A Review on IoT Operating Systems

Amal Antony
Student
Department of Computer Science, Naipunnya Institute of Management and Information Technology, Pongam, Thrissur

Sarika S., PhD
Assistant Professor
Department of Computer Science, Naipunnya Institute of Management and Information Technology, Pongam, Thrissur

ABSTRACT
An IoT development board is a small-form-factor system, complete with microprocessor(s), memory, input/output functions providing the user with all the features of a functional computer. The MCU based smaller variants house limited hardware resources and do not demand an operating system. But the more powerful single board computers require an operating system to efficiently manage its resources and control the hardware. The choice of operating system depends on the microcontroller architecture, on-board memory, software stack used, real-time computing requirements, implementation environment and cost of the system. Operating systems for IoT applications require additional functionalities like network support, power usage monitoring, secondary storage management, multithreading and so on. This paper intends to survey the different IoT operating systems available in the market and studies the various considerations on the selection of OS for IoT development boards.

General Terms
Internet of Things, Operating System, Single Board Computer, Embedded Systems

Keywords
IoT, operating system, embedded system, smart devices, embedded Linux, open-source, development board

1. INTRODUCTION
The "Internet of Things" abbreviated as IoT is a comprehensive model including all kinds of computing devices, that are connected to the Internet. These devices are otherwise called “things” or “smart objects” [1]. In theory, a device can be attached to almost any real world object like vehicles, home appliances, industrial, mechanical or electrical machines and even a person to let the object communicate to the Internet. IoT finds applications in buildings and home automation, smart cities, smart industry or manufacturing, wearables, healthcare devices and automotive. The recent developments in IoT are focused on edge computing and on-device AI capabilities [2]. In order to comply with all of these requirements and applications, an operating system is essential for every IoT device. Having an operating system simplifies the developers’ job and contributes to standardization. Continuous development by industry practitioners and researchers is very essential in this domain so as to provide support for changing hardware configurations and communication standards. An ideal IoT OS has to support different hardware architectures, boards and devices.

There are a number of IoT OSs like Contiki-OS, RIOT and Zephyr to name a few. IoT devices run on low capacity microcontrollers. So, applications running on this platform has to be lean and energy-efficient. It is a prerequisite for an IoT OS to have the essential Transmission Control Protocol/Internet Protocol (TCP/IP) capabilities for seamless integration with the global internet. Apart from providing support for TCP and UDP, modern IoT operating systems are trying to accommodate new standards like Internet Protocol version 6 (IPv6) over Low-Power Wireless Personal Area Networks (6LoWPAN), Routing over Low Power and Lossy Networks (ROLL), Bluetooth Low Energy (BLE) and Bluetooth Meshes.

This paper aims to establish the need for an IoT operating system and provide a framework for choosing an OS. The study is done from a hobbyists or builder’s perspective, that the complexities of academic research is overlooked at some parts. Only those IoT OSs that are available in the market or maintained as open source projects have been considered for the purpose of the study. Proprietary and special purpose software products do not come under the purview of this paper.

The rest of the paper is organized as follows. Section 2 introduces the common IoT hardware platforms. Section 3 delves into IoT operating systems in detail, with subsections discussing the functions of an IoT OS, parameters in the selection of an OS and popular IoT operating systems. Section 4 discusses the usage and adoption of IoT OSs. Section 5 includes the conclusion and insights.

2. IOT HARDWARE PLATFORMS
IoT hardware encompasses all devices capable of connecting to the Internet. IoT devices can otherwise mean ‘smart objects. These ‘things’ or ‘objects’ are responsible for providing useful information during their transactions on a network. On one hand, there are specialized wearable gadgets like the Google Glass or Fitbit, which are very compact IoT devices. The other category of IoT hardware includes general purpose development boards or Single Board Computers (SBCs) [3]. These development boards allow engineers to create prototypes of IoT solutions and test them, before it can be mass produced. The peculiarity about these development boards is that they are very flexible and can be used to create applications for any domain. This allows open source building and collaboration between engineers. IoT boards can range from an 8-bit MCU to a 32-bit or 64-bit fully functional computer. These boards can have various features like USB interfacing, video output, audio jacks, networking, GPIO pins and wireless communication chips. The IoT environment consists of protocols, network designs, and service architectures that binds together millions of IoT devices to exchange data. It is essential to learn about the hardware platforms and development boards used for IoT prototyping, so that the synergy between IoT hardware platforms and IoT operating systems can be better understood. This section details the IoT development boards in common use and their

features. A summary of IoT hardware platforms has been included as a table (Table 1).

