mellon University. Nano-RK supports multitasking, multi-hop networking, schedulability and priority-based scheduling, such that higher priority tasks always preempt lower priority tasks [23]. Nano-RK supports time-sensitive applications of IoTs in an elegant way. Therefore, it provides an extended lifetime for WSNs with limited resource usage and a small footprint. Nano-RK currently runs on fire-fly sensor networking platform [67] and micaZ platform [68]. Nano-RK uses 2 KB of RAM and 18 KB of ROM. It also supports networking bandwidth reservation, hard and soft real-time applications through various scheduling algorithms, for instance rate-monotonic scheduling and rate-harmonized scheduling. An efficient task scheduling technique has also been integrated into this OS and the resources saved by the Nano-RK provide fixed priority-based preemptive multitasking and ensure timely and guaranteed delivery of network packets [69]. Nano-RK follows the monolithic kernel architecture that is similar in approach to TinyOS [30] and it has a real-time nature for the reason. Nano-RK uses a static design time framework, for instance, it offers task priorities, a deadline period, and the reservations can be assigned offline. It may be efficient for admission control procedures as it supports fixed-priority based preemptive multitasking that guarantees that task deadlines are met [23].

E. Architecture of LiteOS

LiteOS is another UNIX-like OS that provides an abstraction for IoTs. LiteOS follows the modular approach in its architecture. It allows the application of user-friendly operations and provides an efficient way for minimizing learning complexities for the programming or operating features, such as the file directory command to the IoTs sensor network [70]. LiteOS provides a familiar interface for the user compared to other conventional embedded systems of IoTs, such as vxworks or ecos [93]. It has features that are not present in the other existing IoT's sensor networks, such as shell and a hierarchical file system. Moreover, LiteOS has a much smaller code footprint for running on other platforms, for example, micaZ requires 25MHz, CPU 20225KB of program flash, and 22 Bytes of RAM. Conventional IoTs OS requires far more computation power and more RAM, with the result that these cant not be ported to platforms like Telos [71]. LiteOS is partitioned into three subsystems: liteshell, liteFS, and the kernel.

1) LiteShell: Liteshell is derived from UNIX by adopting UNIX-like commands [23], [37]. Support for process handling, testing and debugging, file management and devices. It provides efficient ways to interact with the complicated system. This results in powerful scripting and automation system. It handles more complex commands because the base station has abundant resources however, Liteshell can only be used when there is a user present on a base station. On the user command, some local processing will be done and after, processed commands will be transmitted to the sensor node.

2) LiteFS: The second architectural subsystem of LiteOS is the file system, named LiteFS, which has done phenomenal work for micaZ sensor nodes such as Matchbox [94], ELF [95], and Capsule where additional flash is plugged. For the sake of implementation MatchBox and executable linkable first (ELF) were not adapted because they do not meet the goal of the deign which is to support hierarchy file system and abstraction for reading and writing. All IoTs devices are mounted as a file and as a directory, then lists down all hop through liteFS. For the user, files can be displayed as they are displayed in UNIX and allows the user to make legitimate commands on those directories.

3) Kernel: The third subsystem is the kernel which features dynamic loading. Dynamic loading is taken for granted in many OS. To implement dynamic loading multiple approaches have been followed. One approach is to implement on virtual memory and it has been implemented. However, the lack of hardware support and limited consumption power, this approach slows down the program execution. LiteOS follows the various efforts and not all of them involve virtual memory such as TinyOS, SOS, and Contiki. The LiteOS provides concurrency through multi-threading and supports for dynamic loading involves round robin scheduling. This allows the programmer to register event handlers in way of callback functions through which synchronization support can be achieved [94], [95].

F. Architecture of MantisOS

IoTs basically consist of resource constrained sensor nodes and these nodes monitor the environment, collect data, and send information back to a collection point or sink. In order to fulfill these requirements, the IoT must combine an integrated hardware platform, embedded OS, communications network, and back-end data services. The MantisOS multimodal system for networks of in-situ wireless sensors (MANTIS) provides a new multi-threaded cross-platform embedded OS for IoT [94]. Mantis is an OS which is energy-efficient, and lightweight. Furthermore, it has a footprint of 500 bytes which includes the kernel, scheduler, and network stack. MantisOS supports portability for various platforms, and its portability can be examined on a personal digital assistant (PDA) or a personal computer [37] and these applications can be ported to the sensor node. It has a support for remote management. The architecture of MantisOS resembles the UNIX-style schedulers and the services provided are subsets of the POSIX threads. Mutex and counting semaphores are also supported [96]. To provide these services in an efficient manner, the MantisOS is developed for use with resource constraint devices in sensor node [37], [38], [72].

RAM is the most limited resource for MantisOS. RAM has been divided into two logically distinct sections; One is, space allocated to global resources compile time and the other residual space of RAM, which is handled as a stack. After creation of a threaded kernel allocates the stack space out of the section and after thread exits the space is recovered. For implementation, there is not any appreciation to dynamically allocate heap space. An API is used for such kind of decisions and it is not inherited by MantisOS. This is necessary for memory management notion [74], [94]. MantisOS follows layered architecture. Services in MantisOS are implemented in layers as follow;

- Network stack, command server and user level threads.
- MantisOS API.
- Kernel/scheduler, communication layer, and device drivers.
- Hardware [72]–[74].

Thread table is used as a fundamental data structure in MantisOS. Thread table is static with fixed number of threads (default 452 at compile time), and fixed level of memory overhead. Furthermore, it contains a current stack pointer, stack boundary information base pointer, and size. Pointers themselves are only two bytes. After thread suspension, its current context and register values are saved on the stack. Semaphores in MantisOS are 5-byte and declared when needed by any application. They have head and tail pointers a lock or count byte. At any instance, each thread belongs to exactly one list. The timer interrupt is received from hardware for a scheduler to perform context switching. The kernel handles these timer interrupts. For an interrupt, a device driver posts a semaphore in order to activate a waiting thread. The kernel does not support any *soft* interrupt [84].

G. Architecture of SOS

For many years, technology has provided solutions for the human race. Technological developments have greatly assisted in the development of civilizations. Previously, data was stored by recording it, by typing, or in the form of digital images. The most important source of data is the human race. However, over time and due to limitations in efficiency, this data cannot be regarded as reliable and accurate. In contrast, the computer can replace the human in gathering information and it can meet the challenges of efficiency and accuracy. Hence, computers need to be empowered to get information from their surroundings with great accuracy. The IoTs stems from the availability of a cheap and tiny energy-efficient inter-node communication network.

The OS plays an important role in the domain of the IoTs device network. There is a need for an OS that could become the standard OS for all IoTs devices. The fundamental functionality of the OS is the abstraction as it hides the details and provides a clear interface [42]. The architecture of the OS has an enormous effect on the services it can provide and the first decision should be the type of architecture or design to be used. SOS was developed in 2005 by Mobisys. In addition to the traditional techniques, the SOS kernel provides dynamically linked modules, supple priority scheduling, and a dynamic memory subsystem. These services can help to support modifications after deployment. Furthermore, the higher level API liberates the programmer from management of the underlying services and the re-implementation of a popular abstraction. The SOS kernel consists of a base operating environment which has the ability to distribute and modernize programs running on sensor nodes, therefore, SOS's modules can be added and removed during run-time [75]–[77].

To implement a task or a function, position independent modules are used. Elopement of drivers, protocols, and application component are handled at module layer. In order to change hardware or resources, management and modifications are required. SOS has one and more interacting modules for the application. SOS has modular structure after distribution by implementing modules with the well-defined point of entry and exit. For execution flow, modules may follow two methods: Either they deliver message from the scheduler or they register a function call. A handler function is used for message handling. In all module function, *init* and *final* functions are used for module insertion and for module removal, respectively. *Init* function sets the state of the module at the initial timer, function registration, and subscription. The *final* message handler unleashes all node resources which includes timer, memory, and registered function. Moreover, modules-message handler processes module-specific message including handling of timer trigger, sensor reading, and incoming data messages. These messages are asynchronous. Modules in SOS are relocatable because program state is managed by SOS kernel. These modules interact through message calls to function registered, timer, and subscription [78].

H. Architecture of RETOS

The poorly designed network architecture of IoTs can lead to serious issues. In the light of IoT's constraints, RETOS has been designed to deal with the various problems created by sensor nodes for IoT's application. RETOS was developed to aim at reconfiguration, vigorous activity, and efficiency of resources. RETOS follows the modular architecture design approach. RETOS also provides two techniques to ensure the efficiency and reliability of the system including dual model operation and application code checking. These techniques are discussed below.

1) Dual Model Operation: The dual model operation is attained by stack switching. When the application is in user area, it will use the user stack at first. However, upon the system calls and interrupts, the stack changes to kernel stack. The dual mode may lay itself open to memory overhead on resource-constrained sensor node due to the thread kernel stack. To attain memory consumption, RETOS maintains a single kernel stack of the system. Before returning to user mode, thread switching is performed and the job assigned to the kernel is done. Single kernel stack enables memory efficient implementation. However, it is not capable of preempting threads in the kernel mode [43], [97].

2) Application Code Checking: RETOS provides application code checking mechanism. This mechanism comprises of dynamic and static checks. Moreover, during compile time, static code checks are used to validate direct or immediate addressing instruction. However, dynamic checks take care of verification for accurate usage of indirect addressing. In addition, the instruction can be affected by a buffer overrun therefore, dynamic checks are required for *return* instructions. To limit the access to the memory of user application, this mechanism prevents the application's memory access by inspecting the destination field of the machine instruction [97].

3) Source Field Instruction: Source field instruction may also come handy and can be examined to prevent accessing memory, hardware manipulation, kernel reading, and other data reading [98]–[102].

Review: Various OSs use different architectures for their kernel and internal systems. Each architecture have its own benefits and some lack behind. Some popular architecture is mentioned above and what features they have are also given. Mostly used architecture is modular/hybrid architecture because it gives higher memory footprint, with less development cost and maximum flexibility while development. While other architectures such as monolithic and layered ones are also used for some OSs, they have fewer features as compared to the others.

IV. Algorithms, Scheduler Types, and Real-Time Support for IoTs

Scheduling determines how tasks are executed on any CPU in an IoTs OS. A scheduler targets high throughput, energy efficiency, fairness, and good resource utilization. The various applications use different scheduling algorithms some of these applications of IoTs are required to be real-time efficient. For real-time applications, task execution within a limited time is mandatory. An algorithm with minimum power and memory consumption is known as an efficient algorithm [103], [104]. The scheduler can be classified as optimal, preemptive, cooperative, static, and dynamic [85], [105], [106].

Soft real-time can tolerate it if deadlines have not been met within a given amount of time, however, if too many are missed, that is undesirable. For example, applications like a pacemaker, reservation system, and multimedia application are soft real-time. Firm real-time is generally soft however, it

TABLE V
SCHEDULING ALGORITHMS FOR IOTS

Contiki	TinyOS	RIOT	Nano-RK	LiteOS	MantisOS	SOS	RETOS	Scheduler Type	References	Supported Algorithm	Basic Concept	Advantages	Disadvantages
×	✓	×	×	×	×	×	✓	Cooperative	[109]	FIFO	First task gets served first by process	Easy and less Overhead	Low task throughput
×	×	×	×	✓	✓	×	✓	Preemptive	[110] [111]	Round Robin	Tasks are handled as circular queue.	Fairness	Context switching overhead
×	✓	✓	✓	✓	✓	✓	×	Preemptive	[112] [113]	Priority Driven	It has a fixed size queue with assigned priority	Simple and efficient	Starvation and lost information
×	✓	×	×	×	×	×	×	Preemptive	[114]	Co-routine	Threads has their own stack and performs multitasking	Ease of implementation	Race conditions
×	×	×	✓	×	✓	×	×	Preemptive	[115] [116] [117]	Rate Monotonic	Task with less time is picked to execute	Easy to implement	Less schedualabilty
×	×	×	×	×	×	×	×	Preemptive	[118]	Deadline Monotonic	Tasks are prioritize according to their deadlines	Simple and easy to implement	Less schedualabilty
×	✓	×	×	×	×	×	×	Cooperative	[119]–[121]	Earliest-Deadline First	Task with minimum remaining time has the highest priority	Efficiency RT guarantee	High energy consumption
×	✓	×	×	×	×	×	×	Preemptive	[114]	Job Scheduling policy	A solution for large task with multi-threading scheduling	Support for longer tasks	Collision can occur
×	×	×	×	×	×	×	×	Cooperative	[23] [122] [123]	Least Laxity First	High priority is assigned to task of minimum deadline	Max schedualabilty	Complex
×	✓	×	×	×	×	×	×	Preemptive/Cooperative	[114]	ADRS	Two task cycle queues are created with high and low priority	Efficient in use	High power consumption
×	✓	×	×	×	×	×	×	Preemptive	[114]	RTS	Handles real time tasks	Efficient real-time application handing	Complex

can have hard deadlines as well. Examples would be a base band network and radio network applications. Hard real-time can not tolerate any loss, if it occurs the system is considered failed, e.g., traffic control and a nuclear power plant.

A. Scheduling Algorithms

Scheduling is essential because the system has to decide at what time interval a specific task should will be executed. Real-time scheduling algorithms need to maximize throughput but as well, also complete the task within the given time constraint [106]. IoTs provide numerous possibilities for the development of applications, yet there will be still more in the future. Most of them will require real-time support on a massive platform filled with many devices in the smarter future. There are multiple suggested algorithms for IoTs [107], [108]. Table V is presented in order to give a detailed overview of algorithms for IoTs.

Among the suggested algorithms for IoTs, they can be categorized into the following:

1) Non-Real-Time Scheduling Algorithm.
2) Real-Time Scheduling Algorithm.

1) Non-Real-Time Scheduling: While IoTs require real-time computation, some non real time algorithms are still used by applications and OS. These algorithms include, first in first out (FIFO) [109], round robin (RR) [110], [111], priority scheduling or priority round robin (PRR) [112], [113], co-routine scheduling, adaptive double ring scheduling (ADRS), and job scheduling [114].

2) Real-Time Scheduling Algorithms: Real-time scheduling mechanism handles real-time tasks. This technique employs a preemptive technique to execute a given task in TinyOS [124]–[126]. Real-time tasks are further categorized into *periodic* and *aperiodic* tasks. Similarly, these tasks are scheduled with *periodic* and *aperiodic* scheduler, respectively. Response time for *aperiodic* tasks is comparatively better than the energy consumption, which is high [127], [128].

Commonly used algorithms include rate monotonic (RM) [115]–[117], deadline monotonic (DM) [118], earliest-deadline-first (EDF) [119]–[121], and lease laxity first (LLF) [23], [122], [123].

B. Scheduling Mechanism and Algorithms for Contiki

Contiki is primarily an event driven OS and it also implements threads, therefore, it has a hybrid programming model. It supports context switching and integrates preemptive multi-threading. As Contiki is event-driven, processes run to completion, they share a common stack and have additional support for multi-threading [129].

Contiki supports preemptive multi-threading, and multi-threading is implemented as a library on top of the event-driven kernel. Library is linked with application that needs multi-threading and is divided into two parts:

- Platform independent (Interface to event kernel)
- Platform dependent (Implements stack switching and preemption)

Proto-threads are used to implement the multi-threading. Proto-threads have multiple advantages in this use, such as their being lightweight, their taking up to two bytes, and being stack-less because they need no extra stack for a thread, and finally they are highly portable because they are written in C language. Contiki supports multi-threading though its mt library. The *mt_threads* have their own private stack program counter. This can be categorized as either architecture independent or architecture dependent. The threads start to be executed when a function *mt_start* is called, however, they can be stopped either by preemption initiated by the system, or the threads can yield themselves. After the threads have yielded, the control will return to the main system thread. Function *mt_yield* is used by threads to yield themselves without argument. *mt_exe* is called and later *mt_start* will restart the execution of the thread. Contiki does not apply any scheduling algorithm because it is mainly event-driven and events are triggered by the application as they arrive and interrupts are handled according to their priority [28], [129].

Contiki is primarily event driven so if any process which will not yield will be caused to lock up the whole system. The preemptive scheduler is on the same level as proto-threads; therefore, proto-threads can lock up the whole system. Contiki does support real-time capabilities to some extent as when an interrupt occurs, its process will wake up a schedule. However, another interrupt may delay a real-time interrupt. These features in Contiki are not that helpful as it seems for real time and Contiki has no real-time scheduling algorithm though it can be achieved but with no quality of service (QoS) guarantees [29].

C. Scheduling Mechanism and Algorithms for TinyOS

TinyOS is written in NesC which provides two abstractions: asynchronous events and asynchronous tasks. The TinyOS execution module runs to completion and interrupts the handler signaled by the hardware. Many scheduling algorithms have been added later as only FIFO was implemented in the early edition of TinyOS because it was a cooperative scheduler. FIFO is bound by the size of the data structure. Since power is an important resource in IoT devices, the primary advantage of this scheduler is that it takes care of power consumption by hibernating the system after the queue is emptied [130]. To detect the race condition, two methods have been mentioned: either an *asynchronous* code that is reachable for at least one interrupt handler, or a *synchronous* code that is only reachable from the tasks. The goal is to develop a concurrent data structure to share data. TinyOS has a two-level scheduling mechanism: Events and Tasks. The single task queue is handled by FIFO and has no interruption because it follows a FIFO algorithm approach. A hardware interrupt is handled by the event handler and it can cause preemption. TinyOS has an exquisite scheduler which has made TinyOS more popular than any other OS and it is known as the *defacto* OS for IoTs. As compared to the previous version of TinyOS, the new version has added some features including priority scheduling, which means a higher priority task can get a first share in the processor to meet its deadline first.

TinyOS implements cooperative techniques such as EDF, ADRS, and co-routine algorithms [131]. Preemption is introduced into the system to ensure that tasks with higher priority have been completed. Therefore, when a high priority task comes in, the system tries to complete it first, and after it is complete, the system can go back to its previous task. Preemption involves context switching and context saving. However, context switching creates complications, therefore, the two principles for context switching are that it should be done to meet deadlines only, and a period of grace time is allowed so that it does not switch immediately. The switch should be done at the latest possible time, so that by then, if the other task is complete, no switching will need to happen. TinyOS has algorithms like real time scheduling (RTS) but it does not fulfill the RTOS criteria. However, TinyOS is still considered one of the best OS for the IoTs platform owing to its efficient scheduling techniques [132].

D. Scheduling Mechanism and Algorithms for RIOT

Across a wide range of devices, RIOT provides a uniform programming platform and it allows multi-threading. RIOT integrates a preemptive, priority based scheduler, and its scheduler aims for energy efficiency for the IoTs devices. The RIOT scheduler provides support for multiple task priorities and support for user interaction. It involves inter-process communication and timer operation and performance is measured $O(1)$ The constant run-time of the scheduler fixes the size of the circular linked list. The scheduler works without any periodic events and it has a preemptive tickle scheduler. Being timer that triggers periodically in order to compete with concurrent execution by switching threads continuously, whereas other OSs check if there is any process, or ask whether it needs to be executed.

When there are no tasks for execution, the system goes into an idle state, and maximum time spent in sleep mode is introduced for energy efficiency. Only an interrupt will wake the system up and interrupts are triggered in two situations: either a kernel interrupt or a thread switch interrupt, which also involve less context switching. Therefore, the RIOT kernel provides a minimized scheduler, which can be called out of an interrupt service routine. It also guarantees that the highest priority thread, other than that which is blocked or not sleeping, will never be interrupted with the minimum cycles used. It is not necessary to save the old threads context. Therefore, a task switch can be performed in very few clock cycles. Every thread is given a priority on its creation [35].

RIOT scheduler architecture mainly depends on the real-time environment. It also provides real-time multi-threading [36].

E. Scheduling Mechanism and Algorithms for Nano-RK

Nano-RK OS introduces preemptive priority-based scheduling and has amazing ability to support not only *periodic* or *aperiodic* task, but also *speriodic* task. In Nano-RK, the tasks with the highest priority are handled by OS. RM scheduling algorithm is used for real-time scheduling and priority is assigned according to their deadline. Nano-RK configures tasks offline because RM is a static algorithm. These *periodic* tasks suspend themselves using *wait_untill_next_peroid*() statement. In Nano-RK, timer is in enabled one shot mode. However, next timer interrupts are triggered when a task is either scheduled to wake by event or ready for scheduling. Nano-RK proposed rate harmonized scheduling for energy saving [26] and this helps to eliminate idle CPU time. Moreover, conventional semaphores are supported [38].

Highest locker priority protocol is used to bind the blocking time encountered by higher priority process due to priority inversion. Each *mutex* is linked with a priority ceiling algorithm, which means the shared resources required by the higher priority task instead of less priority. When the *mutex* is locked, its priority of a task is evaluated to priority ceiling of *mutex* and once *mutex* is relapsed, task goes back to its original priority. This helps for offline scheduler real time support [38].

F. Scheduling Mechanism and Algorithms for LiteOS

LiteOS is thread-based OS and it provides multiple applications. For its scheduling, it implements priority-based scheduling and RR. It has a task queue known as ready queue, in which tasks execute according to their priority. In LiteOS, a task runs to their completion. When an unavailable resource is requested by any task, it will generate an interrupt and goes to its sleeping mode. Later on, when a resource becomes available, the task resumes itself. Once a task completes its execution, it will leave the kernel. When there is no task, the running system goes to its sleeping mode, but with an interrupt. Therefore the system can wake up again.

LiteOS is inappropriate for real-time scheduling because a higher priority task can miss its deadline which is a basic need of RTOS to complete its deadline [39], [94].

G. Scheduling Mechanism and Algorithms for MantisOS

MantisOS is multi-threading OS and attempts to provide support for intricate tasks with less consumption of resources. MantisOS involves time-sliced multi-threading. Its length is 10 ms and size is 16 bytes by default. Therefore, MantisOS introduces atomic preemption. It has a benefit that application segment can not block other tasks [114]. It avoids buffer access by time sliced multi-threading because the consumer can execute and left earlier. In contrast run-to-completion can cause overflow when a consumer is obligated to wait for the procedure to execute first. It implies priority-based scheduling with RR, meaning that the highest priority will finish an execution first and low priority task can starve. RR has various priorities with each priority class. It has five priority states as ready, queue, sleep, high, and idle. Tasks are executed according to their priorities. Deep sleep state can be achieved by a *sleep*(). Furthermore, it maintains a separate queue for this function. Task awaken at first will be executed first. When there is no thread, system goes to sleep or idle mode. If a task is suspended on I/O, it goes to moderate idle or sleep mode.

Threads current context and associated register value is reserved on its stack when it is suspended. To improve performance, during thread manipulation the kernel keeps a ready list and a trail pointer. The thread can be the part of either ready thread list or semaphore list. Context switches are triggered by the scheduler when it receives any interrupt. Context switches can also be triggered by system or semaphore. The timer interrupt is handled by kernel. Other interrupts may associate themselves with device drivers. When any interrupt occurs, device driver sets semaphore as in charge for activating a waiting thread and this thread handles whatever was the reason which initiated the interrupt [41]. MantisOS provides better scheduling as compared to Contiki and TinyOS, but still not better enough for IoTs applications [40].

H. Scheduling Mechanism and Algorithms for SOS

SOS has a dynamic application modules and a common kernel. SOS uses cooperative scheduling technique for scheduling of task execution. A concept of the jump table is introduced which has a purpose of module-kernel communication. SOS deploys dynamic kernel and module to take the edge off of spaghetti code and for boost re-usability. Priority scheduling is another feature added by SOS developer for the refinement of potential. The priority queue of SOS provides responsive services for the interrupt rather to entertain interrupt context. An application is composed of more than one interacting module, having two entry points for the flow of execution. The scheduler is used for message delivery and module for external use. Tasks are asynchronous comparable to TinyOS. Scheduling takes the responsibility to deliver a message from priority queue to the message handler of the destination module. SOS has a high priority queue for some critical message. Priority queue writes a high priority message and drops out. Therefore, this approach can reduce potential concurrency errors [42].

SOS does not support real-time application. Therefore, it fails to attain deadlines [114].

I. Scheduling Mechanism and Algorithms for RETOS

RETOS is a multi-threaded OS. It provides high concurrency with preemption and blocking input/output (I/O) underlying system. RETOS entails per thread stack and implies context switching between them via its scheduling algorithms. Basic policies used by RETOS are *SCHED_RR*, *SCHED_FIFO*, and *SCHED_OTHER* policies are used to increase response time. RETOS implements boosting thread scheduler by introducing event aware thread scheduling. This approach boosts the priority of threads. Therefore, when an event occurs, the priority boosted thread will preempt another continuing task [43].

RETOS implements POSIX 1003.1b and it enables developers to assign priority explicitly and kernel dynamic priority management. RETOS supports real-time application and lies under RTOS category [45].

Review: IoT's applications are highly required to have real-time capabilities. Therefore, this is a key feature for any OS. OSs need to require preemptive or cooperative schedulers. For this purpose, OSs implemented scheduling algorithms. An algorithm can affect the performances of an OS. It has a direct impact on power usage as well. We have discussed many scheduling algorithms in detail above, including FIFO, RR, priority driven, and more. Some algorithms are well suited for real-time environments, whereas others are better suited for non-real time environments. However, the widely used scheduling algorithm is priority driven because it is the most efficient and simple, although a priority-driven algorithm also has its own drawbacks. The aforementioned discussion shows the value of an appropriate scheduling algorithm in the OS for the IoTs. The IoT's devices communicate with the Internet in real time. Therefore, real-time OSs are more encouraged.

TABLE VI
PROGRAMMING MODEL FOR OSs OF IoTs

S.No	References	OSs	Concurrency Support	Programming Model	Programming Language
1	[134]	Contiki	Yes	Hybrid	C
2	[135]–[140]	TinyOS	Yes	Event based component model	NesC
3	[141]	RIOT	Yes	Hybrid	LiteC++
4	[142]–[144]	Nano-RK	Yes	Multi-Threading	C
5	[145] [146]	LiteOS	No	Event driven	C
6	[129]	MantisOS	Yes	Multi-Threading	C
7	[42]	SOS	No	Multi-Threading	C
8	[40]	RETOS	Yes	Event driven	C

V. PROGRAMMING MODELS, CONCURRENCY, AND LANGUAGES SUPPORT FOR IoTs

IoTs is an emerging research field as it is attracting a lot of researcher not only enormous application potentials, but also its innovative microelectronic circuits. The IoTs devices have fine computational power and capabilities. However, they are reliable in the conventional system. According to [133], factors like energy efficiency, scalability, failure resilience, and collaboration are required to be considered for developing programming models of IoTs devices.

Energy efficiency and failure resilience are essential for all IoTs devices for environmental monitoring and they need to be run over for long amount of time. Their functionality should remain consistent for the maximum time period and should oppose defective communication, surprising failure, and dead devices. Scalability is an important feature of IoTs. Network consists upon numerous devices with bandwidth constraints. Developers need to develop a scalable mechanism that would achieve efficiency as much as possible. Collaboration is categorized into data collection and collaborative information processing where data is sent to a central server for instance habitat and environmental. In contrast, collaborative information processing is used to the application in which tracking is involved by processing data from multiple sensors. Programming model can be classified into low-level or high-level programming model [133].

Low-level programming model provides an abstraction of hardware and enhancing flexibility. Furthermore, it is more reliable where dynamic reprogramming is needed. Whereas, high-level programming model takes an application-centeric view rather than platform-centeric view resulting in flexible in systems performance. Group level abstraction and network level abstraction is decedent from high-level programming model. In group level, a group of motes are handled as a single entity. However, network level considers a whole network as a single network. Programming model drives the performance of the OS for IoTs and it defines how a developer can model the OS. The typical IoTs devices OS domain has multiple programming models support [25]. Mostly used programming models are event driven and multi-threading. Other programming models and their detailed features have been discussed in Table VI.

A. Programming Model and Language Support for Contiki

Programming Contiki for IoTs has various constraints because of its scattered nature of the nodes which can be challenging for programmers. Various programming models have been adapted for this purpose. Before considering any programming model three attributes should be noted like nature of the sensing node, network type and what tasks the nodes are going to perform.

Contiki supports lightweight preemptive multi-threading, which is based on proto-threads. Proto-threads is a memory efficient programming abstraction that shares futures of both event-based approach and multi-threading approach. It has implemented as a library, on top of the event based kernel. The library is occasionally linked with application that requires explicitly multi-threading. Kernel invokes a proto-thread in response to any internal or external event. This library is divided into two parts: a platform independent part and platform specific part. The platform independent part interfaces to the event kernel and platform specific part kernel stack. In practice, every code needs to be rewritten when porting the specific part of the library. Proto-threads are designed for memory constraints for the reason of their being stackless and light weight. Contiki is implementing the stack switching and preemption primitives. Usually, preemption is implemented through a timer interrupt which saves the thread state on the stack and switches. Proto-threads have following features:

- Small memory overhead.
- No extra stack for a thread.
- Highly portable.

Since events run to completion and Contiki does not allow interrupt handlers to post new events, process synchronization is not provided in Contiki. Whereas, in Contiki, processes each thread requires a separate stack. The library provides the necessary stack management functions and threads run until they are explicitly deferred or preempted [134].

B. Programming Model and Language Support for TinyOS

Choosing a programming model for any OS is a challenging job because there are constraints which need to be considered by developers while developing such application. TinyOS programming model inherits some of the component-based programming concepts. TinyOS is programmed using NesC. NesC is a programming language and it supports concurrency and communication. NesC provides assistance for simplicity, reduces code complexity, and eradicates occurrence of bugs [135].

In TinyOS, concurrency model, operations, and architecture have an impact on NesC design. Multiple components are used which are further categorized into abstraction, events, and tasks. Components communicate with each other using task. Whenever a command is originated, it is narrated into a request message. Components are also provided with the interface. Some interfaces are used by components and some are provided by the component. NesC in TinyOS primarily pays attention to two features: concurrency and execution of the task. TinyOS follows event driven concurrency model. Therefore, some complexities are associated with its programming model [136].

TinyOS programming language, NesC, is sensitive to interrupts. The various tasks run to completion, though the foremost problem is the occurrence of an interrupt. TinyOS has a wide domain of hardware platform in IoTs and these platforms have their own methods to handle interrupts. For better execution, multiple applications which are consuming resources such as OS and communication protocols can be virtually partitioned. Under limited resources like power and memory, the event-driven model provides maximum efficiency. While TinyOS did not integrate any multi threading support, but novel TinyOS provides multithread support called "TinyOS threads". The event driven model provides benefits like high concurrency. However, threads can offer an intuitive programming model and efficiency of the event-driven kernel [137]–[140].

C. Programming Model and Language Support for RIOT

RIOT is simple in its programming environment because C language is used to write its architecture. In addition to C langauge, C++ is also used which can be helpful for utilization of GNU's not unix (GNU) compiler collection. RIOT is an open source OS and licensed under GNU lesser general public license (LGPL)v2.1.

The software development kit (SDKs), for the development of applications in RIOT OS, includes the famous GNU compiler collection (GCC), GNU debugger (GDB), and Valgrind. Furthermore, the SDK framework supports application programming in C and C++.

RIOT is multi-threaded OS and has features such as zero latency interrupt handler and minimum context switching. Highest priority thread will never be interrupted more than a fixed cycle. The kernel of RIOT is kept as modest as possible and comprises besides the scheduler and threading system only in *mutex* and inter-process communication. The micro-kernel approach, functionality of the system such as drivers, and the file systems run in threads [141].

D. Programming Model and Language Support for Nano-RK

Performing multiple and different task operate at different priority or frequency. Therefore, in the network, there would be some sender nodes and some receiver nodes. Sender node can perform or use to support multiple applications, such as optimizing temperature, light or sounds. For various applications, data gathered by sensor nodes will be sent to the receiver node. Nano-RK uses C programming language to write any task. Nano-RK implements static approach while creating a task. On the other hand, it provides API for dynamic configuration [89], [142], [143]. Nano-RK introduces Task control block (TCB) as the method of controlling the collaborative and synchronized execution of tasks. In Nano-RK, TCB are crowded during initialization and system call. While OS provides APIs to modify the fields dynamically however, the static configuration is highly encouraged by Nano-RK because memory footprint can have various effects if not handled carefully. TCB comprises register and the stack of tasks, priority, reference, reservation size and ports. Tasks are suspended by the OS when they have pending events for the sake of energy efficiency. Furthermore, Nano-RK provides reservation support by *tick_consumed* field, which holds responsibility for maintaining CPU usage as per if a task is consuming more and exceeds the reservation size then it will be handled as depending on its policy of reservation. Nano-RK keeps complete knowledge of stack cycles and avoids such task which can cause for long system reservation [144].

E. Programming Model and Language Support for LiteOS

LiteOS uses threads for execution because of it's multitasking os and supports multi-threading. To aid semantic errors, each thread allocates its own memory space and also provides support for event handling. LiteOS also completely eliminates the use of events. There are two types of functions: *atomic_start*() and *atomic_end*() function to handle race condition. Shared data of threads is accessed by these functions. Implementation of these functions is not well discussed in the LiteOS documentation [145]. Separate ways are used to handle internally and externally generated events. Internally generated events are implicitly handled by using threads where these events are no longer visible however, externally generated events require special attention.

There are two solutions provided by LiteOS for handling these events. The first solution is used to create a *createThread* system calls to block until the message arrives. This approach is not sufficient for handling eternally generated threads. In addition, it introduces overhead for creating and terminating threads. While it does not cause any problem for less than 0.1 ms, it wastes a few hundred CPU cycle. However, overhead should be diminished for heavy computation. LiteOS involves callback function to provide another primitive. The callback function registers to an event. Which ranges from radio events to target detection events and is invoked when such an event occurs. The event *registerradioEvent* informs that a message has been received. For this purpose, the interface requires the user thread to provide buffer space for receiving incoming messages then kernel copies the message to the buffer and calls back function is finally invoked [146].

F. Programming Model and Language Support for MantisOS

MantisOS uses language C for its programming paradigm and multi-threading approach. MantisOS also supports preemptive multi-threading. In MantisOS, program can be written without thread parting. Hence, it gives programmer an edge to write multiple threads. This enables the same application code to be re-written multiple times on multiple IoTs devices consequently attaining high portability and re-usability. Neither there is a need of extraordinary knowledge to write code nor any program will block or halt another process. MantisOS aims for remote management in IoTs network, meaning dynamic reprogramming and remote debugging can be supported. MantisOS can handle context overhead efficiently. However, the stack size of MantinOS is only 128 byte and has a RAM 4KB and this issue can lead to stack overflow [129].

G. Programming Model and Language Support for SOS

SOS programming model is written in C and it takes an edge by using many compilers, development environments, debuggers, and other tools designed for C. C programming language is suitable for such systems with limited resource or 8-bit micro controllers. Surge code shows generic state stored in and passed from the SOS kernel into the module's internal representation. SOS kernel always stores a module's state to allow them easy modification at run-time. Modules ask for a permit from the system and they subscribe to a function. By routing technique, layer modification do not break the application.

The SOS uses the *final* message to release the allocated resources. Furthermore, the garbage collector method is used which is efficient for the resources if they do not get released explicitly [42].

H. Programming Model and Language Support for RETOS

Multi-threading is the ability of a program or of an OS to manage its resource utilization. It helps to manage more than one user at a time or multiple requests of a same user. It is done without having multiple copies of the program running on the system. Multi-threading approach is useful in RETOS as it requires a per-thread stack and context switching. However, this is not the appropriate solution. Therefore, these multi-threading techniques are specified only for micro sensor nodes. Whereas, RETOS has some phenomenal features in term of its programming model. These features are discussed below.

1) Reducing Energy Consumption With Variable Timer: The tasks create energy overheads due to scheduling and context switching. RETOS uses its timer technique for less consumption of energy. The timer depends on the *periodic* timer interrupt. The system manages timer requests with the help of threads.

2) Minimizing Memory Usage: There are two techniques to minimize the memory usage: the first one is single kernel stack and second is stack size analysis. RETOS implements single kernel stack management for data memory efficiency. In the single kernel stack system, the kernel stack is shared among every thread. With memory management unit (MMU)-less hardware, application developers should estimate accurate thread stack size in order to reduce the memory usage. RETOS implements a stack-size thread automatically.

3) Event Aware Threads Scheduling: It is a real-time scheduling interface with which the user can assign priority and also kernel assign dynamic priority [40].

Review: It is concluded that the programming models of OSs can be event-driven and multithreaded. However, it can also be a hybrid. Whole performance depends on its programming model. OSs use programming languages, for instance, LiteOS uses LiteC++ and TinyOS uses NesC++. The most commonly used a programming language in OSs is C language.

VI. Reprogramming Techniques and Protocols for IoTs

IoTs is the name of growing technology of sensor node. However, this advancement comes at a price because IoTs devices require supreme attention of developers for reprogramming. Reprogramming is the capability to change software functionality of devices at the run-time. These devices are operated at different intensities with different environment. Moreover, these devices are inaccessible after deployment. To reprogram a device solely is hefty. Consequently, dynamic reprogramming scheme is highly required and getting much attention of developer as well as researchers [11], [132].

Reprogramming OS simplifies management of software by providing ease for adding, deleting, modification of parameter, and bug fixing. There are now numerous applications in IoTs, each of them has their own reliability criteria according to their nature. A reprogramming scheme should follow a debugging and testing cycle. Moreover, interruption should not occur. IoTs network membership can be static or dynamic. which requires reprogramming scheme to tolerate node densities.

In the deployment of large-scale observation systems in remote areas, when there is not a permanent connection with the Internet, WSNs are calling for replication and distribution techniques that increase the amount of data stored within the WSNs and reduce the probability of data loss. During the communication of different nodes, data needs to be delivered reliably, so different dissemination protocols are used. At different layers, different protocols are used for the efficient communication of devices. Dissemination is the process of delivering data to nodes in a network. This process uses the routing protocol for creating a network and different dissemination protocols for giving the data to nodes in the network. Usually, dissemination occurs by broadcasting data to the surrounding nodes. One of the famous protocol is Trickle [147]. Trickle uses a polite gossip policy, where motes periodically broadcast a code summary to local neighbors but stay quiet if they have recently heard a summary identical to theirs. Trickle is used at the network or application layer. It gives an efficient algorithm to control traffic. For software update

TABLE VII
DATA DISSEMINATION PROTOCOLS FOR OS IN IOTS : AN OVERVIEW

Ref	Name	Concept	Advantages
[147]	Trickle	Protocol for propagation and upholding code updates. Implements technique of polite gossip. In case of old summary the broadcast will be updated.	It does not overflow the network with packets however, also manages the control rate.
[148]	Density inference protocol (DIP)	Stores version numbers of each data item and maintains hash for update, meaning, hash will be altered if a node requires a change.	Increases dissemination efficiency.
[149]	DHV	It lets the node to carefully select and transmit only required bit information.	Reduces size of the message
[150]	MCP	It maintains a table and handles requests of send out code. Also, polite gossip strategy is used. Requester sends out a request to its closest source.	Reduces signal collision, saves time and energy with minimum message size.
[151]	Scalable Reliable Multicast Protocol (SRM)	Data is delivered without any particular order and receivers help each other.	Scalability is increased.
[152]	Multi-hop Over air Programming (MOAP)	Uses store and forward approach. The previous way of broadcasting and its neighbor can be responsible for any missing data.	Reduces latency and overhead by 60%
[153]	Stream	Sends what is actually needed. It implements a pre-installation technique through flash memory.	Send data with minimal system support.
[154]	DRIP	Provides a transport layer interface to multiple channels Introduce complexity.	Provides reliability
[155]	SYNAPSE	It is designed to handle error recovery phase. It provides a three-way handshake technique.	Less complexity
[156]	MELETE	MELETE aims to support multiple concurrent application meaning only interested nodes will receive code.	Less time for code transportation Aims to less overload of network
[157]	Deluge	Deluge is implemented to reduce the time for completion of a larger code. It′s a three-way handshake protocol allows further propagation of newly received pages. Also, it gets negative acknowledgment.	It provides a manageable unit of transfer data And minimizes the overhead

algorithms, it is used for managing the propagation of updates. Whenever new information is available in the network, it is given quickly to all other nodes in the network and if there are no updates, then the communication overhead is kept the minimum. Drip [154] is based on Trickle algorithm, which is the most naive of all dissemination protocols and for each variable in the data, it establishes an independent trickle. If the same information has already been received by the nodes in the neighborhood, then by avoiding redundant transmissions, Drip achieves great efficiency. Deluge [157] is a reliable data dissemination protocol. It is used for propagating large data objects by dividing the entire data image into different packets of a fixed size. The packet is the smallest unit of reliability that Deluge considers. Single message is transmitted through a packet and this message is either received or dropped according to the consistency of the message. More designed protocols have been explained in Table VII.

Reprogramming flow can be divided into two tasks. Dissemination protocol is triggered when update call is sensed. The design concentrates on three things mainly;

- Selection of nodes to conserve energy.
- Segment management.
- Reliability assurance [132].

Time critical application requires to avoid all disruption during code transportation and completion time. Rapid code updates are essential, such as in military and defense attack [158]–[161].

A. Reprogramming Techniques for Contiki

Contiki is an OS for networks of embedded systems. The core code and program code are kept separate in ROM. Program codes are loaded at run-time. They can be loaded from ROM or RAM. Contiki allows "over the air programming" for networks of sensors. Furthermore, it implies dynamic linking and uses ELF or compact ELF. Contiki divides its whole system is two parts.

- The core which consists of a kernel, device drivers, application, language libraries and symbol table.
- Loadable programs which are usually loaded at top of the core.

The core has no information about the loadable program. However, loadable program keeps the information about the core. The kernel communicates between one loaded program to another by looking up the target program in an in-kernel list of active processes [31]. This one-way dependency makes the loading and unloading of programs at run-time possible. It does not require either to patch the core or to reboot any module.

B. Reprogramming Techniques for TinyOS

Overhead can occur during the process of configuration for any device. TinyOS turns the system off before any updates or exchange of components. TinyOS supports reprogramming via a deluge protocol. In TinyOS, deluge thrusts 90bytes per second, one-ninth the maximum transmission rate of the radio supported under TinyOS [162]. TinyOS current protocols include, TinyOS oppportunistic routing protocol (TROP) [163], low-energy adaptive clustering hierarchy (LEACH) [164], [165], load-balanced routing scheme for TinyOS-based WSNs [166], the implementation of an energy balanced routing protocol with dynamic power scaling in TinyOS [167], evaluating the performance of RPL and 6LoWPAN in TinyOS [168], IPv6 heterogeneous networking infrastructure for energy efficient [169], and sensor protocol for information via negotiation (SPIN) [170].

C. Reprogramming Techniques for RIOT

Reprogramming techniques of RIOT have not been discussed. However, RIOT supports both static and dynamic memory allocation [35].

D. Reprogramming Techniques for Nano-RK

Although Nano-RK implements a static approach for its architecture and model. It provides support for dynamic programming as well. The static approach in Nano-RK, ensures that dynamic configuration does not leave any negative impact on the system. For instance, If a user intends to alter the task periods, reservation resources or priorities then each task can be altered to accommodate the user. Nano-RK will verify these dynamic changes offline to address resource constraints challenges [37]. Therefore, it achieves a lightweight OS with small memory footprint. Nano-RK is a resource reserved OS and it has been proved magical for both static and dynamic setting.

E. Reprogramming Technique for LiteOS

LiteOS introduces two approaches for dynamic loading into two scenarios:

- When source code is available.
- When source code is not available.

When source code is available, there will be no memory overhead and smaller binary images will be generated. Because for a different location, different memory setting will be assigned. In contract, when source code will not be available it will require differential patching [38].

F. Reprogramming Technique for MantisOS

Since IoTs applications serve multiple purposes and devices are deployed far away. It is not an easy job to reprogram a device or update its software when these devices are within hundreds of miles from the server.

MantisOS provides a remote shell. The shell makes access for the user to log in and examines the system for remote debugging easy. MantisOS uses two modes for dynamic reprogramming: Simple and advance programming mode.

1) Simple: This approach involves direct communication through serial port. If a node is connected to a personal computer (PC), then MantisOS shell starts to check its status. It is also used to check and modify nodes memory.

2) Advance Programming Mode: In this scenario, the node is already deployed and direct access is not required. Right now, MantisOS supports remote log in and changing of the variable. This dynamic quality is implemented as system call library as a built-in function in MantisOS kernel. Reflashing parts are hard, therefore, the current solution for reprogramming in MantisOS is virtual machines (VM). VM dwells under sensor OS and it allows binary updates to reprogram any node [41]. Therefore, developing is simple and new stack-based learning is not required.

G. Reprogramming Technique for SOS

SOS has made dynamic reprogramming possible by its modules binary image with minimum related data. Distributing protocol is used for insertion of module by snooping for radical advertisement hovering around the network. After the protocol harked to an ad, it checks for two conditions to meet. It checks either, the module is already updated or not. After verification, the installation process begins. SOS examines meta data during this process because it comprises of the unique identity of module, the memory required and version information [42].

H. Reprogramming Technique for RETOS

The sensor node hardware can not provide all the functionality at the same time. RETOS use a modular mechanism to design the kernel, known as self-reconfiguration. RETOS performs modules relocation by loading kernel modules which are required by the application. Applications may require different modules. Dynamic memory location which is not generally supported by the MMU-less hardware is essential for implementation of modules. RETOS uses a memory relocation techniques. Hardware-specific information is presented for the convenience of relocation process of a parallel hardware. Every accessible address will be relocated by the kernel. RETOS has a flexible structure, therefore, relocation is easily supported [97].

Review: From above discussion, we can conclude that IoTs devices require serious attention of developers for reprogramming. Reprogramming is the capability to change software functionality of devices at the run-time because these devices are inaccessible after deployment. Dynamic reprogramming scheme is highly required now. Recently, there are many applications in IoTs and they have their own reliability criteria according to their nature. There are various data dissemination protocols for communication in IoTs. They have different levels of granularity and scope. Each OS has its own methods for reprogramming and uses a different protocol for this purpose. These OSs use one or more protocols to communicate over the air. Table VIII is gives a comparative demonstration for all aforementioned reprogramming methods.

VII. NETWORKING STACK AND TECHNOLOGIES FOR IOTS

Connectivity is an essential requirement in IoTs to transfer the data to central locations for further processing. The essential elements of these devices are: device, local network, and the Internet. Context-aware applications can support presentation, execution, and tagging. These applications, which can adapt their behaviors to changing surroundings, are attracting more attention. To simplify the complications of developing applications, context-aware middleware, which introduces context awareness into the traditional middleware, has been adapted to provide solutions for the IoTs. As a result, context is not only supposed to offer services and applications that are better suited to the users needs, but also to reduce the quantity of data transiting over the IoTs networks. Context-aware communication and networking (CACN) have been developed to be performed at all layers of networking and communications [171]–[174].