2.1 Arduino

Arduino is undoubtedly the most favourite development board among developers and hobbyists. There is an active community working on this platform. Arduino has a broad range of boards, from simple 8-bit microcontroller boards to products for wearables, IoT items, three-dimensional (3-D) printing, and much more. The most popular model of Arduino, the Arduino Uno (Figure 1) is based on ATmega328P microcontroller. An Arduino board is programmed with the help of an IDE, using a type B USB cable. The company also offers boards like Arduino Yun, with on-board Wi-Fi (IEEE 802.11 b/g/n) and Ethernet (IEEE 802.3 10/100Mb/s) [3]. But, Arduino is known around the world for its low-cost 8-bit and 16-bit models. Low end boards like these does not require an OS. Code for Arduino follows the structure and conventions of C language. Support for peripherals, sensors and actuators is provided through header files. Once the code is saved on the board, it is executed in an endless loop [4].

Figure 1 Arduino Uno

2.2 Raspberry Pi

Raspberry Pi (Figure 2) is a family of single-board computers that comes with great processing power in a small package and provides the functionalities of a desktop computer. Raspberry Pi's development began in 2006 it was finally released on 19 February 2012 as two models: Model A and Model B. After the sale of 3 million units by May 2014, Model B+ was announced in July 2014. Raspberry Pi or RPi, as it is known in the hobbyist circles, can support an operating system and is seen as a low-cost replacement for desktop PCs. The operating system is installed on an SD card and provides you the familiar interface that you expect from a Linux or Windows computer [5]. Raspberry Pi supports a number of operating systems, including Raspbian Linux, Ubuntu Mate, and Windows 10 IoT Core. As Raspberry Pi can maintain a complete operating system, it has support for languages like C/C++, Python, and JavaScript. The latest iteration of the board, the Raspberry Pi 4 features a Quad core 64-bit ARM-Cortex A72 running at 1.5GHz.

Figure 2 Raspberry Pi Model B

Table 1. Comparison of IoT hardware platforms

Brand	Model	CPU	RAM
Arduino	17+ models	Atmel AVR, ARM Cortex-M0+,ARM Cortex-M3, Intel Quark	16KB – 64MB
Raspberry Pi	A,A+,B,B+,3,4,Zero	ARM Cortex-A7, ARM Cortex A-53, ARM Cortex-A72	256 MB - 4 GB
Nvidia	Jetson Nano, Jetson Xavier NX, Jetson AGX Xavier, Jetson TX2	NVIDIA Carmel ARM v8.2,Denver 2/ ARM A57 complex	4 GB-16 GB
BeagleBoard	BeagleBone Black	ARM Cortex-A8	512 MB

2.3 BeagleBone Black

BeagleBone Black is a development board from the family of BeagleBoard platforms (Figure 3). It is powered by a TI Sitara AM335x ARM Cortex-A8 processor running at 1 GHz, with 4 GB of on-board flash memory, 512 MB of DDR3L DRAM, and a 3-D graphics accelerator. It has 46-pin headers, an Ethernet port, and several other means to enable communication. It supports the Debian, Android, and Ubuntu operating systems. BeagleBone has been developed as an open-source platform and all of its datasheets are available in BeagleBone community's website [3]. The BeagleBoard, especially the BeagleBone Black version, is easy and inexpensive to set up and use. It consumes low power and thus needs no additional cooling or heat sinks. BeagleBoard also has products such as BeagleBone Blue, PocketBeagle, BeagleBone AI, BeagleBoard X15and BeagleBoard XM.