TABLE VIII
COMPARISON OF REPROGRAMMING MECHANISM FOR OF IOTS

Operating Systems	Levels of granularity			Hop		Scope	Reprogramming Method
	Component Level	Application Level	Not Available	Single	Multi-Hop		
Contiki [30]	✓	–	–	✓	–	Whole Network	Contiki implements ELF and compact-ELF method to make alteration.
TinyOS [114]	–	✓	–	✓	–	Whole Network	In TinyOS, reprogramming techniques are implemented using Deluge and Drip.
RIOT [36]	–	–	✓	–	✓	Whole Network	Not available
Nano-RK [38]	–	–	✓	–	✓	Whole Network	Nano-RK does not provide methods for reprogramming however, it has some method for dynamic changes.
LiteOS [94]	–	–	✓	✓	–	Whole Network	LiteOS directly encodes relocated information into application binaries.
MantisOS [40]	✓	–	–	–	✓	Whole Network	A shell is used to remotely access the device for any modification.
SOS [42]	✓	–	–	–	✓	Whole Network	Reprogramming is performed through module insertion, using MOAP protocol.
RETOS [43]	✓	–	–		✓	Whole Network	RETOS relocates and links modules to perform alterations.

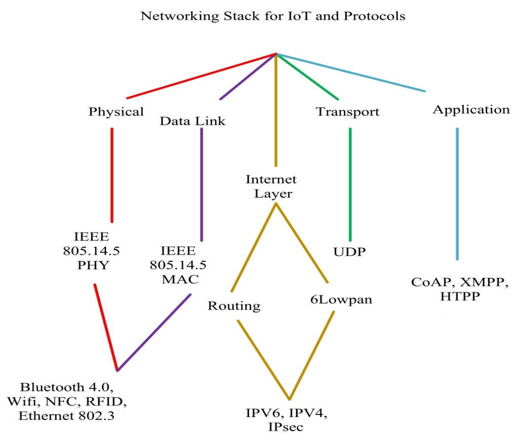

Fig. 2. Networking Stack and Protocols for IoTs.

The IoTs is now a reality, and industries and commercial markets have made a lot of contributions by creating new devices, connections to various IPs, and local networks. Therefore, before moving on to discuss how devices should work, we should know how they should connect.

There are also software-defined aspects. Software-defined networks (SDNs) provide the new network control paradigm. In SDN architecture, applications interact with controllers. Software-defined optical networking (SDON) examines optical transmissions and switching components that are flexible for SDN networks. Enhanced network structures can implement SDON control studies. Moreover, SDON-based application layer studies have been developed to achieve QoS, access control, security, and energy efficiency [175]. A SDN-based, multi-operator is a resource allocation mechanism that allocates limited backhaul link capacities to multiple smart gateways. The results show that the proposed smart gateway (Sm-GW) enables flexibility for a large number of small base stations [176]. The structure of an IoTs networking stack is presented in Figure 2 various areas such as smart homes, industries, malls, hospital, and colleges have deployed the Internet and connected to a smart world, therefore, one solution would not provide a complete fit [177]–[179]. The development of machine-to-machine (M2M) communications and the IoTs is continuing apace. The two are fundamentally linked. M2M is basically becoming acknowledged as the communications layer of the IoTs. The amount of investment in this sector is reaching astonishing proportions. The M2M/IoTs market is very fragmented, with some growth projections being scaled back. This fragmented nature of the M2M market led to the creation of oneM2M, an alliance of standards organizations looking to develop a single horizontal platform for the exchange and sharing of data among all applications. The organization is creating a distributed software layer, which will facilitate unification by providing a framework for inter-working with different technologies and between applications [180].

Wireless M2M communication with synchronized traffic patterns plays a significant role in many aircraft systems. Sensors are acute for a wide range of functions on board of aircraft. Reference [181] introduces and evaluates a new class of tree-based random access (TRA) MAC protocols for synchronous M2M traffic. It chains collision avoidance techniques, such as smoothing and access barring, with tree collision resolution to form the class of hybrid collision avoidance-tree collision resolution random access MAC protocols. Moreover, it introduced the class of hybrid collision avoidance-tree resolution protocols for the MAC of M2M traffic. Precisely, the PreBOTRA protocol combines PreBO with tree collision resolution. Protocols are based on knowledge of the number of communicating devices which are commonly available for wireless communication systems on board of aircraft. Evaluations indicate that a collision avoidance mechanism is critically important for an effective MAC protocol for bursty M2M traffic.

A. Networking Technologies

1) Wi-Fi: Everyone is aware of its importance and it stands at the top and is therefore a great solution for many applications because everyone knows its functionality and everyone has it in their homes, offices and other places they visit. However, Wi-Fi requires high power and when devices have low power and are deployed far away, they are constrained in relation to their power, and they consume considerable energy in order to work for days [182], [183].

2) IEEE 802.15.4: One of the chart-busting technologies is the IEEE 802.15.4 radio standard. It was released in early 2003 and has met the need for low power devices. It has helped to reduce power a fair amount and this will be improving each year with new devices [184]–[186].

3) IPv6 Over Low Power Personal Area Network (6Lowpan): If a task is listed as critical and has specific time needs, it introduces extra challenges. For such applications, 6LoWPN has been introduced in IoTs [187]–[189].

4) The Paramount - IPv6: These devices can be deployed away from their base station. When they cannot be accessed, IPv6 is the solution. For the IoTs to be successful, applications should not be limited to local areas only however, also should be globally accessible. The current IPv4 is limited, however, with IPv6 it will be easy to reach a global IP address and efficient communication can be achieved [190].

5) M2M Communications: In recent years, the concept of IoTs has become particularly popular through some representative applications. Under the architecture of IoTs, M2M mainly concentrates machine-type-communication (MTC) meaning, no human intervention whilst devices are communicating end-to-end. It emphasizes the practical applications of IoTs. Creating reliable networks, particularly complex mesh networks, for M2M systems could be complex and expensive. In addition, security is another important issue [191].

6) Fifth Generation (5G): Next-generation networks and standards will need to solve complex challenge of combining communications and computing together. Intelligence is at our fingertips and available to the machines that make up the IoTs.

IoTs can be an ideal application of 5G. The current IoTs technologies are standing in the way for better smart environments. Currently, there is a lot of research exploring how to develop non-orthogonal multiple access by putting a number of users into the limited bandwidth channels [192]. Furthermore, 5G has been in progress for some time, however, the 3rd generation partnership project (3GPP), the organization that administers cellular standards has officially signed off on the first specification for 5G. This is an enormous step towards what 5G networks will actually look in commercial 5G networks up and running. This will give establishments a firm requirement for what they need to be building [193].

B. Networking Stack for Contiki

In sensor networks, another fundamental need is communication. The ability to communicate is an essential aspect of the IoTs because gathering information will not be of any use if it can not be communicated for evaluation. Due to low power constraints, an individual node can easily fail. Therefore, there is also a need for a robust mesh routing algorithm. In addition to the system interface, the sensor network will support the protocol and multiple applications in visualizing, using and debugging the communication.

If the data is transported through the TPC/IP, then the whole is implemented as a service which can help to load multiple communication stacks simultaneously. It includes a μIP with support for IPv6, 6LoWPAN, routing protocol for low power (RPL), constrained application protocol (CoAP), [81], [197], [198] and the rime stack [199]. Contiki supports both a lightweight communication stack designed for low power radios and a full TCP/IP stack. In addition, communication methods use the service mechanism to call each other and synchronous events to communicate with the application programs. In Contiki, an application can use both versions of IP, i.e., IPV4 and IPv6. Contiki provides for implementation of μIP, a TCP/IP protocol stack for use with small 8-bit micro-controllers.

1) μIP: It is known as TCP/IP implementation in Contiki. μIP stack was the first standalone stack and merged into Contiki after version 0.9. μIP is used for memory constrained devices however, supports IPV4 and IPv6. TCP and user datagram protocol (UDP) protocol are also supported. μIP does not requires its peers to have a complete protocol stack. However, it can communicate with the lightweight stack. Moreover, Contiki has sample applications to use μIP stack as talent server, an email client, a Web server, and dynamic host configuration protocol (DHCP) client to name a few.

μIP implementation has the minimum set feature for the need of TCP/IP stack. μIP is implemented in C language which can support one interface. It supports TCP, UDP, Internet control message protocol (ICMP) adds IP protocols. Memory is a highly considerable resource for embedded devices in IoTs. Therefore, μIP uses memory management mechanism for efficient use of memory. It does not use explicit dynamic memory. It has a global buffer through which it stores the incoming data packet and notifies the TCP/IP stack. Appropriate application will need to copy the data in the secondary buffer or it can immediately process the data. Once done, application will overwrite the global buffer. In case of delay of data processing by application, it can be overwritten by new incoming packet. Contiki provides an implementation of RPL as ContikiRPL. ContikiRPL operates over low power wireless links and lousy power line links.

2) Rime: A stack, designed for low radio communication and supports various mode of operations like transfer and broadcast. Rime provides single hop uni-cast, single hop broadcast, and multi-hop communication support. The architecture of Rime consists of multiple layers of building blocks. Rime supports the effort and reliable transmission. It lets the applications to run their own routing protocols. It gives applications to implement protocols. The individual module is implemented mostly as simply functions, it helps to create complex stack individual modules. Contiki does not support multi-cast protocols including ICMP and mulitcast listener discovery (MLD) protocol.

C. Networking Stack for TinyOS

IoTs is all about devices that are connected to the Internet in order to connect and communicate with each other. Therefore, we need a device which can do one-to-one communication using limited energy. TinyOS 2.1.1 also provides support for 6lowpan, an IPv6 host. We need a network stack that is based directly on IP protocols for IoTs devices. Table IX presents the list of communication stacks used for TinyOS. An earlier version, TinyOS 1.0, used a multi-hop protocol, i.e., dissemination [200]. In the latest version of TinyOS, the protocols are usually used for transport, networking, and a medium access control (MAC) layer. They are designed to consume a smaller networking layer within the TinyOS network [125], [201].

TABLE IX
TINY OS NETWORKING STACK

Name	Benefit	References
Unstacked C	Interrupts overhead by its lazy preemption technique	[170]
TOSSTI	During idle time when there is no process, it helps to preserve system energy	[194]
HPL	High power listening (HPL) keeps estimating power usage and dynamically allocate power to devices.	[195]
EATT	Energy Aware target tracking (EATT) keeps a track of energy consumption each of processor, transmission module and sensing module	[196]

TABLE X
LAYERS AND THEIR KERNELS OF COMMUNICATION STACK FOR RIOT

Layers	Purpose	Kernel
MLL	MAC and data link layer (MLL) controls the network devices, manages physical connection and transmission of packet	Static Kernel
NSL	Support for logical connection, manages neighboring node, transmission of data	Static Kernel
Dynamic network layer (DNL)	Implements networking protocol and provides API	Dynamic Kernel

TinyOS utilizes active message [202] technique. This technique is predominantly used for TinyOS's message-oriented system instead of the so-called message passing. TinyOS, therefore, achieves reliability and robustness. Its communication protocols can be used in an IoTs OS [203].

D. Networking Stack for RIOT

RIOT supports network protocols for IoTs that include 6LoWPAN and RPL as well as full support for IPv6, UDP, and TCP. The employment of the RIOT network stack is completely modular, allowing for easy communication of each protocol at any layer. An adaptation layer is provided above the driver for the radio transceiver. That offers an IEEE 802.15.4 compliant interface, even if the radio transceiver itself is not proficient in this protocol [204]–[206].

The RIOT architecture also incorporates a second interface named net-dev, which communicates between the MAC and the device driver. Through this, portability can be achieved. Modules in RIOT are implemented as a single thread. However, RIOT can also be implemented as multiple threads. A message passing scheme is used for communication between these modules. Table X is presented to give an overview of structure for RIOT's networking stack and its kernel. This OS even offers an interface for communication between high-level modules, called net API. The main idea of this interface is that each layer in the network stack services have an identical interface. Therefore, it enables straight-forward extension for new features or by adding other layers. RIOT makes modular implementation possible.

E. Networking Stack for Nano-RK

Nano-RK has a lightweight protocol stack and implements port based communication. Nano-RK supports automatic package aggregation, network reservation, and buffer management. When any packet arrives, Nano-RK network stack handles it with an interrupt generated by OS. Its transmission is handled by periodic network task and they are responsible for servicing all outgoing function packets.

Nano-RK uses a zero-copy method for network buffer management. OS generates an interrupt and sets a flag to ready when the data requires to be altered. Application buffers are identified through their port numbers. Nano-RK makes a copy of received data and places it into application buffer. It notifies the application. Data will be copied until OS or application reads the previous data. Nano-RK implements time-synchronized link protocol (RT-Link) [207] for IoTs energy constraint network. It has two motives; energy efficiency and end-to-end delay guarantee.

RT link makes real-time communication effcient for Nano-RK. Furthermore, collision free transmission is achieved. It is implemented as link layer protocol. Transmission occurs into specific slot attaining energy efficiency. RT link cycle consists of 32 frames and 5ms each per frame. Types of slots are either scheduled slots (SS) or contention slots (CS). SS are allocated to those member nodes which require bounded end-to-end delay. A node makes its reservation request to a gateway using CS. Once a SS is assigned, the node becomes a member node and operates during the SS period. A mobile node always operates in the CS period because its membership changes rapidly with time, disturbing the scheduling plan set by the gateway. Since Nano-RK is a reservation-based OS. It provides APIs to application as described in Table XI. It can reserve network bandwidth according to the application requirements. Nano-RK provides reserving sender and receiver side APIs.

F. Networking Stack for LiteOS

LiteOS handles MAC and routing protocol as threads or files. Therefore, it provides a simpler way for communication and dynamically loading. During protocol usage, its application duty is to arrange packets in such a way that they can provide correspondence for communication protocols or radio interface to accurately parse it. Later on, they will be sent to their destined ports. If data arrives for a node, it will be placed in the radio file along with the port number for the intended application [208].

LiteOS implements a multi-hop geographic forwarding protocol which consists of 300 line of code (LoC) and 12 entries of the neighbor table. It can sustain with other protocols without causing any overhead because its binary images devour 1436 byte of program flash and 260 bytes of RAM.

G. Networking Stack for MantisOS

The networking stack facilitates are developing distributed IoTs applications. It is also essential to remote system maintenance after a IoTs is deployed. A networking stack for the OS should support high-level services such as dissemination and collection. It also handles low-level details including radio chip configuration, MAC, and queue management.

MantisOS provides flexibility in networking and communication. For this purpose, it divides the network stack into two parts. User space stack and MAC and PHY layer operation. The first part (user space) includes the implementation of layer 3 and above. User space is used when a user wants to add its own routing protocol. User may also use the system's provided API, resulting in context switches, memory, and computational overheads.

The communication layer of MantisOS is the known as *comm* layer and it is the second part of the network stack. It provides MAC layer functionality, API for device driver, and performs packet buffering. If a packet arrives for a thread which is not scheduled, then *comm* will buffer the packet [209].

H. Networking Stack for SOS

Each OS has its own way of a mechanism for communication. SOS implements a method known as message-passing. It is used for latency sensitive communication between kernel and its modules. This message has a header consisting of its source and destination address. The message payload is used for longer operations and through a chain of modules message will be passed one by one.

There can be some features of message passing. If there is a sensitive issue, SOS implements a high priority message to make performance better and fast response. Hardware interrupts will be sensed by the module. Furthermore, chaining of the interrupt into module and kernel is responsible for concurrency management to eliminate errors. Watchdog-rest is used for two purposes: detection of trouble making execution in modules and to their termination [78].

I. Networking Stack for RETOS

RETOS's stack has two main advantages including, API ease for developers and efficient implementation for resource constraint device.

Network stack can be static or dynamic. Dynamic part is implemented as loadable modules and static part is performance critical. Protocol for routing and transport layer can be implemented as the dynamic kernel. RETOS dynamic kernel makes network stack efficient. The networking supporting layer (NSL) is responsible for maintaining the information and to provide an interface to dynamic network layer (DNL). Moreover, DNL packet routing or transport algorithms are implemented in the form of re-configurable kernel modules.

TABLE XI
NANO-RK NETWORKING APIs (ADAPTED FROM [38])

API	Description
`port`	Set Parameter and returns port_des
`connect()`	Register port_des with network task
`listen()`	Listen for messages on port_des
`port_send()`	Non-blocking send
`port_send_status()`	Check if message was sent
`port_receive()`	Non-blocking read
`port_receive_release()`	App finished with rx buffer
`wait_until_received()`	Block until packet received
`wait_until_sent()`	Block until packet sent
`select()`	Block until one wake-up event or timeout

Review: In IoTs, networking is a major element and new techniques to overcome challenges have been proposed over the years. Some recent technologies are 6LoWPN and IPv6. IoTs stack uses multiple protocols and technologies for each layer of communication. Some OSs such as Contiki uses CoAP and IPv6 and TinyOS has an active message technique. All of OSs and techniques have been discussed above. Such aforementioned methods and techniques affect network performances which make an OS better than the others.

VIII. MEMORY ALLOCATION/DE-ALLOCATION TECHNIQUES IN IOTS

In IoTs, devices are getting information, analyzing it, processing it, or storing it constantly. We know all of them are going to cost a lot of memory. In IoTs, application may collect data of a patient, a city or of a house. It should be always efficient or else it can cost a life. Therefore, before developing any application in IoTs, a serious consideration of its management for memory is required. In the traditional OS, memory management refers to the technique that is used to allocate and de-allocate memory for different process and threads. Memory management can be done in two ways static or dynamic. Static is simple and efficient for stern memory resources. However, it has a disadvantage which can result in the inflexible system because run-time memory allocation can not occur. In contrast, dynamic memory management results into the more flexible system due to its allocation and de-allocation at the run time.

A. Memory Management Technique for Contiki

Contiki supports dynamic memory management and it supports dynamic linking of the programs. Service implementation can be changed at run-time [26], [210]. Contiki uses a managed memory allocation, which compacts the memory blocks when they are free to avoid memory fragmentation issues. Therefore, a program using the memory allocation module can not be sure if allocated memory stays in place.

Contiki provides a library which includes simple yet powerful memory management functions for blocks and fixed length. Contiki does not provide proper memory management techniques which might cause transparency while reprogramming. *MEMB*() macro is used to declare a memory block. To allocate and de-allocate memory, *memb_alloc*() and *memb_dealloc*() functions are used, respectively. In early OS structures, there was not any memory management mechanism available. By the time new application domain for IoTs, it demands the support for multi-thread execution [211].

B. Memory Management Technique for TinyOS

Better performance requires an efficient method of memory management. In the previous version, TinyOS supported static memory allocation because of limited space. Therefore, resources were allocated at design time. The devices on which TinyOS runs, have a minimum amount of memory. NesC language of TinyOS does not support dynamic memory management.

OS programming model has a major impact on its memory code. TinyOS introduces UnStacked C [170] for TinyOS. As compared to multi-thread models, this approach aids for better memory efficiency because UnStacked C has less memory footprint. Basically, what makes UnStacked C powerful is its lazy preemption. Through this approach, TinyOS gets better data concurrency. Unstacked C translator converts multi-thread code into the stack-less thread however, the code of application will not be modified. This approach results into 35 percent less RAM usage. Also, overhead is increased by 12 percentage [212], [213].

C. Memory Management Technique for RIOT

RIOT has an efficiency of $O(1)$ for its task execution. It provides run-time guarantee and has a static system. Dynamic memory management is provided for applications in RIOT. RIOT is designed in such a modular way through which maximum memory can be saved. To meet the specification, the system is configured, resulting less size of kernel and less program storage [25].

D. Memory Management Technique for Nano-RK

For efficient use of memory, Nano-RK implies static approach and introduces reservation techniques. The well-known resource kernels technique is used by Nano-RK. Applications specify timeliness and resource requirements. Nano-RK implements definite access to system resources and scheduling of task. Application timeliness requirements are satisfied. These resource kernel have been a suitable fit for the dynamic system as well as static. Nano-RK supports reservation for CPU usage and network bandwidth, for both sender and receiver. Nano-RK buffer management is presented in Figure 3.

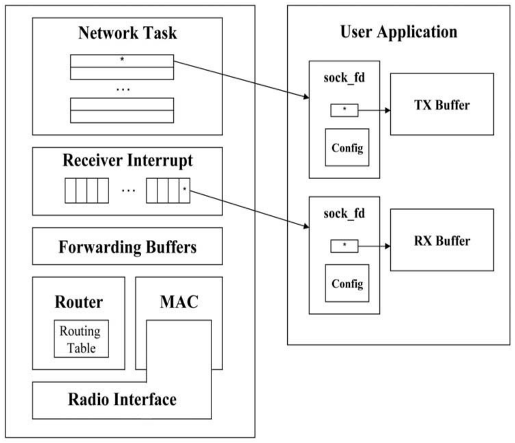

Fig. 3. Nano-RK Buffer Management (adapted from [38]).

E. Memory Management Technique for LiteOS

LiteOS supports memory management at the kernel level. Kernel compiles services which are separate from the user application. LiteOS has APIs for user including *malloc* and free for the purpose of dynamic memory allocation. User manipulates the applications through system calls. Dynamic memory expands opposite to LiteOS stack and dwells completely into unexploited memory. Therefore, it is sandwiched between the bottom of kernel variable and at the top of user application memory block. User applications can relocate memory dynamically. Application can reduce and increase size according to their need [94], [212], [214]–[216].

F. Memory Management Technique for MantisOS

OS, middleware, algorithms, and virtual machines should survive in small amount of memory. Therefore, dynamic memory allocation is necessary for the OS at run-time [217]. Memory management includes the management of primary memory (i.e., RAM), management of process between memories function, and track management for each location.

MantisOS address this issue by dividing the RAM into two portions: one for variables space run at compile time and second is managed as a heap. After thread creation, the kernel allocates the space out of the heap and recovers. All this information is kept under record by a thread table. Kernel is responsible for management of the thread table. A limited number of threads are supported because MantisOS allocates memory statistically [218], [219].

G. Memory Management Technique for SOS

Dynamic memory is not always well entertained by all OS of IoTs because of their resource restraints. SOS is using dynamic memory with book-keeping interpretations scheme, which provides an efficient solution and easy debugging. SOS uses an approach which is the best fit for fixed block memory allocation with three based block size. Small blocks are for header. Whereas, big blocks are for application. Link list provides constant time allocation and de-allocation. It helps to reduce the overhead of dynamic memory. SOS has an efficacious temporal memory reuse. The artistry of SOS is that, it can dynamically tune memory consumption. It prevents errors like module allocation of all the dynamic memory.

All allocated memory is owned by some module. SOS implements garbage collection and keeps an eye out to suspect abnormal memory usage. Dynamic memory blocks are annotated with a tiny amount of data for the purpose to detect basic sequential memory overruns. These features are on guard for any dubious memory owner. Eventually, garbage collection is enabled [220].

H. Memory Management Technique for RETOS

For rational memory management RETOS implies two techniques:

- Single kernel stack (reduces the size of thread stack requirement).
- Stack size analysis (assigns appropriate stack size to each thread).

RETOS is a multi-threaded system. It demands stack reservation for each thread and threads reservation is directly proportional to the resources required. System call, interrupt handler and context saving all require resources. Therefore, single kernel stack is the strategy used for memory efficiency. Thread stack is reduced by dividing kernel stack threads and user stack threads. Kernel stack is shared among every thread, therefore, it is implemented in a way that system will not interleave execution flow. It is the responsibility of developers to estimate the accurate size of thread stack for better memory management. A less size thread stack results in system overflow. Whereas, maximum size thread stack results in waste of memory. RETOS aims for the third solution, known as stack size analysis. This approach will produce a control flow graph of an application. The binary image is used for this operation resulting from linking application and system codes [165], [221].