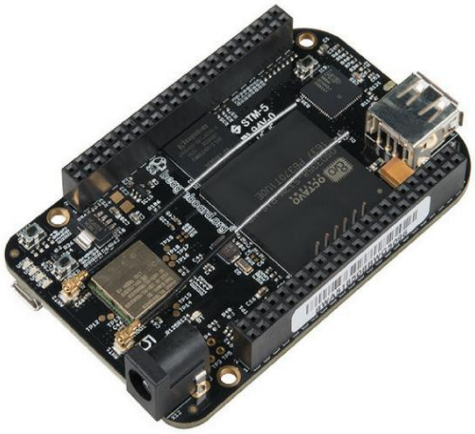

Figure 3 BeagleBone Black

2.4 NVIDIA Jetson

Jetson is designed as a powerful computer that lets the user run artificial intelligence and machine learning applications, while taking up only very little power and space [3]. Products in the Jetson portfolio include Nano (Figure 4), Xavier NX, AGX Xavier, and TX2. Among these products, Jetson AGX Xavier is the first computer designed specifically for autonomous machines. All Jetson boards come bundled with software libraries for deep learning, computer vision, GPU computing, multimedia processing, and much more. Jetson relies on the Linux environment and provides better performance than other development boards. The availability of proprietary AI and ML tools from NVIDIA reduces the work for developers. One of the biggest advantage that Jetson offers is the GPU-accelerated parallel processing. This makes Jetson an ideal choice for developers looking to implement machine learning projects. Jetson hardware is ideal for training and inference phase of deep learning problems. Jetson is commonly deployed as Network Video Recorders (NVRs), smart home robots, and intelligent gateways [6].

Figure 4 NVIDIA Jetson Nano

3. OPERATING SYSTEMS FOR IOT DEVICES

3.1 Definition

An IoT Operating System is a relatively new term and a loosely defined one. There is a lack of literature on the taxonomy and development of IoT operating systems. An IoT Operating System is a piece of software which provides for a channel of interaction between the user and IoT device and manages all hardware and software resources. It is no different from a regular operating system like Unix or Windows. The difference lies in the hardware architecture of the host system and the resource constraints. An operating system specifically designed for IoT applications which can run under the minimal resources available in an IoT device, can be termed as an IoT OS. It can be said that IoT OS is an extension of embedded system OS. An IoT OS ensures connectivity between IoT applications and embedded systems [7]. Even though IoT OS is an evolution of embedded OS, IoT brings in additional set of constraints and requirements that need to be addressed. Embedded operating systems have been upgraded or augmented to incorporate IoT-specific features. Popular IoT OSs include TinyOS, Contiki, mbedOS, Ubuntu Core, Yocto, Windows 10 for IoT and so on [8], [9].

3.2 Functions of an IoT Operating System

Technical systems require an operating system as it is the major interface with which the user can interact with the computer and manage how programs functions within the computer system [10]. It takes care of the important processes behind the scenes – managing the hardware resources, providing a user interface and executing and rendering services to application programs [11]. Unlike desktop operating systems or general purpose OS, IoT OS is designed to work with the limited resources and provide capabilities like rapid development tools, standardization, easy maintenance, support for various hardware platforms, portability of application programs and seamless integration with global internet [12], [13]. The functions performed by an IoT OS are outlined below:

3.2.1 Provides a Hardware Abstraction Layer(HAL)

A HAL can be defined as all the software that is directly dependent on the underlying hardware [14]. On more elaborate terms, hardware abstraction layer (HAL) can be defined as a layer of programming or code to allow general communication between the software and hardware components of a system. This reduces the work of application developer. HAL bridges the gap between hardware and software. Otherwise, developers will have to hard-code drivers, kernels, or APIs for each hardware device. This would be a tedious task considering the diversity of IoT hardware platforms. Hardware abstraction provided by IoT operating systems gives developers access to all OS controlled devices like Bluetooth, camera, video output, audio, sensors and storage directly [15].

3.2.2 Power Management

Integrated power management techniques have been implemented on SoCs by several manufacturers. The microcontroller has the ability to enter a sleep state or standby mode to save power. The microcontroller (MCU) stops performing computations in sleep state; But, it will retain any current data, and all peripheral functions will be halted [16]. An OS can efficiently manage these sleep states and monitor the power usage of attached peripherals or sensors. IoT devices which run continuously for infinitely long periods of time can benefit from this power saving technique.