Review: Memory allocation is a process by which services are assigned with physical or virtual memory space. It is the process of reserving a partial or complete portion of computer memory for the execution of programs and processes. However, in dynamic memory allocation, an executing program may request a block of main memory. The program then uses this memory for certain purposes. One of these purposes is to add a node to a data structure. It can be concluded at the end of Section VIII that IoTs devices will continue to send or receive data. Therefore, these devices should never run out of memory. Each OS uses its own methods for memory optimization. Contiki implies the method of dynamic memory management. However, it does not provide any proper technique as compared to TinyOS and Nano-Rk. TinyOS uses static memory allocation and Nano-Rk uses a well-known technique for memory management. Each OS implies its own method for memory management and we have discussed these techniques in detail in Section VIII.

IX. Energy Efficient Methods for IoTs

IoTs is a growing market and from any perspective, we may see its energy efficiency is an essential requirement. These IoTs devices can be deployed at inaccessible points. Therefore, a device should consume less power with less cost. Devices will continuously be communicating over the Internet. They will consume a fair amount of energy, that we can easily expect [222], [223].

Heavy research area is based on energy efficient protocols [160], [224]–[233] or other techniques to propose any adequate method for the future Internet.

A. Power Consumption and Management Methods of Contiki

In IoTs low power devices are able to power down the node when the network is inactive. It is highly required to reduce energy consumption. Power consumption technique depends on the application and the network as well. Contiki's kernel does not contain explicit power management abstractions. However, it lets the application programmers implement a mechanism for power management. Contiki exposes its internal queue state to the application and it allows the application to power down the system. The processor wakes up in response to an interrupt and then poll handlers are run to handle these external events. The IoTs needs power efficient protocols to address this issues several protocols have been used to reduce power consumption.

A power efficient *CoAP* has been introduced to address power management challenges on the application layer. Power consumption of the *CoAP* is analyzed with simple model that only consider application data size. It does not include full energy consumption of the kernel. Furthermore, a lightweight, open source, extensible messaging and presence protocol (μXMPP) is implemented for Contiki. It provides benefits for low power devices [234].

B. Power Consumption and Management Methods of TinyOS

Devices for IoTs require an efficient power usage as they are deployed far away from the base-station and the need to work for days under critical environment. To address this issue various techniques have been proposed. TinyOS have APIs to handle and manage the power usage. Minimizing duty cycle is a key for power efficiency. For this purpose TinyOS implements a fine grain power management technique [114].

C. Power Consumption and Management Methods of RIOT

Deep sleep mode is necessary for IoTs devices. RIOT has a tick-less scheduler which works without any periodic events. When there is no task queued, a system will go to its sleep mode and it will switch to the idle thread. It is created during OS boost and has the lowest priority. The idea behind this task or thread is to increase power efficiency.

Once a task decides to switch, it exits when hardware generates an interrupt. RIOT has a less complexity, so that it consumes less energy. Therefore, context switching and processing time should be minimized. This strategy will favor those scenario, where user interaction is minimum [235]–[238].

D. Power Consumption and Management Methods of Nano-RK

Nano-RK introduces a virtual energy reservation. Any task performed by an OS consumes energy in a form of CPU usage, interface, and turning actuators. There is a crack for this by changing the CPU and reservation size. Nano-RK provides sensor reservation. Sensors are by default in off state. If any operation asks for a sensor to change its state to *on*, its value will be read and will be turned off again through system API. Nano-RK makes sure that there would always be a limit to access over a period. It maps all the resource being used, estimates power usage, and lifetime of the device. Device lifetime and energy usage of system can be modified by verification in CPU, network buffer, and devices [38], [239]–[241].

E. Power Consumption and Management Methods of LiteOS

LiteOS is a Unix based OS and new IoTs devices are highly constrained, LiteOS is implemented on micaZ. There is a comparison between the benchmark TinyOS and LiteOS in HEX format. Results proved that LiteOS has a smaller code size than TinyOS for applications meaning, there will be a less cost of development. For consumption of program flash, LiteOS does not introduce overhead. LiteOS, however, consumes more energy because LiteOS is multi-threaded OS. We can not neglect this fact that LiteOS has a small memory footprint. Therefore, it can avoid energy consumption and reduce it to a minimum. There is no well-defined mechanism discussed for power management [242].

F. Power Consumption and Management Methods of MantisOS

Power management is required to activate the sleep mode for OS system when there is no process queued to handle next. AA batteries sustain for max 5 days as they have 3000 milliampere/hour (mA/hr) energy to support a device. To provide maximum performance using minimum energy is difficult. A better mechanism should know when it is safe for a system to sleep according to its situation and its purpose. MantisOS uses *sleep*() function to activate a hibernate mode for the device to conserve energy. A technique has been proposed for better energy efficiency in MantisOS. It introduces a term known as an "idle thread". Idle thread is a thread initiated when all other threads are blocked. The idle thread has low priority and created by kernel at the start-up. This achieves energy efficiency for MantisOS. As it may work for better CPU utilization and conserve energy. MantisOS has different running cycle for sense and forwarding which consumes 20mA [243].

G. Power Consumption and Management Methods of SOS

Every OS for being eligible for IoTs devices requires a power efficient mechanism or else they can not stand in the competition of this swift technology. SOS stands high for new technologies to optimize services for system performance however, there have not been any specific mechanism for energy efficiency [42].

Every OS has its own way for memory management. Some OS use methods to allocate or deallocate while other use techniques known as UnStacked C. However, RIOT is developed in such a smart way that it utilizes less memory. Other OS use dynamic memory management as well.

H. Power Consumption and Management Methods of RETOS

In a threaded system, the frequency of a scheduling is lower than passing messages like an event driven system. RETOS being multi-threaded system adds in little risks including energy overhead, context switching, and many scheduling operations. Context switching and overhead act as a foe for device energy. In the threaded system, there is a continuous call for interrupts handlers and this can result in high energy consumption. To address this issue, RETOS uses variable timer tick, it helps to minimize the energy consumption for RETOS. These timer requests are managed and updated for the running remaining threads accordingly [90].

The IoTs represents a new reality. The perceptions derived from data collected from new Internet-connected devices can be used to develop new services, enhance productivity and efficiency, improve real-time decision making, solve critical problems, and create new and innovative experiences. However, as more devices are connected, networks face increased fragmentation, interoperability, security, and energy efficiency challenges. Since the IoTs is expanding every day, it will eventually result in critical energy consumption issues. Wang *et al.* [244] present an energy-efficient architecture composed of three layers to reduce energy consumption. The service networks proposed in the paper may serve as a bridge between physical sense entities and virtual objects. The networks receive the description data from the sense entities. For instance, the product ID or devices IP address are saved to the cloud server to create virtual objects with the semantic description of the sense entities. This study also proposes a sleep schedule and a wakeup protocol for IoTs-based networks to provide energy efficiency.

Review: Energy efficiency presently tends to trail novelty in gauging the appeal of the IoTs. It has already been mentioned that energy efficiency is crucial in the IoTs. It is necessary for the OS to have these techniques or methods. Since smart devices need functionalities to understand energy consumption issues, IoTs applications needs more dynamic and more responsive to energy demands. Contiki has no power management mechanism. Contiki OS addresses power management issues through API like TinyOS, whereas RIOT introduces deep sleep mode and Nano-RK uses virtual energy reservation. Some OSs are built in a manner that consumes less energy, such as LiteOS.

With such massive coverage potential in our daily life, IoTs with reduced energy consumption has attracted more and more attention. In recent years, energy-efficient networking and computing have been extensively studied from many perspectives, such as the framework design, the algorithm design, and the resource reusing. IoTs is expanding its outreach to almost every aspect of our daily life. By utilizing network coding in IoTs, the IoTs energy consumption can be reduced. Since network coding basically allows the intermediate relay nodes to encode the incoming packets, the network could achieve the maximum multi-cast capacity. Adaptive network coding (ANC) scheme in the IoTs core network with the software-defined wireless network (SDWN) is a popular power efficient scheme. Simulation results have demonstrated that the ANC scheme can achieve higher transmission efficiency than existing schemes [245].

Furthermore, another method has been proposed for to minimizing total energy consumption in IoTs environment, while communication. During this communication devices in the network share the information directly. The proposed method reduces a total number of transmissions for the information sharing by using an effective network coding-based technique which dynamically selects a node and a data packet for each transmission. Simulation results show that the proposed method has better performance than an existing network coding-based method [246].

Moreover, random linear network coding (RLNC) is a popular coding approach for efficiently transferring data in complex networks of IoTs. The proposed scheme develops and evaluates parallelization strategies for speeding up RLNC on heterogeneous multi-core architectures. It has investigated efficient RLNC for heterogeneous multi-core architectures, which are likely to be widely employed in IoTs communications and applications. Building on dense linear algebra computing principles from numerical computer science, it has developed efficient block-based matrix inversion and multiplication strategies. Moreover, directed acyclic graph (DAG) scheduling algorithm avoids artificial synchronization and only considers the inherent data computing dependencies. IoTs-based node systems indicate that this approach produces excellent speed-ups for various generation and symbol sizes. High speed-ups are achieved even for small symbol sizes which have proven problematic in earlier studies. Future research direction is to examine the loose synchronization method for distributed memory architectures [247].

X. Simulation, Testing, and Debugging Tools for IoTs

To perform at real time, experiments cost can be high and also it be time consuming. Therefore, simulation is required before implementing any application for real at large scale. Simulation helps to implements various factors such as power consumption, coverage area, and a number of the node for that area. Nowadays, simulation is much limited to networking. Currently, multiple protocols are being introduced in IoTs field. Hence, an easy way to test and debug this application in the simulator to avoid high cost and chance of errors after deployment [248], [249]. It is the most common approach for the development of application and test or debugging and have numbers of advantages: a much lower cost, less effort, easy implementation, practical large-scale network testing faster, and better estimation of performance tuning.

With the exponential growth of the use of sensors in IoTs devices, simulation tools might help further developments on because it can actually help to provide a better environment before actual deployment.

A. Simulation Tools for Contiki

Contiki provides a simulator named Cooja. It allows real hardware device to be simulated. Furthermore, Cooja allows cross-level simulation meaning simulation support at many levels of the system. It is a flexible simulator for Contiki and combines low-level and high-level simulation. It is written in C and java which makes it user-friendly simulator [250]–[255].

TABLE XII
TinyOS Simulators and Their Functions

No	Simulator	Function
1	TOSSIM	Event simulator and maintains sorted queue of events.
2	Avrora	Helps to simulate heterogeneous network for TinyOS and AVR devices.
3	EmTOS	Runs TinyOS application as a single module.
4	SmartSim	Geographical simulator, similar to TOSSIM in functionality.
5	mTOSSIM	Helps to predict TinyOS devices battery lifetime.
6	PowerTOSSIM	Provides accuracy for power estimation of device.
7	Viptos	Supports geographical environmental application for TinyOS.
8	QualNet	Provides accuracy and stability for TinyOS devices.

B. Simulation Tools for TinyOS

TOSSIM [229] is the standard simulator for TinyOS. Any other simulator provides any extension for this or any other functionality. However, TOSSIM is used for most application. Other simulator tools for TinyOS have been described in Table XII.

C. Simulation Tools for RIOT

Linux and Mac OS are the systems that can implement the instance of the RIOT as a process which can be helpful for testing in virtual network emulation. It can be used as a native net to emulate a single ether-net link. However, Cooja can be used as a simulator for RIOT because it does not have its own specific simulator platform yet. For smoke and regression testing, unit tests are used. Travis is used for continuous integration test on the Web based services. Additionally, testing can also be done on a number of open test-beds supported by RIOT, e.g., IoT-LAB [256].

D. Simulation Tools for Nano-RK

For the larger real-time application in IoTs, many OSs use simulators. Nano-RK simulation tools have not been yet discussed. However, it has been seen and studies that Nano-RK can simulate application using alf and vegard's reduced instruction set computer (AVR) Studio [114], [239].

E. Simulation Tools for LiteOS

Like many other OSs, LiteOS also performs its simulation using Avrora [257] and have been proved convenient and efficient for LiteOS.

F. Simulation Tools for MantisOS

MantisOS uses Avrora as its simulation tool. Avrora [257] is built in java for specifically IoTs. It was developed by the University of California and can simulate AVR-based micro controllers. Avrora provides energy efficient simulation. Furthermore, it can support simulation of thousands of nodes, and with accuracy, execution time can be conserved. Avrora has no graphical user interface (GUI). Furthermore, it does not support for network communication tool [248], [257].

G. Simulation Tools for SOS

SOS implements its application using Avrora [257]. As Avrora has been a reliable simulator for many OS, therefore, developer use Avrora to measure various factors of network including power efficiency and CPU overhead.

H. Simulation Tools for RETOS

Rational IoTs environment requires a simulation, by using any system as a simulator either, it's large or small. RMtool [97] is used for network management in RETOS. It provides the better support for monitoring. With this tool, user can have a control and access over the networks. It can help to use the less memory, therefore, it provides robust and a modest solution.

Review: Testing and debugging is a major part of software development. Therefore, while development simulation is an important concern. Many OSs use their own simulator such as Contiki uses Cooja simulator. TinyOS uses TOSSIM along with many other. Whereas, other OS third party simulator. In IoTs, Avrora is also mostly used simulator.

A complete review of all OSs has been presented in Table XIII. It presents extensively discussed feature along with others important relevant information of each OS. New available versions of each OS and standards are presented as well. Given that, all features have their own standing. It is concluded in a top to bottom approach that Contiki OS being introduced in 2004 has come a long way. It has managed to stay at the top because of its innovate networking methods and programming model. However, it lacks behind in being real-time OS. Most recent OS is RIOT, therefore, it has much groundbreaking communication techniques and has a support for 6LoWPAN. In addition, it has support for various hardware. TinyOS comes in the third position for being the *defacto* OS in WSNs and IoTs. Moreover, Nano-Rk is popular for its bright power management techniques and has its own kind of importance after TinyOS. LiteOS has its reputation for its programming language and memory management techniques. MantisOS, SOS, and RETOS stand at the bottom yet they have some interesting feature support.

XI. Applications, Features, and Products of IoTs

IoTs is a newly emerging technology claiming the third position after the computer and the Internet. In this technology, things are connected to the Internet in one form or another, resulting in a tremendous amount of data that needs to be processed, stored and represented in an efficient way. IoTs applications vary in different fields and include personal, industrial, and national applications. In recent years, these applications have steadily increased and some of them are already deployed and are being used at different levels [258]–[260].

IoT's application require hardware, middle ware and presentation. These applications have features like communication of devices with real-world, interaction with the environment, interaction between people and devices, automatic routine tasks with less supervision, self-organized infrastructure and communication security. There are many applications being developed these days and in recent years IoTs applications have gone only on top [260]. Applications require advance RFID and addressing technologies, with better visualization and storage capacity. Keeping in mind the requirements of IoTs application architecture for less energy consumption have been developed [244]. These application are briefly explained as follows:

A. Smart Homes

There have been new inventions for products that connect and control our homes from our phones. With just one touch, an application installed on a phone will control the lights, burglar alarms, electronic devices, and drapes. These features can add ease and flexibility to our lives. Furthermore, the control of our home appliances will allow for better security and better management, making it easier to manage all these systems. This is what we can call the revolution of things. To attain this, devices and installed OS should work in a real-time environment to send notification or alert us on our phone if anything needs to be looked at [17], [261], [262]. This does not require a lot of bandwidth to access or control. Usually, WiFi is used to control devices remotely. OS should consume less bandwidth with more energy efficiency. Some popular examples are: smart air conditioning that will turn on a few minutes before the person arrives home so that they do not have to wait for the house to cool, a smart door lock controlled by the phone, a smart thermostat that will gather data about temperature from the Internet to change the internal temperature. On an individual basis, the application in the house of networks, home security, smart control of home appliances, distance learning, and the control of carbon footprint will all be addressed [263]–[265].

B. Wearable Technology

Wearable technology is explained as the technology you can wear at any time. It is best suited for the traveler and people who are always "on the go". A number of products have been designed and implemented in this area, ranging from rings to footwear, and from watches to back-packs. This technology can be seen everywhere [266].

Wearables have received a lot of attention since their launch in 2014. Generally, wearables provide one function and are implemented against various standards. It can provide security, health, and lifestyle versatility [267], [268].

The global market is growing at an extraordinary rate and, because of that, efficient hardware and middle-ware are also required in order to perform at their best. These wearables are customized and upgradable. One can access a wearable

TABLE XIII
COMPARATIVE OVERVIEW OF OSs

Features	**Contiki**	**TinyOS**	**RIOT**	**Nano-RK**	**LiteOS**	**MantisOS**	**SOS**	**RETOS**
Year of Publication	2004	2000	2013	2005	2008	2005	2005	2007
License	BSD	BSD	LGPLv2	Dual License	GPL v3	BSD	Not known	Not known
Open/Closed	Open	Open	Open	Open	Open	Open	Open	Open
Versions	Contiki 3.0	Tiny 2.0	Not available	Nano-RK 1.0	LiteOS 1.0	Mantis 1.0 beta	SOS version 2.0.1	RETOS 1.4
Hardware support	AVR,ARM,cortex-M	AVR, MSP430, px27ax	AVR, MSP430, ARM7, ARM Cortex-M,	AVR, MSP430,	MICAz, IRIS	MICA2, MICAz,	MICA2, MICAz,	MICAz,TelosB/
Architecture	Modular	Monolithic	Microkernel RTOS	Monolithic	Modular	Layerd	Modular	Modular
Scheduling	cooperative	Runs to completion	preemptive, tickless	preemptive	Runs to completion	Priority classes	Preemptive	Preemptive
Scheduling algorithm	Priority based/RR	FIFO	Priority based	Priority based	Priority based/RR	Priority based	Priority based	Priority based
Real-Time support	NO	NO	Yes	Yes	No	No	No	POSIX threads
Programming model	Multi-threading	Event-driven	Multi-threading	Multi-threading	Events/Threads	Threads	Events	Threads
Programming language	C	NesC	C, C++	C	LiteC++	C	C	C
Memory management	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
Power management	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Networking stack	uIP, uIPv6, Rime	Active message	gnrc, OpenWSN, ccn	RT-Link , TDMA	File based communication	Comm	Message	Static and dynamic kernel stack
Supports 6Lowpn	Yes	No	Yes	No	No	No	No	No
Resource sharing	Serialized access	Virtual and completion events	Not Available	Serialized access	Synchronization primitives	Semaphores	Virtualization	Virtualization sharing
Reprogramming	No	Yes	Not Available	Not Available	Yes	No	Yes	Yes
Static/Dynamic system	Dynamic	Static	Dynamic	Static	Dynamic	Dynamic	Dynamic	Dynamic
Concurrency	Yes	Yes	No	Yes	No	Yes	No	No
Simulators	Cooja	TOSSIM	Cooja, IoT-LAB	AVR Studio	Avorara	Avorara	Java SOS	RMtool
Security	Yes	Yes	Not Available	Not Available	No Available	Not Available	Not Available	Not Available
Shell	Yes	No	Yes	Not Available	Yes	Yes	Not Available	Yes
Database	Coffee	TinyDB	Not Available	Not Available	Yes	Not Available	No	No
Documentation	Yes	Yes	Not Available	Not Available	Yes	Yes	No	No
Testing	Yes	Yes	Yes	No	Yes	Yes	No	Yes
Debugging	Yes	Yes	Yes	No	Yes	Yes	No	Yes

through a phone or touchscreen. Some tracking devices only gather data from gestures and give feedback accordingly. Some of the most famous products are Google glasses and business application software integrated solution (BASIS) watches. Other activity trackers include a social media dress, global positioning system (GPS) shoes, a smart headset, wireless footwear for the artist, and wireless drums for the drummer. Wearable technology demands OS be real-time and secure. Information can be transferred between connected devices therefore, less bandwidth consumption is necessary. Moreover, battery life for wearable devices is a topmost priority [266], [269], [270].

C. Smart City

A smart city is an area of technology that serves people, and this technology is essentially built around the user. A smart city is nothing but a network designed to optimize resources. The core is the connected street, in this way a city will be more livable and more alive [271]. For example, street lights will switch on if someone walks by or stops in a parking spot. If a person is in hurry and can not find a place to park, they will be guided in real-time to the nearest convenient parking spot. Furthermore, if your phone or laptop has a dead battery, you will be helped to locate the nearest place where you can find a charger. If a person is disabled, will be able to turn on a light that will generate a signal for others to be aware while crossing a road. The waste collection can be automated as disposal companies can get a notification if they need to collect garbage from a specific place. Regarding changes in the weather, IoTs devices will notify the system about noise and air pollution. In the case of an accident, an immediate alert will be generated and sent to a nearby hospital or to another facility to get there as soon as possible. Traffic will get real-time reports on changed routes or be told to slow down to avoid a roadblock. These applications require real-time notifications with less bandwidth consumption. OS developed for smart cities should be more declined towards being real-time and secure about personal information shared over the network [271], [272].

The communication between operated devices is called machine-type communications (MTC), and the MTC Devices (MTCDs) form an essential part of a smart city. MTC over cellular networks is expected to be an essential part of wireless smart city applications. The long-term evolution (LTE)/LTE-advanced (LTE-A) technology is a foremost candidate for provisioning of MTC applications. In this purpose, random access (RA) is the first step to recruit a data transfer using an LTE network. To resolve the RA congestion in LTE systems, different solutions have been proposed. These proposals mainly focus on key performance metrics such as access delay, access success rate, QoS guarantee, energy efficiency, and the impact on HTC traffic, however, the solutions are not capable of managing massive bursty access attempts. Ali *et al.* [273] provides a review of these proposals to solve the congestion problem during RA in terms of five performance metrics. Through simulation results, it is shown that the collision resolution of the RA model provides reliable and time-efficient access performance.

LTE and LTE-A are considered viable for supporting evolving M2M communications. It is also referred to as machine type communications (MTC). As an enabler for the futuristic IoTs, the performance, and efficiency of MTC applications in LTE/LTE-A networks is a significant research area. A proposed study [274] derives equilibrium models for the connection establishment system in LTE/LTE-A networks. This equilibrium approach models preamble contention through Poisson random variables. This analysis is applicable to both network operation with a single preamble and multiple preambles. It is shown that the uniform backoff in LTE/LTE-A connection establishment results in stationary Poisson distributions for key system variables. The result that the LTE/LTE-A uniform random backoff system can be controlled to give rise to Poisson process model dynamics can serve as the basis for a wide range of future research.

The next generation of wireless access 5G targeting commercial availability. 5G benefits for low-power IoTs devices. Traffic volumes will be multiplied as devices will require connectivity. Cities can become more livable, secure and efficient with the evolution of 5G. Connectivity services are highly scaled and programmable in terms of capacity, coverage, reliability, and efficiency. In future smart cities, real-time behavior, power efficiency, and reliable speed will be achieved [192], [275], [276]

A smart city is another area of research that aims for security and for the management of various functions at the same time. It can help manage a city through an expanded network that will monitor traffic, energy, water supplies, and waste management [277]–[279].

D. Smart Grid

Since the IoTs facilities have been added to the previous grid system, they have become smarter meaning more data more and more information grid. It is a network of power lines that carry electricity to homes and industries. Industries and homes will notify the system on power usage. Since IoTs has helped to become the power supply system two ways information will be sent to the main hub about usage. It will have the information about usage and consumption. Furthermore, in the case of power failure, heated system, better management, and avoidance of blackout smart grid will perform better than traditional methods do. Moreover, customer's information will be sent to the server enabling power supply system to notify the user of who is using what and how much. Devices used for this purpose require real-time guarantee with minimal bandwidth consumption. Therefore, while designing OS, the developer should keep in mind these requirements for smart grid systems application.

In IoTs, 5G will bring new features in the smart grid. It will connect more unconnected IoTs devices to help reduce electricity and energy costs. 5G is itself, more energy efficient therefore it will have its share in energy efficiency in 5G. For instance, dimming public lighting when no pedestrians or vehicles are present. It could save power and reduce light pollution. In a world, where energy needs to be consumed smart grid really seems smart [280]–[282].

E. Smart Industries

The IoTs in industries has been heralded predominantly to increase progress for operations. For growth of opportunities, industries can also benefit greatly by using it as a tool. Companies can grow by boosting profits in form of rapid production, novel hybrid business models, smart technologies, and transform industrial workforce. It is necessary to modernize and bring operations and IoTs together for a new generation of intelligent devices. It will provide operational energy, new services, and unconventional growth [283], [284]. This large enterprising can be challenging for the developers. However, these designed can help to explore the possible and feasible solutions. For example digital industry, robots, digital grid, and energy. IoTs is changing its competitive landscape of industries in control process, monitoring of the industrial environment, product life cycling, energy saving, pollution control, medical, healthcare, and transportation. IoTs has its application in the production of all areas, for these products of IoTs, OS should be able to meet all the challenges. All products require less energy consumption and high reliability. Moreover, these products should consume minimum bandwidth in processing.

F. Smart Traffic

IoTs is changing the way we drive. People are now more connected with things. These things are available at their fingertips. IoTs has been connected the technologies to speak the same language. Gathering information, connecting them with technologies and ultimately changing customer applications. Regarding IoTs development, smart transportation plays an essential role. By 2020, 75 percent cars worldwide will be connected. However, this will introduce challenges including collecting real-time information, enhancing on-time delivery rate, minimizing fuel power, and fleet management to improve performance. The goals of IoTs are safe green and intelligent transportation. Furthermore, to detect people driving while drinking has been proposed as well using IoTs [285]. Location and tracking of devices are now possible. An emerging application is a smart taxi regarding public transportation. Temperature, oil, pressure and engine speed can be monitored through IoTs. Cloud-based device management system has been introduced. The device will be able to control and monitor their vehicle through their phones [286]. Intelligent transportation traffic guidance and intelligence control will bring the intelligent transportation. Enhanced vehicle positioning, with real-time scheduling, remote monitoring, controlling, security, and road coordination programs are the gaps need to filled by an adequate OS [287], [288].

G. Smart Health Care

Health care is a vital element for our health and well being, whether for the hospital, dentistry, or nursing home. It cures sector aims for all age group of the population at the same time. This requires for efficiency and less cost. To keep up with the demand of population, the need for technology is expanding [289]. All of the cost, quality, and safety leads us to smart health care which integrated all health solution. This can promote efficiency in many ways including service of data transmission of a patient without human errors [290]. Regarding health care, real-time applications are compulsory because any kind of delay cannot be tolerated. While developing OS for healthcare application developers should be more concerned about real-time information sharing with security and less bandwidth. In the domain of medical, it will not only bring the intelligent medicine but also hospital management efficiency, patient monitoring, and personal monitoring. Remote medical services will be available for the families. On an individual basis, the application in the house of networks, home security, home appliances, smart control, and control in the footprint of carbon will be addressed [287], [291].

H. Smart Retail

IoTs has sparked a new innovation in the retail industry. IoTs is finding ways for smart retail and sales. It can help to analyze customer behavior by improving sales, the performance of the store, and quality according to customer needs. Furthermore, it analyzes how many people visit store hourly, daily, weekly and monthly. IoTs introduces various management systems for the user such as smart shelf management system, smart retail solutions, supermarket, and hypermarket. Smart signage [292] nd video wall can also be a useful application for this area and for attractive strategies [293]. Battery consumption of OS should be minimum for retail application. Moreover, bandwidth is a major concern for this applications as well.

I. Smart Supply Chain

IoTs has a smart solution for the complex supply chain. Companies are doing business worldwide. IoTs makes it easier by connecting systems. Customer, sales department, factories, and carries use the same platform for communication. The customer can reserve an electronic order for production. Each order can have a unique tracking number, partial will be label accruing to it. The product will be to deliver on time. is another application implementing IoTs at fully to its best. Each product can have an RFID. Later on, packed and scanned through RIFD antenna also using the application on your phone. OS should provide efficiency, rapid delivery, effective precision, and universalism for all products used in this application [294], [295].

J. Smart Agricultural

Our planets climate is changing and this change announces a new challenge for global food security. Farmers, fishers, and herders are getting affected by these changes. Therefore, IoTs is required to adapt to these changes with keeping the resources such as water and soil without further depleting. IoTs can help the system for a sustainable development and food security in changing the climate. Furthermore, maintaining the production and incoming adapting [296], [297].

In agriculture, it will help in better resources utilization. The introduction of quantities techniques into agriculture production processes, cultivation of environmental, and monitoring. With smart e-commerce and smart logistic inventory control, management and traceability can enhance [298]. Moreover, it will enrich public logistic services

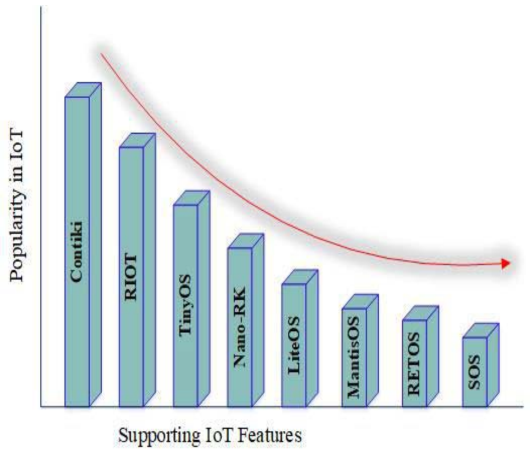

Fig. 4. OSs Support for IoTs and Their Popularity.

platform with converging of different zone and domain [299]. For this purpose, energy efficiency is a major concern as well. Developers should emphasis more on methods to reduce energy consumption.

Methods to increase energy and environmental sustainability have been studied continually and extensively. While there are thousands of sensors collecting data at a standard power plant, only 2 percent of the data is really effectively analyzed and the rest is almost wasted. A key component of energy management is the ability to use only the amount of energy needed for each of the individual devices to satisfy its intended purpose. In todays world, well-analyzed data is extremely valuable. The IoTs will create a large amount of data generated from multiple domains, such as its applications of smart homes, industries, traffic, wearables, and hospitals. It will consume energies for various purposes, such as environmental, social, and economical purposes [300].

Review: IoTs is growing its root through the various applications. The most used applications are discussed above. These applications are already in use in various cities and countries. Therefore, while developing any application of IoTs, OS developers should be aware of the challenges they may come across. Such many applications require reliable communication without data loss. Whereas, others require minimum power usage and longer lifespan. To address these issues, developers should concentrate on the major requirement. Smart home applications and wearable technologies require more battery for the devices. OS with less battery consumption is more preferable. Whereas smart cities and smart industries may have much more exchange of information, therefore, security and memory management mechanism are required for such applications. All discussed applications entail for better communication services with minimum bandwidth consumption.

Keeping in mind Table XIII, it is observed that Contiki OS and RIOT are working on advanced communication mechanism and better techniques. Table XIV is explaining applications according to their use and supporting OS. It is observed that most the suitable OSs for various applications are Contiki Os and RIOT. The rest of the OSs are shown in Figure 4. This visual representation is presented through comparison of features and popularity among IoT's application developers.

XII. Challenges and Limitation Faced by IoTs

Many companies are rushing to build application for IoTs and spending huge sums of money because it is the next big opportunity. Smart washing machines, heating systems and fridges have been introduced. IoTs has brought many changes and positive dimensions. However, these are not without risks and challenges. Some of the limitations of IoTs and its applications are described below [39], [302].

A. Security and Privacy (Authorization of Devices, Open Ports and Encryption)

IoTs applications and smart areas handle lots of data on a daily basis and this data is shared across devices. Information about homes, buildings, and cars is being shared between devices all the time. All this information demands better management system. The existing security systems are not suitable in many ways and they can cause serious issues for the user. One mistake may make the home security, cars, buildings, and saved data vulnerable. Cybercrimes are possible due to these areas of vulnerability. Criminals can break security while driving around; this is known as war driving [303]–[305].

B. Reliable Communication

The IoTs consists of fixed and mobile technologies. For two-way communication without data loss and with complete reliability, cyber-physical systems (CPS) help create new services and applications for IoTs. 5G will be able to transmit data about 10 times faster than 4G. That could revolutionize the IoTs. 5G will bring the very high performance and scale needed to incorporate billions of users within the mobile community. It will also give service providers the ability to create a new generation of personalized services while supporting M2M and the IoTs. In the near future, 5G will enable CPS for device-to-device (D2D) communication. To facilitate constricted and close interactions among an increasing number of physical things, such as smart phones, tablets, smart sensors, and other devices, D2D technology allows these in-proximity devices to communicate directly with one another [306]. The communication can be wireless and its range may vary. All devices, whether deployed in homes, in buildings, in a grid system, or in industry should be reliable and have less cost [256], [272].

The recently successful approaches of D2D-based groups are known to be attractive in sensor network environments with a huge amount of devices because of their effective depression of data congestion and outstanding power efficiency. However, they have been generally developed only for environments with fully reliable D2D links. Proposed technique [307] aims to cope with this issue. It thoroughly investigates the performance of recently flourishing approaches of D2D-based grouped with respect to D2D link reliability. It is shown that unreliable D2D links can significantly reduce the performance of an advanced MAC frame structure of grouped massive machine-type communication (mMTC) transmission and a geolocation database (GDB) signaling approach. Therefore, it is proposed that an enhanced protocol that embeds extra signaling overhead into the user plane data packets in order to reduce the increased requests caused by D2D link exceptions.

TABLE XIV
IoTs Applications, Features, RTOS and Products

Popularity		#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
Area of Application		**Smart Homes**	**Wearables**	**Smart City**	**Smart Grid**	**Industries**	**Smart Traffic**	**Medical**	**Smart retail**	**Smart supply chain**	**Smart Agricultural**
Features [1]–[5]	**Connection to the Internet**	Wifi,3g,4g, Bluetooth	Wifi,3g,4g, Bluetooth	Wi-Fi, Satellite Communication,4g	Wi-Fi, Satellite Communication,4g	Wi-Fi, Satellite Communication	Wi-Fi, 3g, 4g	Wi-Fi, 3g, 4g	Wi-Fi, 3g, 4g	Wi-Fi, Satellite Communication	Wi-Fi, Satellite Communication
	Network Size	Small	Small	Large	Large	Large	Large	Large	Small	Large	Large
	Bandwidth consumption	Low	Low	High	Medium	Large	Medium	Medium	Medium	Medium	Large
	Real-Time	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Reaction of device	Nonfictions	Nonfictions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions	Nonfictions, other solutions
	Visualization	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Data management	Local	Local	Shared	Shared	Shared	Shared	Shared	Shared	Shared	Shared
	User interaction	Phones, touch, speech, any other display	Phones, touch, speech, text, any other display	Phones, touch, gestures, any other display	Phones, touch, gestures, speech, text any other display	Phones, any other display	Phones, touch, gestures, speech, text any other display	Phones, touch, gestures, speech, text any other display	Phones, touch, speech, text, any other display	Phones, PC, touch, text, any other display	Phones, any other display
RTOS [258]–[260]	**Contiki**	✓	–	✓	✓	✓	✓	✓	Unknown	–	–
	TinyOS	✓	–	✓	✓	✓	✓	✓	–	–	–
	RIOT	✓	✓	✓	✓	✓	✓	✓	–	–	–
	Nano-RK	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	✓	Unknown	Unknown	Unknown
	LiteOS	✓	✓	✓	✓	✓	✓	✓	–	–	–
	MantisOS	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
	SOS	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
	RETOS	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown	Unknown
Products [17], [266], [271], [280], [283], [285], [289], [292], [296], [301]		Lights, locks, thermostat, fridges fire alarm	Watches, bands, glasses, footwear, wrist devices and drums	Waste and water management systems, air population, weather	Digital meters	Monitors, alarms, thermostat, locks	Sensors to monitor traffic density, remote locks, tracking location	Hearing aid, ECG monitors	Wall signage and monitors, security tags	RFID tags, data location and tracking	Thermostats, water and soil management and weather updates
Functions [265], [270], [271], [282], [286]		Provides security and accessibility	Better functionality and small.	Monitoring and better management	Monitoring and better management for energy usage	Security for people and less workforce	Security through remote locks and tracking locations	Monitoring and management of patients health	Monitoring customer choice and enhancing efficiency	Rapid and reliable delivery and easy placement of order	Monitoring and better management of available resources

C. Power Consumption

Devices should be able to communicate with less battery usage because these devices are deployed far away and they need to run on batteries. Therefore, batteries consumption should be at a minimum for maximum days. In the coming years, this could become a major issue in IoTs. Efficiency in terms of power is essential, therefore, efficient programming methods and scheduling methods are required. Mechanical energy harvesting is a necessary part of IoTs [302], [308].

D. Bandwidth

IoTs devices are nothing without the Internet. They need good bandwidth to communicate over a particular channel. There are hundreds of devices communicating over the global network at same instance. A device should consume enough bandwidth to perform its function as expected to perform required functionality [309].

E. Interoperability

Interoperability is the major challenge for IoTs because IoTs requires the devices to communicate with one another. This is what IoTs is all about. At the user level, the user can have multiple devices running on multiple platforms. The user will not want to get stuck with one device. Instead, a user may want diversity in case of device choice [310]–[315].

IoTs devices are deployed far for months or year to collect data or perform any other function. Data collection should have less redundancy and mistakes. Otherwise, it can cause any serious damage. Such as if we talk about health or traffic these devices should perform well. Developers have more responsibility now. Moreover, these devices can be deployed for months it is necessary that they have longer life spam and reliable communication. Furthermore, these devices can have some personal information about homes, cars, mobile phone, therefore, security is a major concern as well.

XIII. Case Studies

A. OpenWSN

OpenWSN [316] is an open source project developed in 2010 and licensed under BSD, which promotes and implements a standard-based protocol for IoTs. IEEE802.15.4e implements OpenWSN because of its technique is known as time synchronized channel hopping (TSCH). It has enabled maximum reliability and low-power operation through synchronized nodes. It exploits Berkeley socket abstraction. Therefore, abstraction is as simple as simple Internet host. As a result, OpenWSN supports IoTs multiple platforms such as TelosB, low pin count (LPC), the guidance and inertial navigation assistant (GINA), and k20.

IEEE802.12.4e has replaced typical MAC layer despite changing underlying PHY layer. Therefore, it is implemented as a software update and it is mainly written in C language. OpenWSN implements 6LoWPAN, CoAP and RPL. It implements 6TiSCH stack and simple graphical user interface (GUI). Power efficiency is provided by scheduler in deep sleep mode in its idle state or when the queue is empty.

B. Google Brillo

Google announced Brillo in early 2015. Since then it has been proved that a portable OS to IoTs devices including advanced RISC machines (ARM), million instructions per second (MIPS-base), and intel. Brillo targets devices with minimum footprint and has RAM less than 32MB and 64MB. Furthermore, a communication platform is used. It is a common standards communications layer that implements a common language to communicate with the platform's sensors. Google Brillo, current application includes mostly smart wearable for monitoring a person health and heart rate. Moreover, phone communication is supported through Brillo. Brillo is a winner for requiring fewer hardware resources. However, it can not be focused on tiny devices [48], [317].

C. FreeRTOS

FreeRTOS [23], [28] was developed in 2002, licensed under general public license (GPL). It aims to be simple in abstraction, portable to devices, easy for developers, and user. FreeRTOS implements C language as its programming language and has the simplest architecture of just four files. It is a multi-threaded OS which implements mutexes and semaphores. It has a preemptive scheduler and implements round-robin as its scheduling algorithm. FreeRTOS supports real-time task by using deterministic operations. It does not provide networking techniques however, third parties are used for this purpose. It does not have its own simulator therefore, it is dependent on other simulator for debugging and testing.

D. MbedOS

A recent release by ARM is known as MbedOS and it mainly focuses on 32 bit ARM embedded system [50]. MbedOS is a full stack OS. It is preferred for highly energy constrained micro controllers and it provides energy efficient through its programming model. MbedOS has a built in connectivity and security. Furthermore, it provides re-usability and its application framework is written in C++. MbedOS implements IPv4 and IPv6. Furthermore, it support 6LoWPN, hypertext transfer protocol (HTTP), and CoAP protocols. MbedOS provides security with built-in libraries, cryptography mechanism. It implements transport layer security (TLS) and data gram transport layer security (DTLS) protocols.

Above mentioned OSs are also getting a lot of hype these days. They have remarkable features and some drawback. Later on, with the increasing application, the OS with maximum performance will win over all other OSs.

XIV. Conclusion

The IoTs is already a reality. With minimum human involvement, embedded devices are communicating with each other through the Internet and soon there will be billions of them and IoTs will transform the way we live. As the IoTs is becoming more and more ubiquitous, the demands on technologies are increasing. These devices are low in resources therefore, a suitable and efficient system is required to manage the usage of these resources. Suitable solutions are required to prevent device failure, to manage the system, and to provide secure communication. In this review, we presented a comparative overview of OSs, communication technologies, protocols, innovative application and associated challenges for the IoTs. We analyzed the key requirements for IoTs applications their required architecture, proposed algorithms, language support, management of power and memory. We concluded that there are various areas for application of IoTs and multiple OSs have been proposed. However, they should be selected according to the specific application requirements. This study mainly focused on open source OSs. Some of them are real-time and involve the latest networking technologies, while others do not. Contiki and RIOT are the most widely used OS for IoTs application as they fulfill the most of the requirements. For IoTs, an OS should also provide real-time functionalities and should employ 6loWPAN protocols for communication. IoTs applications are affecting many areas and this paper presented these applications according to their use in the IoTs field. This paper provided an overview of the major challenges facing IoTs and we concluded that security, privacy, and interoperability are the main challenges faced by the IoTs.

REFERENCES

[1] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[2] C. Perera, C. H. Liu, S. Jayawardena, and M. Chen, "A survey on Internet of Things from industrial market perspective," *IEEE Access*, vol. 2, pp. 1660–1679, 2014.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, "Fog computing and its role in the Internet of Things," in *Proc. 1st Ed. MCC Workshop Mobile Cloud Comput.*, Helsinki, Finland, 2012, pp. 13–16.

[4] F. Wang, L. Hu, J. Zhou, and K. Zhao, "A survey from the perspective of evolutionary process in the Internet of Things," *Int. J. Distrib. Sensor Netw.*, vol. 11, Jan. 2015, Art. no. 9.

[5] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: Vision, applications and research challenges," *Ad Hoc Netw.*, vol. 10, no. 7, pp. 1497–1516, 2012.

[6] D. Uckelmann, M. Harrison, and F. Michahelles, *Architecting the Internet of Things*. Heidelberg, Germany: Springer, Jan. 2011, pp. 1–353.

[7] L. D. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.

[8] P. Barnaghi, W. Wang, C. Henson, and K. Taylor, "Semantics for the Internet of Things," *Int. J. Semantic Web Inf. Syst.*, vol. 8, no. 1, pp. 1–21, 2012.

[9] X. Jia, Q. Feng, T. Fan, and Q. Lei, "RFID technology and its applications in Internet of Things (IoT)," in *Proc. 2nd Int. Conf. Consum. Electron. Commun. Netw. (CECNet)*, Yichang, China, 2012, pp. 1282–1285.

[10] L. Li, H. Xiaoguang, C. Ke, and H. Ketai, "The applications of WiFi-based wireless sensor network in Internet of Things and smart grid," in *Proc. 6th IEEE Conf. Ind. Electron. Appl. (ICIEA)*, Beijing, China, 2011, pp. 789–793.

[11] R. Oliver, "Reprogramming embedded systems at run-time," in *Proc. 8th Int. Conf. Sens. Technol.*, Liverpool, U.K., 2014, pp. 124–129.

[12] A. Mallikarjuna, R. V. Avu, P. Kumar, D. Janakiram, and G. A. Kumar, "Operating systems for wireless sensor networks: A survey technical report," Distrib. Object Syst. Lab, Dept. Comput. Sci. Eng., Indian Inst. Technol. Madras, Chennai, India, Rep., pp. 1–30, 2007.

[13] L. Coetzee and J. Eksteen, "The Internet of Things—Promise for the future? An introduction," in *Proc. IST Africa Conf.*, 2011, pp. 1–9.

[14] Y. Liu and G. Zhou, "Key technologies and applications of Internet of Things," in *Proc. 5th Int. Conf. Intell. Comput. Technol. Autom.*, 2012, pp. 197–200.

[15] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A comparative study of wireless protocols: Bluetooth, UWB, ZigBee, and Wi-Fi," in *Proc. Ind. Electron. Conf. (IECON)*, Taipei, Taiwan, 2007, pp. 46–51.

[16] P. Baronti *et al.*, "Wireless sensor networks: A survey on the state of the art and the 802.15.4 and ZigBee standards," *Comput. Commun.*, vol. 30, no. 7, pp. 1655–1695, 2007.

[17] M. Darianian and M. P. Michael, "Smart home mobile RFID-based Internet-of-Things systems and services," in *Proc. Int. Conf. Adv. Comput. Theory Eng. (ICACTE)*, 2008, pp. 116–120.

[18] N. Montavont and T. Noël, "Handover management for mobile nodes in IPv6 networks," *IEEE Commun. Mag.*, vol. 40, no. 8, pp. 38–43, Aug. 2002.

[19] K.-S. Kong, W. Lee, Y.-H. Han, M.-K. Shin, and H. You, "Mobility management for all-IP mobile networks: Mobile IPv6 vs. proxy mobile IPv6," *IEEE Wireless Commun.*, vol. 6, no. 2, pp. 36–45, Apr. 2008.

[20] V. Vujović and M. Maksimović, "Raspberry Pi as a wireless sensor node: Performances and constraints," in *Proc. 37th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. (MIPRO)*, May 2014, pp. 1013–1018.

[21] H. C. Pöhls, "JSON sensor signatures (JSS): End-to-end integrity protection from constrained device to IoT application," in *Proc. 9th Int. Conf. Innov. Mobile Internet Services Ubiquitous Comput. (IMIS)*, Blumenau, Brazil, 2015, pp. 306–312.

[22] J. Betthauser *et al.*, "WolfBot: A distributed mobile sensing platform for research and education," in *Proc. Zone 1 Conf. Amer. Soc. Eng. Educ.*, Bridgeport, CT, USA, 2014, pp. 1–8.

[23] J. J. Livingston and A. Umamakeswari, "Internet of Things application using IP-enabled sensor node and Web server," *Indian J. Sci. Technol.*, vol. 8, pp. 207–212, May 2015.

[24] S. Farah *et al.*, "Real-time microwave remote laboratory architecture," in *Proc. 45th Eur. Microw. Conf. Week Freedom Through Microw. (EuMW)*, Paris, France, 2015, pp. 1315–1318.

[25] O. Hahm, E. Baccelli, H. Petersen, and N. Tsiftes, "Operating systems for low-end devices in the Internet of Things: A survey," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 720–734, Oct. 2016.

[26] W. Dong, C. Chen, X. Liu, and J. Bu, "Providing OS support for wireless sensor networks: Challenges and approaches," *IEEE Commun. Surveys Tuts.*, vol. 12, no. 4, pp. 519–530, 4th Quart., 2010.

[27] M. O. Farooq and T. Kunz, "Operating systems for wireless sensor networks: A survey," *Sensors*, vol. 11, no. 6, pp. 5900–5930, 2011.

[28] T. B. Chandra, P. Verma, and A. K. Dwivedi, "Operating systems for Internet of Things: A comparative study," in *Proc. 2nd Int. Conf. Inf. Commun. Technol. Competitive Strategies*, 2016, p. 47.