3.2.3 Concurrency Management

IoT devices today support multi-core or multi-processor setups. This makes concurrent computing an important element of embedded computing as well. The traditional methods used for parallel programming used in regular operating systems are not suitable for IoT systems with time

and resource constraints and raises the possibility of dead-locks. So, algorithms which prioritize these requirements have to be implemented for IoT OS [17]. In the case of peripheral control or reading data from sensors, concurrent execution is not as important. The usage of event-driven asynchronous execution or collective IO is a viable solution. But when it comes to data and signal processing systems, the OS has to manage the parallelization of processes.

3.2.4 User Interface

As mentioned before, OS provides an interface between the user and the device. Some Linux based distributions available for IoT initially boots into the terminal, from where you can invoke a GUI. As in the NOOBS operating system for Raspberry Pi, the GUI is started by typing 'startx' in the terminal [5]. Most IoT OS vendors go with a graphical interface considering the large number of hobbyists or amateurs as users.

The input and output operations can also be discussed under this head. Inputs to an IoT device maybe through USB interface or GPIO pins. Most IoT development boards provides a full-fledged video output. With additional hardware, IoT boards can also support touch screen input, the example of which can be seen in a POS system or ATM.

3.2.5 Memory Management:

An IoT OS has the responsibility of managing the primary memory of the device. The memory management function keeps track of the current status in every memory location, whether it's allocated or free. It measures how memory is allocated over processes, deciding which gets memory, when they receive it, and how much they are free [18].

3.2.6 Process Scheduling

Regarding an IoT OS, processes Scheduling simply refers to managing the processes currently in the system memory. In very simple terms, it decides the order of executing things [19]. Scheduling prioritizes processes, loads them into the ready queue of the system and then send for the execution. Short, medium, and long-term scheduling can be implemented by the operating system [18].

3.2.7 Shared Libraries

The operating system provides several utility libraries for the developer, which is available to all supported development platforms and programming languages. These dynamically linked libraries can be used by any number of application programs without making copies in the main memory. A lot of memory can be saved this way and is ideally suited for the resource constrained environment of IoT. Developers can make use of these shared libraries and reduce the computing overhead. The functionalities like logging utility, TCP/IP, cryptography, time synchronization are provided by shared libraries [19]. Another example would be the SELinux library providing security in Linux systems [20].

3.2.8 Hardware Virtualization

The primary motive behind IoT device virtualization is to different IoT devices and service functions with several applications. Virtualization enables the limited hardware resources to be shared among multiple users [19]. It ensures efficient usage of the resources available and restricts access to particular group of users. In the wider sense, virtualization also includes dockers and containers, creating secure application environments which can be deployed over cloud or local network [21].

3.3 Parameters for the Choice of an IoT Operating System

The following parameters must be considered before making the selection of an IoT operating system:

3.3.1 Reliability and Stability

An OS installed on IoT devices must not crash unexpectedly. IoT devices are supposed to run without powering down, for a very long time. If the OS crashes or shows glitches, debugging the system software after deployment is a difficult job to carry out. An OS which has support from vendor or an active user community must be selected.

3.3.2 Footprint

IoT devices are bound to operate with limited resources. The OS is expected to have low memory, power and processing requirements [12]. The scheduling and process management algorithms must also suit the IoT paradigm.

3.3.3 Scalability

Scalability here refers to the ability to extend the OS to operate on the two types of IoT devices – nodes and gateways [8]. This way, system architects and administrators will only have to deal with a single OS, which runs on all kinds of devices. Scalability can also mean the ability of the OS to improve over time through updates [10]. A scalable OS, which is able to run on a variety of 8-bit,16-bit or 32-bit microcontrollers, will be easier to deploy.

3.3.4 Portability

Portability means that a system developed in one environment should execute in another environment without the need for rewriting the code [22]. This quality allows developers to switch between IoT hardware platforms without the need to alter the OS. Portability and adaptability is an important feature of embedded system. It is also recommended to have some standard interface (e.g., POSIX), for good portability of applications, for minimizing maintenance, as well as to provide the ability to easily connect to other devices on the Internet [23].