[29] P. Gaur and M. P. Tahiliani, "Operating systems for IoT devices: A critical survey," in *Proc. IEEE Region 10 Symp. (TENSYMP)*, 2015, pp. 33–36.

[30] D. Willmann, "Contiki—A memory-efficient operating system for embedded smart objects," pp. 1–12, 2009.

[31] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, Tampa, FL, USA, 2004, pp. 455–462.

[32] A. A. Hasbollah, S. H. S. Ariffin, and M. I. A. Hamini, "Performance analysis for 6loWPAN IEEE 802.15.4 with IPv6 network," in *Proc. IEEE Region 10 Annu. Int. Conf. (TENCON)*, Singapore, 2009, pp. 1–5.

[33] J. G. Ko, N. Tsiftes, A. Dunkels, and A. Terzis, "Pragmatic low-power interoperability: ContikiMAC vs TinyOS LPL," in *Proc. Annu. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw. Workshops*, vol. 1. Seoul, South Korea, 2012, pp. 94–96.

[34] E. Baccelli, O. Hahm, M. Günes, M. Wahlisch, and T. C. Schmidt, "RIOT OS: Towards an OS for the Internet of Things," in *Proc. 32nd IEEE*, Turin, Italy, 2013, pp. 2453–2454.

[35] E. Baccelli, O. Hahm, M. Wählisch, M. Günes, and T. Schmidt, "RIOT: One OS to rule them all in the IoT," INRIA, Rocquencourt, France, Rep. RR-8176, Dec. pp. 1–16, 2012.

[36] O. Hahm, E. Baccelli, H. Petersen, M. Wählisch, and T. C. Schmidt, "Demonstration abstract: Simply RIOT—Teaching and experimental research in the Internet of Things," in *Proc. 13th Int. Symp. Inf. Process. Sensor Netw. (Part CPS Week) (IPSN)*, Berlin, Germany, 2014, pp. 329–330.

[37] A. Eswaran, A. Rowe, and R. Rajkumar, "Nano-RK: An energy-aware resource-centric RTOS for sensor networks," in *Proc. Real Time Syst. Symp.*, Miami, FL, USA, 2005, pp. 256–265.

[38] C. M. Byers, Y. Yang, Y. Li, and K. Bhaskar, "A reference for the nano-RK real-time operating system," 2010.

[39] J. Ma, X. Zhou, S. Li, and Z. Li, "Connecting agriculture to the Internet of Things through sensor networks," in *Proc. IEEE Int. Conf. Internet Things Cyber Phys. Soc. Comput. (iThings/CPSCom)*, Dalian, China, Oct. 2011, pp. 184–187.

[40] S. Bhatti *et al.*, "MANTIS OS: An embedded multithreaded operating system for wireless micro sensor platforms," *Mobile Netw. Appl.*, vol. 10, no. 4, pp. 563–579, 2005.

[41] Y. Mazzer and B. Tourancheau, "Comparisons of 6LoWPAN implementations on wireless sensor networks," in *Proc. 3rd Int. Conf. Sensor Technol. Appl. (SENSORCOMM)*, 2009, pp. 689–692.

[42] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, "A dynamic operating system for sensor nodes," in *Proc. MobiSys*, Seattle, WA, USA, 2005, pp. 163–176, .

[43] H. Cha *et al.*, "RETOS: Resilient, expandable, and threaded operating system for wireless sensor networks," in *Proc. IPSN*, Cambridge, MA, USA, May 2007, pp. 148–157.

[44] H. Cha *et al.*, "The RETOS operating system: Kernel, tools and applications," in *Proc. 6th Int. Symp. Inf. Process. Sensor Netw.*, Cambridge, MA, USA, 2007, pp. 559–560.

[45] D. Mccracken, "POSIX threads and the Linux kernel," in *Proc. Ottowa Linux Symp.*, 2002, pp. 330–337.

[46] M. Insights and I. Technology, "Dell—Taking part of the mystery out of the industrial Internet of Things," Moor Insights Strategy, Rep., pp. 1–14, Dec. 2015.

[47] J. Whipple, W. Arensman, and M. S. Boler, "A public safety application of GPS-enabled smartphones and the android operating system," in *Proc. IEEE Int. Conf. Syst. Man Cybern.*, San Antonio, TX, USA, Oct. 2009, pp. 2059–2061.

[48] R. Rajkumar and M. Sruthi, "A study on development issues over IoT platforms, protocols and operating system," in *Proc. Int. Conf. Innov. Inf. Embedded Commun. Syst. (ICIIECS)*, Coimbatore, India, Oct. 2016, pp. 4–7.

[49] H. Wei *et al.*, "RGMP-ROS: A real-time ROS architecture of hybrid RTOS and GPOS on multi-core processor," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, 2014, pp. 2482–2487.

[50] M. Asim, "IoT operating systems and security challenges," *Int. J. Comput. Sci. Inf. Security*, vol. 14, no. 7, pp. 314–318, 2016.

[51] K. Elphinstone and G. Heiser, "From L3 to seL4 what have we learnt in 20 years of L4 microkernels?" in *Proc. 24th ACM Symp. Oper. Syst. Principles*, Farmington, PA, USA, 2013, pp. 133–150.

[52] S. Carta *et al.*, "Multi-processor operating system emulation framework with thermal feedback for systems-on-chip," in *Proc. 17th ACM Great Lakes Symp. VLSI*, 2007, pp. 311–316.

[53] C. Berger, A. M. M. Abdullah, and J. Hansson, "Cots-architecture with a real-time OS for a self-driving miniature vehicle," in *Proc. 32nd Int. Conf. Comput. Safety Rel. Security SAFECOMP Workshop ASCoMS Archit. Safety Collaborative Mobile Syst.*, 2013, pp. 1–12.

[54] P. Pagano *et al.*, "ERIKA and open-ZB: An implementation for real-time wireless networking," in *Proc. ACM Symp. Appl. Comput.*, Honolulu, HI, USA, 2009, pp. 1687–1688.

[55] G. Strazdins, A. Elsts, and L. Selavo, "Mansos: Easy to use, portable and resource efficient operating system for networked embedded devices," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*. Zürich, Switzerland, 2010, pp. 427–428.

[56] S. C. Kim, H. Kim, J. Song, M. Yu, and P. Mah, "Nanoqplus: A multi-threaded operating system with memory protection mechanism for WSNs," *Proc. CKWSN*, vol. 20, no. 8, Oct. 2008.

[57] D. Hildebrand, "An architectural overview of QNX," in *Proc. USENIX Workshop Microkernels Other Kernel Archit.*, 1992, pp. 113–126.

[58] M. Barabanov, "A Linux-based real-time operating system," M.S. thesis, Dept. Comput. Sci., New Mexico InstiNle Min. Technol., Socorro, NM, USA, 1997.

[59] R. Kaiser and S. Wagner, "Evolution of the PikeOS microkernel," in *Proc. 1st Int. Workshop Microkernels Embedded Syst.*, 2007, pp. 50–57.

[60] H. Trsek, "Internet of Things at work plug-and-play for industrial automation," 2013.

[61] P. Diogo, L. P. Reis, and N. V. Lopes, "Internet of Things: A system's architecture proposal," in *Proc. Iberian Conf. Inf. Syst. Technol. (CISTI)*, Barcelona, Spain, 2014, pp. 1–6.

[62] A. Kanuparthi, R. Karri, and S. Addepalli, "Hardware and embedded security in the context of Internet of Things," in *Proc. ACM Workshop Security Privacy Dependability Cyber Veh. (CyCAR)*, Berlin, Germany, 2013, pp. 61–64.

[63] R. Ramdhany and G. Coulson, "Towards the coexistence of divergent applications on smart city sensing infrastructure," in *Proc. CEUR Workshop*, vol. 1002, 2013, pp. 26–30.

[64] M. Mozumdar, Z. Y. Song, L. Lavagno, and A. L. Sangiovanni-Vincentelli, "A model-based approach for bridging virtual and physical sensor nodes in a hybrid simulation framework," *Sensors*, vol. 14, no. 6, pp. 11070–11096, 2014.

[65] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, and R. Katz, "Above the clouds: A Berkeley view of cloud computing," Dept. Elect. Eng. Comput. Sci., Univ. California, at Berkeley, Berkeley, CA, USA, Rep. UCB/EECS-2009-28, pp. 7–13, 2009.

[66] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *Proc. 5th Int. Conf. Stochastic Algorithms Found. Appl.*, 2009, pp. 169–178.

[67] A. Rowe, R. Mangharam, and R. Rajkumar, "FireFly: A time synchronized real-time sensor networking platform," in *Wireless Ad Hoc Networking: Personal-Area, Local-Area, and the Sensory-Area Network*, Citeseer, 2006.

[68] J. L. Hill and D. E. Culler, "Mica: A wireless platform for deeply embedded networks," *IEEE Micro*, vol. 22, no. 6, pp. 12–24, Nov./Dec. 2002.

[69] M. Pajic *et al.*, "Architecture for a fully distributed wireless control network," in *Proc. 10th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, Chicago, IL, USA, 2011, pp. 117–118.

[70] B. Mazumder and J. O. Hallstrom, "A fast, lightweight, and reliable file system for wireless sensor networks," in *Proc. EMSOFT*, Pittsburgh, PA, USA, 2016, pp. 1–10.

[71] J. Polastre, R. Szewczyk, and D. Culler, "Telos: Enabling ultra-low power wireless research," in *Proc. 4th Proc. Int. Symp. Inf. Process. Sensor Netw. (IPSN)*, Boise, ID, USA, 2005, pp. 364–369.

[72] I. Khan, F. Belqasmi, R. Glitho, and N. Crespi, "A multi-layer architecture for wireless sensor network virtualization," in *Proc. 6th Joint IFIP Wireless Mobile Netw. Conf. (WMNC)*, Dubai, UAE, 2013, pp. 1–4.

[73] L. F. Wanner, A. S. Hoeller, A. B. D. Oliveira, and A. A. Frohlich, "Operating system support for data acquisition in sensor networks," in *Proc. IEEE Conf. Emerg. Technol. Factory Autom. (ETFA)*, Prague, Czech Republic, 2006, pp. 582–585.

[74] R. Lajara, J. Pelegrí-Sebastiá, and J. J. P. Solano, "Power consumption analysis of operating systems for wireless sensor networks," *Sensors*, vol. 10, no. 6, pp. 5809–5826, 2010.

[75] E. Trumpler and R. Han, "A systematic framework for evolving TinyOS," in *Proc. IEEE Workshop Embedded Netw. Sensors*, 2006, pp. 61–65.

[76] M. Healy, T. Newe, and E. Lewis, "Power management in operating systems for wireless sensor nodes," in *Proc. IEEE Sensors Appl. Symp. (SAS)*, San Diego, CA, USA, Feb. 2007, pp. 1–6.

[77] M. Sirisha and S. Swetha, "A survey on WSN OS using real-time scheduling strategy," *Int. J. Technol. Enhancement Emerg. Eng. Res.*, vol. 1, no. 5, pp. 13–19, 2013.

[78] C. Emmanouilidis, S. Katsikas, and C. Giordamlis, "Wireless condition monitoring and maintenance management: A review and a novel application development platform," in *Proc. 3rd World Congr. Eng. Asset Manag. Intell. Maintenance Syst. Conf.*, 2008, pp. 2030–2041.

[79] L. F. Friedrich, "A survey on operating system support for embedded systems properties," in *Proc. Anais do Workshop de Sistemas Operacionais VI (JULIO)*, 2009, pp. 2393–2404.

[80] A. Hornsby and E. Bail, "μXMPP: Lightweight implementation for low power operating system Contiki," in *Proc. Int. Conf. Ultra Mod. Telecommun. Workshops*, St. Petersburg, Russia, 2009, pp. 1–5.

[81] M. Kovatsch, S. Duquennoy, and A. Dunkels, "A low-power CoAP for Contiki," in *Proc. 8th IEEE Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Valencia, Spain, 2011, pp. 855–860.

[82] T. Luo, H.-P. Tan, and T. Q. S. Quek, "Sensor OpenFlow: Enabling software-defined wireless sensor networks," *IEEE Commun. Lett.*, vol. 16, no. 11, pp. 1896–1899, Nov. 2012.

[83] B. Porter and G. Coulson, "Lorien: A pure dynamic component-based operating system for wireless sensor networks," in *Proc. 4th Int. Workshop Middleware Tools Services Run Time Support Sensor Netw. (MidSens)*, Champaign, IL, USA, 2009, pp. 7–12.

[84] L. Saraswat and P. S. Yadav, "A comparative analysis of wireless sensor network operating systems," in *Proc. INDIACom*, 2011, May, p. 4.

[85] M. L. Dertouzos and A. K. Mok, "Multiprocessor online scheduling of hard-real-time tasks," *IEEE Trans. Softw. Eng.*, vol. 15, no. 12, pp. 1497–1506, Dec. 1989.

[86] R. Shimizu, K. Tei, Y. Fukazawa, and S. Honiden, "Toward a portability framework with multi-level models for wireless sensor network software," in *Proc. Int. Conf. Smart Comput. (SMARTCOMP)*, Hong Kong, 2014, pp. 253–260.

[87] X. Su, M. Wu, and J. Xu, "A novel virtual storage area network solution for virtual desktop infrastructure," in *Proc. Int. Symp. Wireless Pers. Multimedia Commun. (WPMC)*, vol. 2015. Sydney, NSW, Australia, 2015, pp. 204–208.

[88] H. Wang, Z. Chen, G. Xiao, and Z. Zheng, "Network of networks in Linux operating system," *Physica A Stat. Mech. Appl.*, vol. 447, pp. 520–526, Apr. 2016.

[89] H. Will, K. Schleiser, and J. Schiller, "A real-time kernel for wireless sensor networks employed in rescue scenarios," in *Proc. Conf. Local Comput. Netw. (LCN)*, Zürich, Switzerland, Oct. 2009, pp. 834–841.

[90] A. Rowe, D. Goel, and R. Rajkumar, "FireFly mosaic: A vision-enabled wireless sensor networking system," in *Proc. Real Time Syst. Symp.*, Tucson, AZ, USA, 2007, pp. 459–468.

[91] N. Dziengel *et al.*, "Energy-aware distributed fence surveillance for wireless sensor networks," in *Proc. 7th Int. Conf. Intell. Sensors Sensor Netw. Inf. Process. (ISSNIP)*, Adelaide, SA, Australia, 2011, pp. 353–358.

[92] J. Yang, Y. Xie, and T. Chen, "Research on Web server application on multi-core embedded system," in *Proc. Int. Conf. Embedded Softw. Syst. (ICESS)*, Zhejiang, China, 2009, pp. 412–416.

[93] M. Poongothai, "ARM embedded Web server based on DAC system," in *Proc. Int. Conf. Process Autom. Control Comput. (PACC)*, 2011, pp. 1–5.

[94] Q. Cao, T. Abdelzaher, J. Stankovic, and T. He, "The LiteOS operating system: Towards unix-like abstractions for wireless sensor networks," in *Proc. 7th Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, St. Louis, MO, USA, 2008, pp. 233–244.

[95] H. Dai, M. Neufeld, and R. Han, "ELF: An efficient log-structured flash file system for micro sensor nodes," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst.*, Baltimore, MD, USA, 2004, pp. 176–187.

[96] Q. Cao, T. Abdelzaher, J. Stankovic, K. Whitehouse, and L. Luo, "Declarative tracepoints," in *Proc. SenSys*, Raleigh, NC, USA, 2008, pp. 85–98.

[97] H. Cha and I. Jung, "RMTool: Component-based network management system for wireless sensor networks," in *Proc. 4th IEEE Consum. Commun. Netw. Conf.*, Las Vegas, NV, USA, 2007, pp. 614–618.

[98] Y. Kim, H. Shin, and H. Cha, "Y-MAC: An energy-efficient multichannel MAC protocol for dense wireless sensor networks," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, 2008, pp. 53–63.

[99] A. Rajan, "A survey on MAC protocols for wireless sensor networks and cognitive radio sensor networks design," *Int. J. Sci. Eng. Appl. Sci.*, vol. 1, no. 6, pp. 112–119, 2015.

[100] J. J. M. Castillo and K. A. Rodriguez, "Security architecture for ad hoc NOMOHi networks: Development of a project based on emergency rural telecommunications," in *Proc. World Congr. Internet Security (WorldCIS)*, Guelph, ON, Canada, 2012, pp. 183–187.

[101] B. Kerkez, T. Watteyne, M. Magliocco, S. Glaser, and K. Pister, "Feasibility analysis of controller design for adaptive channel hopping," in *Proc. 4th Int. ICST Conf. Perform. Eval. Methodol. Tools*, Pisa, Italy, 2009, pp. 1–6.

[102] G. Ekbatanifard, "Multi-channel medium access control protocols for wireless sensor networks: A survey," *J. Comput. Eng.*, vol. 1, no. 2009, pp. 3–11, 2011.

[103] A. Kaliszan and P. Zwierzykowski, "Application of real time operating system in the Internet of Things," in *Proc. Proc. 10th Int. Symp. Commun. Syst. Netw. Digit. Signal Process. (CSNDSP)*, Prague, Czech Republic, 2016, pp. 1–6.

[104] Z. Zeng, A. Liu, D. Li, and J. Long, "A highly efficient DAG task scheduling algorithm for wireless sensor networks," in *Proc. 9th Int. Conf. Young Comput. Sci.*, Hunan, China, 2008, pp. 570–575.

[105] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. Assoc. Comput. Mach.*, vol. 20, no. 1, pp. 46–61, 1973.

[106] K. Akkaya and M. Younis, "An energy-aware QoS routing protocol for wireless sensor networks," in *Proc. 23rd Int. Conf. Distrib. Comput. Syst. Workshops*, Providence, RI, USA, 2003, pp. 710–715.

[107] M. B. Jones *et al.*, "An overview of the Rialto real-time architecture," in *Proc. SIGOPS Eur. Workshop (SIGOPS)*, 1996, pp. 249–256.

[108] M. Kaladevi and S. Sathiyabama, "A comparative study of scheduling algorithms for real time task," *Int. J. Adv. Sci. Technol.*, vol. 1, no. 4, pp. 8–14, 2010.

[109] M. E. Mustafa, "The effect of queuing mechanisms first in first out (FIFO), priority queuing (PQ) and weighted fair queuing (WFQ) on network's routers and applications," *Int. J. Eng. Innov. Res.*, vol. 4, no. 1, pp. 188–191, 2015.

[110] Y. C. R. Ramesh, "Design of real time scheduler simulator and development of modified round Robin architecture," *Int. J. Comput. Sci. Netw. Security*, vol. 10, no. 3, pp. 43–47, 2010.

[111] L. Zhang and F. Chen, "A round-Robin scheduling algorithm of relay-nodes in WSN based on self-adaptive weighted learning for environment monitoring," *J. Comput.*, vol. 9, no. 4, pp. 830–835, 2014.

[112] T.-W. T.-w. Kuo and C.-h. C.-H. Li, "A fixed-priority-driven open environment for real-time applications," in *Proc. 20th IEEE Real Time Syst. Symp. (RTSS)*, Phoenix, AZ, USA, 1999, pp. 256–267.

[113] K. Jeffay and D. Stone, "Accounting for interrupt handling costs in dynamic priority task systems," in *Proc. Syst. Symp.*, Dec. 1993, pp. 212–221.

[114] M. Amjad, M. Sharif, M. K. Afzal, and S. W. Kim, "TinyOS-new trends, comparative views, and supported sensing applications: A review," *IEEE Sensors J.*, vol. 16, no. 9, pp. 2865–2889, May 2016.

[115] G. C. Buttazzo, "Rate monotonic vs. EDF: Judgment day," *Real Time Syst.*, vol. 29, no. 1, pp. 5–26, 2005.

[116] I. S. Rajput and D. Gupta, "A priority based round robin CPU scheduling algorithm for real time systems," *Int. J. Innov. Eng. Technol.*, vol. 1, no. 3, pp. 1–11, 2012.

[117] J. P. Lehoczky, L. Sha, and Y. Ding, "The rate monotonic scheduling algorithm: Exact characterization and average case behavior," in *Proc. Real Time Syst. Symp.*, 1989, p. 5.

[118] N. C. Audsley, A. Burns, M. F. Richardson, and A. J. Wellings, "Hard real-time scheduling: The deadline-monotonic approach," in *Proc. IEEE Workshop Real Time Oper. Syst. Softw.*, 1991, pp. 133–137.

[119] F. Zhang and A. Burns, "Schedulability analysis for real-time systems with EDF scheduling," *IEEE Trans. Comput.*, vol. 58, no. 9, pp. 1250–1258, Sep. 2009.

[120] X. Li and X. He, "The improved EDF scheduling algorithm for embedded real-time system in the uncertain environment," in *Proc. IEEE Conf.*, vol. 4. Chengdu, China, 2010, pp. V4-563–V4-566.

[121] C. Lu, J. A. Stankovic, G. Tao, and S. H. Son, "Design and evaluation of a feedback control EDF scheduling algorithm," in *Proc. RTSS*, Phoenix, AZ, USA, 1999, pp. 56–67.

[122] S.-H. Oh and S.-M. Yang, "A modified least-laxity-first scheduling algorithm for real-time tasks," in *Proc. 5th Int. Conf. Real Time Comput. Syst. Appl.*, 1998, pp. 31–36.

[123] W. Zhang, S. Teng, Z. Zhu, X. Fu, and H. Zhu, "An improved least-laxity-first scheduling algorithm of variable time slice for periodic tasks," in *Proc. 6th IEEE Int. Conf. Cogn. Informat. (ICCI)*, 2007, pp. 548–553.

[124] P. Li and B. Ravindran, "Fast, best-effort real-time scheduling algorithms," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1159–1175, Sep. 2004.

[125] S. Baskiyar and N. Meghanathan, "A survey of contemporary real-time operating systems," *Informatica Slovenia*, vol. 29, no. 2, pp. 233–240, 2005.

[126] H. G. Hawkins, D. L. Picha, M. D. Wooldridge, F. K. Greene, and G. Brinkmeyer, "Performance comparison of three freeway guide sign alphabets," vol. 1692, no. 1, pp. 9–16, 1999.

[127] H. Kaneko, J. A. Stankovic, S. Sen, and K. Ramamritham, "Integrated scheduling of multimedia and hard real-time tasks," in *Proc. Conf. Converg. Technol. AsiaPacific Region (TENCON)*, vol. 4. Washington, DC, USA, 2003, pp. 206–217.

[128] M. Shahrbanoo, M. Ali, and M. Mehran, "An approach for Agile SOA development using Agile principals," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, no. 1, pp. 237–244, 2012.

[129] G. Teng, K. Zheng, and W. Dong, "A survey of available tools for developing wireless sensor networks," in *Proc. 3rd Int. Conf. Syst. Netw. Commun. (ICSNC) Includes I-CENTRIC Int. Conf. Adv. Human Orient. Personalized Mech. Technol. Services*, 2008, pp. 139–144.

[130] S. Raman, "TinyOS an operating system for tiny embedded networked sensors," presented at the Adv. Oper. Syst. Course, 2002, pp. 1–12.

[131] M. Yu, S. Xiahou, and X. Y. Li, "A survey of studying on task scheduling mechanism for TinyOS," in *Proc. Int. Conf. Wireless Commun. Netw. Mobile Comput. (WiCOM)*, 2008, pp. 1–4.

[132] Q. Wang, Y. Zhu, and L. Cheng, "Reprogramming wireless sensor networks: Challenges and approaches," *IEEE Netw.*, vol. 20, no. 3, pp. 48–55, May/Jun. 2006.

[133] R. Sugihara and R. K. Gupta, "Programming models for sensor networks," *ACM Trans. Sensor Netw.*, vol. 4, no. 2, pp. 1–29, 2008.

[134] R. Von Behren, J. Condit, and E. Brewer, "Why events are a bad idea (for high-concurrency servers)," in *Proc. USENIX*, 2003, p. 6.

[135] T. Alliance, "TinyOS 2.1 adding threads and memory protection to TinyOS," in *Proc. 6th ACM Conf. Embedded Netw. Sensor Syst.*, Raleigh, NC, USA, 2008, pp. 413–414.

[136] M. Zheng, J. Sun, Y. Liu, J. S. Dong, and Y. Gu, "Towards a model checker for NesC and wireless sensor networks," in *Proc. 13th Int. Conf. Formal Methods Softw. Eng. (ICFEM)*, vol. 072. Durham, U.K., 2011, pp. 372–387.

[137] R. Gao, H. Zhou, and G. Su, "Structure of wireless sensors network based on TinyOS," in *Proc. Int. Conf. Control Autom. Syst. Eng. (CASE)*, 2011, pp. 1–4.

[138] V. Handziski *et al.*, "Flexible hardware abstraction for wireless sensor networks," in *Proc. 2nd Eur. Workshop Wireless Sensor Netw. (EWSN)*, vol. 2005. Istanbul, Turkey, 2005, pp. 145–157.

[139] D. Gay, P. Levis, and D. Culler, "Software design patterns for TinyOS," *ACM Trans. Embedded Comput. Syst.*, vol. 6, no. 4, pp. 40–49, 2007.

[140] P. Lindgren, M. Lindner, A. Lindner, D. Pereira, and L. M. Pinho, "RTFM-core: Language and implementation," in *Proc. IEEE 10th Conf. Ind. Electron. Appl. (ICIEA)*, 2015, pp. 990–995.

[141] M. Malenko and M. Baunach, "Real-time and security requirements for Internet-of-Things operating systems," in *Internet der Dinge: Echtzeit 2016*. Heidelberg, Germany: Springer, 2016, pp. 33–42.

[142] K. Lorincz, B.-R. Chen, J. Waterman, G. Werner-Allen, and M. Welsh, "Resource aware programming in the Pixie OS," in *Proc. ACM Conf. Embedded Netw. Sensor Syst.*, 2008, pp. 211–224.

[143] M. Pathak, "OS design challenges & research opportunities in real-time WSNs & approach for real time support in Nano-RK," *Compusoft*, vol. 2, no. 7, pp. 198–203, 2013.

[144] J. A. Stankovic and K. Ramamritham, "The spring kernel: A new paradigm for real time operating systems," *ACM Oper. Syst. Rev.*, vol. 23, no. 3, pp. 54–71, 1988.

[145] P. Buonocunto, A. Biondi, and P. Lorefice, "Real-time multitasking in arduino," in *Proc. 9th IEEE Int. Symp. Ind. Embedded Syst. (SIES)*, Pisa, Italy, 2014, pp. 1–4.

[146] M. Takahashi, B. Hussain, and B. Tang, "Demo abstract: Design and implementation of a Web service for LiteOS-based sensor networks," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, San Francisco, CA, USA, 2009, pp. 407–408.

[147] H. Kermajani, C. Gomez, and M. H. Arshad, "Modeling the message count of the trickle algorithm in a steady-state, static wireless sensor network," *IEEE Commun. Lett.*, vol. 16, no. 12, pp. 1960–1963, Dec. 2012.

[148] D. He, S. Chan, S. Tang, and M. Guizani, "Secure data discovery and dissemination based on hash tree for wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 12, no. 9, pp. 4638–4646, Sep. 2013.

[149] T. Dang, N. Bulusu, W.-C. Feng, and S. Park, *DHV: A Code Consistency Maintenance Protocol for Multi-Hop Wireless Sensor Networks* (Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5432. Heidelberg, Germany: Springer, 2009, pp. 327–342.

[150] W. Li, Y. Zhang, and B. Childers, *MCP: An Energy-Efficient Code Distribution Protocol for Multi-Application WSNs* (Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 5516. Heidelberg, Germany: Springer, 2009, pp. 259–272.

[151] C.-Y. Wan, A. T. Campbell, and L. Krishnamurthy, "PSFQ: A reliable transport protocol for wireless sensor networks," in *Proc. 1st ACM Int. Workshop Wireless Sensor Netw. Appl.*, Atlanta, GA, USA, 2002, pp. 1–11.

[152] A. Hagedorn, D. Starobinski, and A. Trachtenberg, "Rateless deluge: Over-the-air programming of wireless sensor networks using random linear codes," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, St. Louis, MO, USA, 2008, pp. 457–466.

[153] R. K. Panta, I. Khalil, and S. Bagchi, "Stream: Low overhead wireless reprogramming for sensor networks," in *Proc. IEEE INFOCOM*, Barcelona, Spain, 2007, pp. 928–936.

[154] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," in *Proc. 2nd Eur. Workshop Wireless Sensor Netw.*, Istanbul, Turkey, Jan./Feb. 2005, 2005, pp. 121–132.

[155] L. Liquori *et al.*, *Synapse: A Scalable Protocol for Interconnecting Heterogeneous Overlay Networks* (Lecture Notes in Computer Science Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 6091. Heidelberg, Germany: Springer, 2010, pp. 67–82.

[156] W. Li *et al.*, "Adaptive buffer management for efficient code dissemination in multi-application wireless sensor networks," in *Proc. 5th Int. Conf. Embedded Ubiquitous Comput. (EUC)*, vol. 1. Shanghai, China, 2008, pp. 295–301.

[157] J. W. Hui and D. Culler, "The dynamic behavior of a data dissemination protocol for network programming at scale," in *Proc. 2nd Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, Baltimore, MD, USA, 2004, pp. 81–94.

[158] H. C. Leligou, C. Massouros, E. Tsampasis, T. Zahariadis, and D. Bargiotas, "Reprogramming wireless sensor nodes," *Int. J. Comput. Trends Technol.*, vol. 1, no. 2, pp. 1–8, 2011.

[159] J. Heidemann and R. Govindan, "Embedded sensor networks," in *Handbook of Networked and Embedded Control Systems*. Boston, MA, USA: Birkhäuser, 2005, pp. 721–738.

[160] B. Baranidharan and B. Shanthi, "A survey on energy efficient protocols for wireless sensor networks," *Int. J. Comput. Appl.*, vol. 11, no. 10, pp. 35–40, 2010.

[161] M. Stolikj, P. J. L. Cuijpers, and J. J. Lukkien, "Efficient reprogramming of wireless sensor networks using incremental updates," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. Workshops (PERCOM Workshops)*, San Diego, CA, USA, 2013, pp. 584–589.

[162] P. Levis *et al.*, "The emergence of networking abstractions and techniques in TinyOS," in *Proc. NSDI*, San Francisco, CA, USA, 2004, p. 1.

[163] J. Carnley, B. Sun, and S. K. Makki, "TORP: TinyOS opportunistic routing protocol for wireless sensor networks," in *Proc. CCNC*, Las Vegas, NV, USA, 2011, pp. 111–115.

[164] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and scalable simulation of entire TinyOS applications," in *Proc. 1st Int. Conf. Embedded Netw. Sensor Syst.*, Los Angeles, CA, USA, 2003, pp. 126–137.

[165] Y. T. Hou and Y. Shi, "Variable bit rate flow routing in wireless sensor networks," *IEEE Trans. Wireless Commun.*, vol. 6, no. 6, pp. 2140–2148, Jun. 2007.

[166] Y. Liang, "Study of protocol for wireless sensor network based on TinyOS," in *Proc. ICCDA*, vol. 2. Qinhuangdao, China, 2010, pp. V2-602–V2-605.

[167] K. Daabaj, "Load-balanced routing scheme for TinyOS-based wireless sensor networks," in *Proc. IEEE Int. Conf. Wireless Inf. Technol. Syst. (ICWITS)*, Honolulu, HI, USA, 2010, pp. 1–4.

[168] W. Wang, J.-H. Youn, and H. R. Sharif, "The implementation of an energy balanced routing protocol with dynamic power scaling in TinyOS," in *Proc. IEEE Int. Conf. Sensor Netw. Ubiquitous Trustworthy Comput.*, vol. 2. Taichung, Taiwan, 2006, pp. 262–267.

[169] S. S. Bhunia, D. K. Sikder, S. Roy, and N. Mukherjee, "A comparative study on routing schemes of IP based wireless sensor network," in *Proc. 9th Int. Conf. Wireless Opt. Commun. Netw. (WOCN)*, Indore, India, Sep. 2012, pp. 1–5.

[170] B. Saadallah, A. Lahmadi, and O. Festor, "CCNx for Contiki: Implementation details," INRIA, Rocquencourt, France, Rep. RT-0432, Nov. 2012.

[171] J. Wu *et al.*, "Context-aware networking and communications: Part 1 [guest editorial]," *IEEE Commun. Mag.*, vol. 52, no. 6, pp. 14–15, Jun. 2014.

[172] C. Perera, A. Zaslavsky, P. Christen, and D. Georgakopoulos, "Context aware computing for the Internet of Things: A survey," *IEEE Commun. Surveys Tuts.*, vol. 16, no. 1, pp. 414–454, 1st Quart., 2014.

[173] S. Faieq, R. Saidi, H. Elghazi, and M. D. Rahmani, "C2IoT: A framework for cloud-based context-aware Internet of Things services for smart cities," *Procedia Comput. Sci.*, vol. 110, pp. 151–158, Jul. 2017.

[174] X. Li, M. Eckert, J.-F. Martinez, and G. Rubio, "Context aware middleware architectures: Survey and challenges," *Sensors*, vol. 15, no. 8, pp. 20570–20607, 2015.

[175] A. S. Thyagaturu, A. Mercian, M. P. McGarry, M. Reisslein, and W. Kellerer, "Software defined optical networks (SDONs): A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 18, no. 4, pp. 2738–2786, 4th Quart., 2016.

[176] A. S. Thyagaturu, Y. Dashti, and M. Reisslein, "SDN-based smart gateways (Sm-GWs) for multi-operator small cell network management," *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 4, pp. 740–753, Dec. 2016.

[177] Z. Sheng *et al.*, "A survey on the IETF protocol suite for the Internet of Things: Standards, challenges, and opportunities," *IEEE Wireless Commun.*, vol. 20, no. 6, pp. 91–98, Dec. 2013.

[178] J. N. Al-Karaki and A. E. Kamal, "Routing techniques in wireless sensor networks: A survey," *IEEE Wireless Commun.*, vol. 11, no. 6, pp. 6–28, Dec. 2004.

[179] S. D. Muruganathan, D. C. F. Ma, R. I. Bhasin, and A. O. Fapojuwo, "A centralized energy-efficient routing protocol for wireless sensor networks," *IEEE Commun. Mag.*, vol. 43, no. 3, pp. S8–S13, Mar. 2005.

[180] S. Husain, A. Prasad, A. Kunz, A. Papageorgiou, and J. Song, "Recent trends in standards related to the Internet of Things and machine-to-machine communications," *J. Inf. Commun. Converg. Eng.*, vol. 12, no. 4, pp. 228–236, 2014.

[181] H. M. Gürsu, M. Vilgelm, W. Kellerer, and M. Reisslein, "Hybrid collision avoidance-tree resolution for M2M random access," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 53, no. 4, pp. 1974–1987, Aug. 2017.

[182] A. F. Molisch *et al.*, "IEEE 802.15.4a channel model-final report," presented at the IEEE P802.15 Working Group for WPANs, 2004, pp. 1–40.

[183] A. J. D. Rathnayaka, V. M. Potdar, and S. J. Kuruppu, "Evaluation of wireless home automation technologies," in *Proc. IEEE Int. Conf. Digit. Ecosyst. Technol.*, Daejeon, South Korea, May/Jun. 2011, pp. 76–81.

[184] J.-F. Wauthy and L. Schumacher, "Implementation of an IEEE 802.15.4-2006 protocol stack on the Texas instrument CC2430," in *Proc. 7th ACM Workshop Perform. Eval. Wireless Ad Hoc Sensor Ubiquitous Netw. (PE-WASUN)*, Bodrum, Turkey, 2010, pp. 33–39.

[185] G. R. González, M. M. Organero, and C. D. Kloos, "Early infrastructure of an Internet of Things in spaces for learning," in *Proc. 8th IEEE Int. Conf. Adv. Learn. Technol. (ICALT)*, Santander, Spain, 2008, pp. 381–383.

[186] P. Urien, "LLCPS: A new security framework based on TLS for NFC P2P applications in the Internet of Things," in *Proc. IEEE 10th Consum. Commun. Netw. Conf. (CCNC)*, Las Vegas, NV, USA, 2013, pp. 845–846.

[187] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "DTLS based security and two-way authentication for the Internet of Things," *Ad Hoc Netw.*, vol. 11, no. 8, pp. 2710–2723, 2013.

[188] S. Raza, T. Voigt, and V. Jutvik, "Lightweight IKEv2: A key management solution for both the compressed IPsec and the IEEE 802.15. 4 security," in *Proc. Workshop Smart Object Security*, 2012.

[189] S. Raza, D. Trabalza, and T. Voigt, "6LoWPAN compressed DTLS for CoAP," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Hangzhou, China, 2012, pp. 287–289.

[190] T. Kothmayr, C. Schmitt, W. Hu, M. Brünig, and G. Carle, "A DTLS based end-to-end security architecture for the Internet of Things with two-way authentication," in *Proc. IEEE Conf.*, Clearwater, FL, USA, 2018, pp. 956–963.

[191] M. Chen, J. Wan, and F. Li, "Machine-to-machine communications: Architectures, standards and applications," *KSII Trans. Internet Inf. Syst.*, vol. 6, no. 2, pp. 510–527, 2012.

[192] M. R. Palattella *et al.*, "Internet of Things in the 5G era: Enablers, architecture, and business models," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 510–527, Mar. 2016.

[193] S. Li, L. Da Xu, and S. Zhao, "5G Internet of Things: A survey," *J. Ind. Inf. Integr.*, Feb. 2018.

[194] Z. D. Purvis and A. G. Dean, "TOSSTI: Saving time and energy in TinyOS with software thread integration," in *Proc. IEEE Real-Time Embedded Technol. Appl. Symp. (RTAS)*, St. Louis, MO, USA, 2008, pp. 354–363.

[195] P. Madden *et al.*, "The emergence of networking abstractions and techniques in TinyOS," in *Proc. 1st Symp. Netw. Syst. Design Implement. (NSDI)*, vol. 4. San Francisco, CA, USA, 2004, pp. 1–14.

[196] S. K. Sarna and M. Zaveri, "EATT: Energy aware target tracking for wireless sensor networks using TinyOS," in *Proc. 3rd IEEE Int. Conf. Comput. Sci. Inf. Technol. (ICCSIT)*, vol. 1. Chengdu, China, 2010, pp. 187–191.

[197] K. Kuladinithi, O. Bergmann, T. Pötsch, M. Becker, and C. Görg, "Implementation of CoAP and its application in transport logistics," in *Proc. Extending Internet Low Power Lossy Netw. (IP+SN)*, 2011, pp. 1–7.

[198] T. A. Alghamdi, A. Lasebae, and M. Aiash, "Security analysis of the constrained application protocol in the Internet of Things," in *Proc. 2nd IEEE Int. Conf. Future Gener. Commun. Technol. (FGCT)*, London, U.K., 2013, pp. 163–168.

[199] Dunkels, "Rime-a lightweight layered communication stack for sensor networks," Eprints.Sics.Se, 2007.

[200] F. C. Neto and C. M. F. A. Ribeiro, "Dynamic change of services in wireless sensor network middleware based on semantic technologies," in *Proc. 6th Int. Conf. Auton. Auton. Syst. (ICAS)*, Cancún, Mexico, 2010, pp. 58–63.

[201] V. Kumar, G. Oikonomou, T. Tryfonas, D. Page, and I. Phillips, "Digital investigations for IPv6-based wireless sensor networks," *Digit. Invest.*, vol. 11, no. 2, pp. S66–S75, 2014.

[202] T. V. Eicken, D. E. Culler, S. C. Goldstein, and K. K. Schauser, "Active messages: A mechanism for integrated communication and computation," in *Proc. 19th Int. Symp. Comput. Archit.*, Gold Coast, QLD, Australia, May 1992, pp. 256–266.

[203] S. Khan, S. Basharat, M. S. H. Khiyal, and S. A. Khan, "Investigating energy consumption of localized and non localized ad hoc routing protocols in TinyOS," in *Proc. 10th IEEE Int. Multitopic Conf. (INMIC)*, Islamabad, Pakistan, 2006, pp. 355–358.

[204] Z. Rehena, K. Kumar, S. Roy, and N. Mukherjee, "Spin implementation in TinyOS environment using nesC," in *Proc. IEEE Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT)*, Karur, India, 2010, pp. 1–6.

[205] F. De Rango, P. Fazio, F. Veltri, and S. Marano, "Interference aware routing protocols over ad hoc UWB networks," in *Proc. 4th IEEE Int. Symp. Wireless Commun. Syst. (ISWCS)*, Trondheim, Norway, Oct. 2007, pp. 637–641.

[206] H. Petersen, C. Adjih, O. Hahm, and E. Baccelli, "Demo: IoT meets robotics—First steps, RIOT car, and perspectives," in *Proc. Int. Conf. Embedded Wireless Syst. Netw.*, Graz, Austria, 2016, pp. 269–270.

[207] H. Petersen, E. Baccelli, and W. Matthias, "Interoperable services on constrained devices in the Internet of Things," in *Proc. W3C Workshop Web Things*, 2014, pp. 1–4.

[208] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, "U-connect: A low-latency energy-efficient asynchronous neighbor discovery protocol," in *Proc. 9th ACM/IEEE Int. Conf. Inf. Process. Sensor Netw.*, Stockholm, Sweden, 2010, pp. 350–361.

[209] Q. Cao and T. Abdelzaher, "Faithful reconstruction of application behavior based on event traces in the LiteOS operating system," in *Proc. Int. Conf. Inf. Process. Sensor Netw. (IPSN)*, St. Louis, MO, USA, 2008, pp. 549–550.

[210] C. C. P. Hall, A. Carzaniga, J. Rose, and A. L. Wolf, "A content-based networking protocol For sensor networks," Dept. Comput. Sci., Univ. Colorado, Denver, CO, USA, Rep. CU-CS-979-04, 2004.

[211] T. Baumgartner. (2011). *Generic Implementation for Heterogeneous Tiny Artifacts*. [Online]. Available: https://www.Fronts.cti.gr

[212] W. P. Mccartney and N. Sridhar, "Stackless preemptive threads for TinyOS," in *Proc. 7th IEEE Int. Conf. Distrib. Comput. Sensor Syst. (DCOSS)*, Barcelona, Spain, 2011.

[213] A. Bernauer and K. Römer, "A comprehensive compiler-assisted thread abstraction for resource-constrained systems," in *Proc. IPSN*, Philadelphia, PA, USA, 2013, pp. 167–177.

[214] P. M. Pathak, "An approach to memory management in wireless sensor networks," *Int. J. Comput. Sci. Eng. Technol.*, vol. 4, no. 8, pp. 1171–1176, 2013.

[215] R. Chu, L. Gu, Y. Liu, M. Li, and X. Lu, "Versatile stack management for multitasking sensor networks," in *Proc. IEEE 30th Int. Conf. Distrib. Comput. Syst.*, Genoa, Italy, 2010, pp. 388–397.

[216] L. Machaya and S. Tembo, "A study on memory management in wireless sensor nodes during key agreement generation," *Int. J. Netw. Commun.*, vol. 6, no. 2, pp. 19–23, 2016.

[217] J. David and S. Kumar, "Investigation on secondary memory management in wireless sensor network," *Int. J. Comput. Eng. Res. Trends*, vol. 876, no. 6, pp. 387–391, 2015.

[218] H. Min, S. Yi, Y. Cho, and J. Hong, "An efficient dynamic memory allocator for sensor operating systems," in *Proc. ACM Symp. Appl. Comput. (SAC)*, 2007, p. 1159.

[219] F. Ullah, C. Science, and A. Wali, "A survey: Embedded systems supporting by different operating systems," *Int. J. Sci. Res. Sci. Eng. Technol.*, vol. 2, no. 2, pp. 664–673, 2016.

[220] M. I. N. Hong, C. H. O. Yoo-Kun, and J. I. M. Hong, "Dynamic memory allocator for sensor operating system design and analysis," *J. Inf. Sci. Eng.*, vol. 26, no. 1, pp. 1–14, 2010.

[221] F. Wu, M. Q.-H. Meng, L. Wu, Z. Zhang, and X. Chen, "The study and improvement of memory management based on SOS," in *Proc. IEEE Int. Conf. Robot. Biomimetics (ROBIO)*, Bangkok, Thailand, 2008, pp. 2040–2044.

[222] S. Das, A. Singh, S. P. Singh, and A. Kumar, "A low overhead dynamic memory management system for constrained memory embedded systems," in *Proc. 2nd Int. Conf. Comput. Sustain. Glob. Develop. (INDIACom)*, New Delhi, India, 2015, pp. 809–815.

[223] M. Vellanki, S. P. R. Kandukuri, and A. Razaque, "Node level energy efficiency protocol for Internet of Things," *J. Theor. Comput. Sci.*, vol. 3, no. 1, p. 140, 2015.

[224] K. Cengiz and T. Dag, "A review on the recent energy-efficient approaches for the Internet protocol stack," *EURASIP J. Wireless Commun. Netw.*, vol. 2015, no. 1, pp. 1–22, 2015.

[225] A. Barbato, M. Barrano, A. Capone, and N. Figiani, "Resource oriented and energy efficient routing protocol for IPv6 wireless sensor networks," in *Proc. IEEE Online Conf. Green Commun.*, Piscataway Township, NJ, USA, Oct. 2013, pp. 163–168.

[226] J. Beaudaux, A. Gallais, and T. Noël, "Heterogeneous MAC duty-cycling for energy-efficient Internet of Things deployments," *Netw. Sci.*, vol. 3, nos. 1–4, pp. 54–62, 2013.

[227] S.-H. Park, S. Cho, and J.-R. Lee, "Energy-efficient probabilistic routing algorithm for Internet of Things," *J. Appl. Math.*, vol. 2014, Apr. 2014, Art. no. 213106.

[228] S. A. Chelloug, "Energy-efficient content-based routing in Internet of Things," *J. Comput. Commun.*, vol. 3, no. 12, pp. 9–20, Dec. 2015.

[229] A. Caracaş *et al.*, "Energy-efficiency through micro-managing communication and optimizing sleep," in *Proc. 8th Annu. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw. (SECON)*, Salt Lake City, UT, USA, 2011, pp. 55–63.

[230] T. Rault, A. Bouabdallah, and Y. Challal, "Energy efficiency in wireless sensor networks: A top-down survey," *Comput. Netw.*, vol. 67, pp. 104–122, Jul. 2014.

[231] C. E. Jones, K. M. Sivalingam, P. Agrawal, and J. C. Chen, "A survey of energy efficient network protocols for wireless networks," *Wireless Netw.*, vol. 7, no. 4, pp. 343–358, 2001.

[232] M. R. Mundada, S. Kiran, S. Khobanna, R. N. Varsha, and S. A. George, "A study on energy efficient routing protocols in wireless sensor networks," *Int. J. Distrib. Parallel Syst.*, vol. 3, no. 3, pp. 311–330, May 2017.

[233] S. Rani *et al.*, "A novel scheme for an energy efficient Internet of Things based on wireless sensor networks," *Sensors*, vol. 15, no. 11, pp. 28603–28626, 2015.

[234] M. Eshaftri, A. Y. Al-Dubai, I. Romdhani, and M. B. Yassien, "A new energy efficient cluster based protocol for wireless sensor networks," in *Proc. Federated Conf. Comput. Sci. Inf. Syst.*, vol. 5. Łódź, Poland, Sep. 2015, pp. 1209–1214.

[235] E. Baccelli, O. Hahm, M. Günes, M. Wählisch, and T. C. Schmidt, "OS for the IoT—Goals, challenges, and solutions," in *Proc. Workshop Interdisciplinaire sur la Sécurité Globale (WISG)*, 2013.

[236] S. Karthik, P. Muthukrishnan, T. S. Balaji, and S. Balaji, "Comparitive study of hardwares and softwares in Internet of Things," *Middle-East J. Sci. Res.*, vol. 24, no. S1, pp. 59–63, 2016.

[237] C. Chilipirea, A. Ursache, D. O. Popa, and F. Pop, "Energy efficiency and robustness for IoT: Building a smart home security system," in *Proc. IEEE 12th Int. Conf. Intell. Comput. Commun. Process. (ICCP)*, 2016, pp. 43–48.

[238] K. Roussel, Y.-Q. Song, and O. Zendra, "RIOT OS paves the way for implementation of high-performance MAC protocols," in *Proc. 4th Int. Conf. Sensor Netw.*, Loire Valley, France, 2015, pp. 5–14.

[239] X. Liu, K. M. Hou, C. D. Vaulx, H. Zhu, and X. Liu, "Memory optimization techniques for multithreaded operating system on wireless sensor nodes," in *Proc. Int. Conf. Progr. Informat. Comput. (PIC)*, Shanghai, China, 2014, pp. 503–508.

[240] V. Gupta *et al.*, "Nano-CF: A coordination framework for macro-programming in wireless sensor networks," in *Proc. 8th Annu. IEEE Commun. Soc. Conf. Sensor Mesh Ad Hoc Commun. Netw. (SECON)*, Salt Lake City, UT, USA, 2011, pp. 467–475.

[241] A. Rowe, K. Lakshmanan, H. Zhu, and R. Rajkumar, "Rate-harmonized scheduling for saving energy," in *Proc. Real Time Syst. Symp.*, Barcelona, Spain, 2008, pp. 113–122.

[242] Q. Cao, D. Fesehaye, N. Pham, Y. Sarwar, and T. Abdelzahe, "Virtual battery: An energy reserve abstraction for embedded sensor networks," in *Proc. Real Time Syst. Symp.*, Barcelona, Spain, 2008, pp. 123–133.

[243] C. Duffy, U. Roedig, J. Herbert, and C. J. Sreenan, "Improving the energy efficiency of the MANTIS kernel," in *Proc. Eur. Conf. Wireless Sensor Netw.*, Delft, The Netherlands, 2007, pp. 261–276.

[244] K. Wang, Y. Wang, Y. Sun, S. Guo, and J. Wu, "Green industrial Internet of Things architecture: An energy-efficient perspective," *IEEE Commun. Mag.*, vol. 54, no. 12, pp. 48–54, Dec. 2016.

[245] J. Li, Y. Liu, Z. Zhang, J. Ren, and N. Zhao, "Towards green IoT networking: Performance optimization of network coding based communication and reliable storage," *IEEE Access*, vol. 5, pp. 8780–8791, 2017.

[246] J.-H. Kim, D. Park, and H.-Y. Song, "Network coding-based information sharing strategy for reducing energy consumption in IoT environments," *J. Korean Inst. Commun. Inf. Sci.*, vol. 41, no. 4, pp. 433–440, 2016.

[247] S. Wunderlich, J. A. Cabrera, F. H. P. Fitzek, and M. Reisslein, "Network coding in heterogeneous multicore IoT nodes with DAG scheduling of parallel matrix block operations," *IEEE Internet Things J.*, vol. 4, no. 4, pp. 917–933, Aug. 2017.

[248] G. Leelavathi, K. Shaila, K. Venugopal, and L. Patnaik, "Design issues on software aspects and simulation tools for wireless sensor networks," *Int. J. Netw. Security Appl.*, vol. 5, no. 2, p. 47, 2013.

[249] H. Sundani, H. Li, V. Devabhaktuni, M. Alam, and P. Bhattacharya, "Wireless sensor network simulators a survey and comparisons," *Int. J. Comput. Netw.*, vol. 2, no. 5, pp. 249–265, 2011.

[250] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proc. Conf. Local Comput. Netw. (LCN)*, Tampa, FL, USA, 2006, pp. 641–648.

[251] A. Sehgal, *Using the Contiki Cooja Simulator*, vol. 1, Comput. Sci., Jacobs Univ. Bremen, Bremen, Germany, 2013.

[252] K. Roussel, Y.-Q. Song, and O. Zendra, "Using Cooja for WSN simulations: Some new uses and limits," in *Proc. EWSN NextMote Workshop*, 2016, pp. 319–324.

[253] J. Eriksson *et al.*, "COOJA/MSPSim: Interoperability testing for wireless sensor networks," in *Proc. 2nd Int. ICST Conf. Simulat. Tools Techn. (Simutools)*, Rome, Italy, 2009, pp. 1–7.

[254] M. Prist, S. Longhi, A. Monteriù, F. Giuggioloni, and A. Freddi, "An integrated simulation environment for wireless sensor networks," in *Proc. IEEE 16th Int. Symp. World Wireless Mobile Multimedia Netw. (WoWMoM)*, 2015, pp. 1–3.

[255] N. Tsiftes *et al.*, "A framework for low-power IPv6 routing simulation, experimentation, and evaluation," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 479–480, 2010.

[256] G. Z. Papadopoulos, J. Beaudaux, A. Gallais, T. Noel, and G. Schreiner, "Adding value to WSN simulation using the IoT-LAB experimental platform," in *Proc. IEEE 9th Int. Conf. Wireless Mobile Comput. Netw. Commun. (WiMob)*, Lyon, France, 2013, pp. 485–490.

[257] B. L. Titzer, D. K. Lee, and J. Palsberg, "Avrora: Scalable sensor network simulation with precise timing," in *Proc. 4th Int. Symp. Inf. Process. Sensor Netw. (IPSN)*, vol. 2005. Boise, ID, USA, 2005, pp. 477–482.

[258] R. Want, B. N. Schilit, and S. Jenson, "Enabling the Internet of Things the IOT vision," *Computer*, vol. 48, no. 1, pp. 28–35, 2015.

[259] C. Doukas and F. Antonelli, "Developing and deploying end-to-end interoperable & discoverable IoT applications," in *Proc. IEEE Int. Conf. Commun.*, vol. 2015. London, U.K., Sep. 2015, pp. 673–678.

[260] A. A. Abed, "Internet of Things (IoT): Architecture and design," in *Proc. Al Sadeq Int. Conf. Multidisciplinary IT Commun. Sci. Appl. (AIC-MITCSA)*, Baghdad, Iraq, 2016, pp. 1–3.

[261] A. Saeed, M. Ammar, K. A. Harras, and E. Zegura, "Vision: The case for symbiosis in the Internet of Things," in *Proc. 6th Int. Workshop Mobile Cloud Comput. Services*, Paris, France, 2015, pp. 23–27.

[262] G. Chong, L. Zhihao, and Y. Yifeng, "The research and implement of smart home system based on Internet of Things," in *Proc. Int. Conf. Electron. Commun. Control (ICECC)*, Ningbo, China, 2011, pp. 2944–2947.

[263] Q.-B. Sun, J. Liu, S. Li, C.-X. Fan, and J.-J. Sun, "Internet of Things: Summarize on concepts, architecture and key technology problem," *J. Beijing Univ. Posts Telecommun.*, vol. 3, no. 3, pp. 1–9, 2010.

[264] X. Li *et al.*, "Smart community: An Internet of Things application," *IEEE Commun. Mag.*, vol. 49, no. 11, pp. 68–75, Nov. 2011.

[265] M. Wang, G. Zhang, C. Zhang, J. Zhang, and C. Li, "An IoT-based appliance control system for smart homes," in *Proc. 4th Int. Conf. Intell. Control Inf. Process. (ICICIP)*, Beijing, China, 2013, pp. 744–747.

[266] J. Wei, "How wearables intersect with the cloud and the Internet of Things: Considerations for the developers of wearables," *IEEE Consum. Electron. Mag.*, vol. 3, no. 3, pp. 53–56, Jul. 2014.

[267] A. G. Ferreira *et al.*, "A smart wearable system for sudden infant death syndrome monitoring," in *Proc. IEEE Int. Conf. Ind. Technol.*, vol. 2016. Taipei, Taiwan, May 2016, pp. 1920–1925.

[268] S. Hiremath, G. Yang, and K. Mankodiya, "Wearable Internet of Things: Concept, architectural components and promises for person-centered healthcare," in *Proc. 4th Int. Conf. Wireless Mobile Commun. Healthcare Transforming Healthcare Through Innov. Mobile Wireless Technol. (MOBIHEALTH)*, Athens, Greece, 2015, pp. 304–307.

[269] W. Zhou and S. Piramuthu, "Security/privacy of wearable fitness tracking IoT devices," in *Proc. Iberian Conf. Inf. Syst. Technol. (CISTI)*, Barcelona, Spain, Sep. 2014, pp. 1–5.

[270] S. W. Yoon, B. Petrov, and K. Liu, "Advanced wafer level technology: Enabling innovations in mobile, IoT and wearable electronics," in *Proc. Electron. Packag. Technol. Conf. (EPTC)*, vol. 2016. Singapore, Feb. 2016, pp. 1–5.

[271] D. C. Bogatinoska, R. Malekian, J. Trengoska, and W. A. Nyako, "Advanced sensing and Internet of Things in smart cities," in *Proc. 39th Int. Conv. Inf. Commun. Technol. Electron. Microelectron. (MIPRO)*, 2016, pp. 632–637.

[272] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, "An information framework for creating a smart city through Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 112–121, Apr. 2014.

[273] M. S. Ali, E. Hossain, and D. I. Kim, "LTE/LTE-A random access for massive machine-type communications in smart cities," *IEEE Commun. Mag.*, vol. 55, no. 1, pp. 76–83, Jan. 2017.

[274] R. R. Tyagi, F. Aurzada, K.-D. Lee, and M. Reisslein, "Connection establishment in LTE-A networks: Justification of Poisson process modeling," *IEEE Syst. J.*, vol. 11, no. 4, pp. 2383–2394, Dec. 2017.

[275] K. E. Skouby and P. Lynggaard, "Smart home and smart city solutions enabled by 5G, IoT, AAI and CoT services," in *Proc. Int. Conf. Contemp. Comput. Informat. (IC3I)*, Mysore, India, 2014, pp. 874–878.

[276] M. Centenaro, L. Vangelista, A. Zanella, and M. Zorzi, "Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios," *IEEE Wireless Commun.*, vol. 23, no. 5, pp. 60–67, Oct. 2016.

[277] R. Petrolo, A. Roukounaki, V. Loscr, N. Mitton, and J. Soldatos, "Connecting physical things to a SmartCity-OS," in *Proc. IEEE Int. Conf. Sens. Commun. Netw. (SECON Workshops)*, London, U.K., 2016, pp. 1–6.

[278] B. Kantarci and H. T. Mouftah, "Trustworthy sensing for public safety in cloud-centric Internet of Things," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 360–368, Aug. 2014.

[279] C. Tao and L. Xiang, "Municipal solid waste recycle management information platform based on Internet of Things technology," in *Proc. Int. Conf. Multimedia Inf. Netw. Security*, Nanjing, China, 2010, pp. 729–732.

[280] M. Yun and B. Yuxin, "Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid," in *Proc. Int. Conf. Adv. Energy Eng. (ICAEE)*, Beijing, China, 2010, pp. 69–72.

[281] N. Bui, A. P. Castellani, P. Casari, and M. Zorzi, "The Internet of energy: A Web-enabled smart grid system," *IEEE Netw.*, vol. 26, no. 4, pp. 39–45, Jul./Aug. 2012.

[282] W. H. Chin, Z. Fan, and R. Haines, "Emerging technologies and research challenges for 5G wireless networks," *IEEE Wireless Commun.*, vol. 21, no. 2, pp. 106–112, Apr. 2014.

[283] H. P. Breivold and K. Sandstroem, "Internet of Things for industrial automation–Challenges and technical solutions," in *Proc. IEEE Int. Conf. Data Sci. Data Intensive Syst.*, Sydney, NSW, Australia, 2015, pp. 532–539.

[284] M. F. Muñoz-Doyague and M. Nieto, "Individual creativity performance and the quality of interpersonal relationships," *Ind. Manag. Data Syst.*, vol. 112, no. 1, pp. 125–145, 2012.

[285] S. Sahabiswas *et al.*, "Drunken driving detection and prevention models using Internet of Things," in *Proc. IEEE 7th Annu. Inf. Technol. Electron. Mobile Commun. Conf. (IEMCON)*, Vancouver, BC, Canada, 2016, pp. 1–4.

[286] T. T. Thakur, A. K. Naik, S. Vatari, and M. Gogate, "Real time traffic management using Internet of Things," in *Proc. Int. Conf. Commun. Signal Process. (ICCSP)*, 2016, pp. 1950–1953.

[287] S. Chen, H. Xu, D. Liu, B. Hu, and H. Wang, "A vision of IoT: Applications, challenges, and opportunities with China perspective," *IEEE Internet Things J.*, vol. 1, no. 4, pp. 349–359, Aug. 2014.

[288] M. Zhang, T. Yu, and G. F. Zhai, "Smart transport system based on 'the Internet of Things'", *Appl. Mech. Mater.*, vols. 48–49, pp. 1073–1076, Feb. 2011.

[289] S. Amendola, R. Lodato, S. Manzari, C. Occhiuzzi, and G. Marrocco, "RFID technology for IoT-based personal healthcare in smart spaces," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 144–152, Apr. 2014.

[290] L. Catarinucci *et al.*, "An IoT-aware architecture for smart healthcare systems," *IEEE Internet Things J.*, vol. 2, no. 6, pp. 515–526, Dec. 2015.

[291] H. S. Ahmed and A. A. Ali, "Smart intensive care unit desgin based on wireless sensor network and Internet of Things," in *Proc. Al Sadeq Int. Conf. Multidisciplinary IT Commun. Sci. Appl. (AIC-MITCSA)*, Baghdad, Iraq, 2016, pp. 1–6.

[292] C. Bauer, P. Dohmen, and C. Strauss, "Interactive digital signage—An innovative service and its future strategies," in *Proc. Int. Conf. Emerg. Intell. Data Web Technol. (EIDWT)*, Tirana, Albania, 2011, pp. 137–142.

[293] R. Porkodi and V. Bhuvaneswari, "The Internet of Things (IOT) applications and communication enabling technology standards: An overview," in *Proc. Int. Conf. Intell. Comput. Appl. (ICICA)*, 2014, pp. 324–329.

[294] D. McFarlane and Y. Sheffi, "The impact of automatic identification on supply chain operations," *Int. J. Logistics Manag.*, vol. 14, no. 1, pp. 1–17, 2003.

[295] M. Kärkkäinen, "Increasing efficiency in the supply chain for short shelf life goods using RFID tagging," *Int. J. Retail Distrib. Manag.*, vol. 31, no. 10, pp. 529–536, 2003.

[296] A. Na and W. Isaac, "Developing a human-centric agricultural model in the IoT environment," in *Proc. Int. Conf. Internet Things Appl. (IOTA)*, Pune, India, 2016, pp. 292–297.

[297] R. Nukala *et al.*, "Internet of Things: A review from 'farm to fork,'" in *Proc. 27th Irish Signals Syst. Conf. (ISSC)*, 2016, pp. 1–6.

[298] M. K. Gayatri, J. Jayasakthi, and G. S. A. Mala, "Providing smart agricultural solutions to farmers for better yielding using IoT," in *Proc. IEEE Int. Conf. Technol. Innov. ICT Agricult. Rural Develop. (TIAR)*, 2015, pp. 40–43.

[299] V. V. H. Ram, H. Vishal, S. Dhanalakshmi, and P. M. Vidya, "Regulation of water in agriculture field using Internet of Things," in *Proc. IEEE Int. Conf. Technol. Innov. ICT Agricult. Rural Develop. (TIAR)*, Chennai, India, 2015, pp. 112–115.

[300] J. Wu, S. Guo, J. Li, and D. Zeng, "Big data meet green challenges: Greening big data," *IEEE Syst. J.*, vol. 10, no. 3, pp. 873–887, Sep. 2016.

[301] R. Angeles, "Emerging technologies: Supply-chain applications and implementation issues," *Inf. Syst. Manag.*, vol. 22, pp. 51–65, Sep. 2005.

[302] Y.-K. Chen, "Challenges and opportunities of Internet of Things," in *Proc. 17th Asia South Pac. Design Autom. Conf.*, Sydney, NSW, Australia, 2012, pp. 383–388.

[303] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad, *Proposed Security Model and Threat Taxonomy for the Internet of Things (IoT)* (Communications in Computer and Information Science), vol. 89. Heidelberg, Germany: Springer, 2010, pp. 420–429.

[304] T. Polk and S. Turner, "Security challenges for the Internet of Things," in *Proc. Workshop Interconnect. Smart Objects Internet*, 2011.

[305] M. A. Chaqfeh and N. Mohamed, "Challenges in middleware solutions for the Internet of Things," in *Proc. Int. Conf. Collaboration Technol. Syst. (CTS)*, Denver, CO, USA, 2012, pp. 21–26.

[306] R. Atat *et al.*, "Enabling cyber-physical communication in 5G cellular networks: Challenges, spatial spectrum sensing, and cyber-security," *IET Cyber Phys. Syst. Theory Appl.*, vol. 2, no. 1, pp. 49–54, Apr. 2017.

[307] B. Han, O. Holland, V. Sciancalepore, M. Dohler, and H. Schotten, "D2D-based grouped random access to mitigate mobile access congestion in 5G sensor networks," *Arxiv.org*, 2018.

[308] M. U. Farooq, M. Waseem, A. Khairi, and S. Mazhar, "A critical analysis on the security concerns of Internet of Things (IoT)," *Int. J. Comput. Appl.*, vol. 111, no. 7, pp. 1–6, 2015.

[309] A. Ukil, *Embedded Security for the Industrial Internet of Things*, M2 Presswire, Coventry, U.K., 2015.

[310] D. Singh, G. Tripathi, and A. J. Jara, "A survey of Internet-of-Things: Future vision, architecture, challenges and services," in *Proc. IEEE World Forum Internet Things (WF-IoT)*, Seoul, South Korea, 2014, pp. 287–292.

[311] T. Xu, J. B. Wendt, and M. Potkonjak, "Security of IoT systems: Design challenges and opportunities," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design*, San Jose, CA, USA, 2014, pp. 417–423.

[312] V. Gazis *et al.*, "Short paper: IoT: Challenges, projects, architectures," in *Proc. 18th Int. Conf. Intell. Next Gener. Netw. (ICIN)*, Paris, France, 2015, pp. 145–147.

[313] C.-W. Tsai, C.-F. Lai, and A. V. Vasilakos, "Future Internet of Things: Open issues and challenges," *Wireless Netw.*, vol. 20, no. 8, pp. 2201–2217, 2014.

[314] I. Lee and K. Lee, "The Internet of Things (IoT): Applications, investments, and challenges for enterprises," *Bus. Horizons*, vol. 58, no. 4, pp. 431–440, 2015.

[315] I. Alqassem, "Privacy and security requirements framework for the Internet of Things (IoT)," in *Proc. 36th Int. Conf. Softw. Eng. ICSE Companion*, 2014, pp. 739–741.

[316] W. Zhang *et al.*, "OpenWSN, an layer first component architecture and package for sensor networks," in *Proc. IET Conf. Wireless Mobile Sensor Netw. (CCWMSN)*, Shanghai, China, 2007, pp. 1047–1050.

[317] V. J. P. Amorim, S. Delabrida, and R. A. R. Oliveira, "A constraint-driven assessment of operating systems for wearable devices," in *Proc. VI Brazil. Symp. Comput. Syst. Eng. (SBESC)*, 2016, pp. 150–155.

Farhana Javed received the B.S. degree in computer science from the COMSATS Institute of Information Technology, Wah Cantt, Pakistan. She is currently pursuing the M.S. degree from the Huazhong University of Science and Technology, Wuhan, China. Her research interests include wireless sensor networks, Internet of Things, networking, protocol design, optimization, and performance evaluation of body area nano-network.

Muhamamd Khalil Afzal (SM'16) received the B.S. and M.S. degrees in computer science from the COMSATS Institute of Information Technology, Wah Cantt, Pakistan, in 2004 and 2007, respectively, and the Ph.D. degree from the Department of Information and Communication Engineering, Yeungnam University, South Korea, in 2014. He has served as a Lecturer with Bahauddin Zakariya University, Multan, Pakistan, from 2008 to 2009 and King Khalid University Abha, Saudi Arabia, from 2009 to 2011. He is currently an Assistant Professor with the Department of Computer Science, COMSATS. His research interests include wireless sensor networks, ad hoc networks, smart cities, 5G, and IoT. He is serving as a Guest Editor for *Future Generation Computer Systems* (Elsevier), the IEEE ACCESS, the *Journal of Ambient Intelligence and Humanized Computing* (Springer), and the *IEEE Communication Magazine* and a Reviewer for the IEEE ACCESS, *Computers and Electrical Engineering* (Elsevier), the *Journal of Network and Computer Applications* (Elsevier), *Future Generation Computer Systems*, and the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY.

Muhammad Sharif is an Associate Professor with COMSATS, Wah Cantt, Pakistan. His area of specialization is artificial intelligence and image processing. He is into teaching field since 1995. He has over 140 research publications in IF, SCI, and ISI journals and national and international conferences. He has so far supervised 40 M.S. (CS) thesis. He is currently supervising five Ph.D. (CS) students and co-supervisor of ten others. More than 300 undergraduate students have successfully completed their project work under his supervision. His research interests are image processing, medical imaging, computer vision, and algorithms design and analysis.

Byung-Seo Kim (M'02–SM'17) received the B.S. degree in electrical engineering from Inha University, Incheon, South Korea, in 1998 and the M.S. and Ph.D. degrees in electrical and computer engineering from the University of Florida in 2001 and 2004, respectively. His Ph.D. study was supervised by Dr. Y. Fang. From 1997 to 1999, he was with Motorola Korea Ltd., PaJu, South Korea, as a Computer Integrated Manufacturing Engineer in Advanced Technology Research and Development. From 2005 to 2007, he was with Motorola Inc., Schaumburg, IL, USA, as a Senior Software Engineer in Networks and Enterprises. His research focuses in Motorola Inc. were designing protocol and network architecture of wireless broadband mission critical communications. He is currently an Associate Professor with the Department of Software and Communications Engineering, Hongik University, South Korea. He was a Chairman of the Department from 2012 to 2014. His work has appeared in around 161 publications and 22 patents. His research interests include the design and development of efficient wireless/wired networks, including link-adaptable/cross-layer-based protocols, multiprotocol structures, wireless CCNs/NDNs, mobile edge computing, physical layer design for broadband PLC, and resource allocation algorithms for wireless networks. He served as the General Chair for General Chair of 3rd IWWCN 2017, and the TPC Member for the IEEE VTC 2014-Spring and the EAI FUTURE 2016, and ICGHIC 2016–2018 conferences. He served as a Guest Editor of special issues of the *International Journal of Distributed Sensor Networks*, the IEEE ACCESS, and the *Journal of the Institute of Electronics and Information Engineers*. He was also served as the member of Sejong-city Construction Review Committee and Ansan-city Design Advisory Board. He is an Associative Editor of the IEEE ACCESS.