3.3.5 Modularity

Apart from the kernel, every functionality can be designed as an add-on to the operating system so that a minimal version of the OS can be run if the situations demand it [8]. A modular architecture allows to replace or add kernel components dynamically at run time. In a modular kernel, components providing similar functionality will be stored in files called modules and can be loaded when the system needs that functionality [24]. An IoT device will require a modular operating system that separates the core kernel from middleware, protocols, and applications.

3.3.6 Hardware agnostic operation

There are several IoT hardware platforms available in the market. Keeping this in mind, it is important that the OS supports different hardware platforms – leading to standardization and ease in deployment [12].

3.3.7 Network Connectivity & Protocol Support

The Internet of Things is a model of "connected" devices. So, the OS must support different connectivity and networking protocols, such as Ethernet, Wi-Fi, BLE, IEEE 802.15.4, TCP/IP, etc. [1]. An IoT OS will allow the developers to select the specific protocol stacks required for the application, thus saving memory on the device and reducing the costs. It

can help upgrade existing devices to new connectivity options without altering the core code of the OS [23].

3.3.8 Security

The operating system must provide the first layer of security for the IoT system. As the device is continuously communicating with the Internet, it is susceptible to threats and attacks. OS can have add-ons that bring security to the device by way of encryption, SSL/TLS certification management, user authentication routines, VPN and firewall [12].

3.3.9 Eco-system & Application Development

The suitability of the OS for application development and debugging can make a big impact on the speed of development and time-to-market. If the OS is developer friendly, deployment of IoT applications can be done more efficiently. In addition to the basic C environment, the use of other programming languages and libraries is highly desirable, for example python, C ++ and STL, but that highly depends on the development toolchain adopted by the developer or organization [23].

3.4 Popular IoT Operating Systems

A partial list of popular IoT operating systems is presented here. An overview of all the operating systems discussed in this section can be seen in Table 2.

3.4.1 Mbed OS

MbedOS is the Real Time Operating System (RTOS) developed by ARM delivered as an open source product with an Apache 2.0 license. Mbed OS is designed for Cortex-M microcontrollers, and incorporates a tiny RTOS based on CMSIS-RTOS RTX, TLS protocol, networking standards and common device drivers in a modular architecture [1]. It is specifically designed for 32-bit ARM architecture. MbedOS supports features such as multithreading, 6LoWPAN, BLE, Wi-Fi, sub-GHz, Near Field Communication (NFC), Radio-Frequency Identification (RFID) and Long Range Low-Power Wide Area Network(LoRaLPWAN) [13]. Minimal system requirements and support for different development boards make it a highly preferred IoT OS.

3.4.2 Contiki

Among the IoT research community, Contiki has greater acceptance. The low memory requirements make Contiki well suited for low power devices. It is written in C language. Contiki offers multithreading through protothread and uses the cooperative or preemptive scheduling for the processes [13]. Contiki provides support for multiple network stacks with a comprehensive set of features like IPv6, 6LoWPAN, RPL and CoAP. It can run on IoT platforms like wismote, sky and z1. Contiki and its code simulator Cooja has been used as a development tool in several wireless sensor projects [1].

3.4.3 RIOT OS

RIOT is an open source IoT operating system. RIOT was initially developed as part of a research project by FU Berlin, INRIA, and HAW Hamburg. It is based on the microkernel named FireKernel, that was targeting wireless sensor networks. RIOT is designed to be energy efficient and modular with very low memory requirements [1]. RIOT implements modular design and uniform API access for independent hardware abstraction. RIOT supports C and C++ programming languages. It also provides multithreading with tickless, pre-emptive and priority based scheduler. RIOT also offers an emulator called Native which acts as a hardware virtualizer, helping in application development without actually having a development board [13]. RIOT has support for hardware architectures such as AVR, ARM7, Cortex-M0 - M0+ -M3 -M4 -M7, Cortex-M23, ESP8266, ESP32, MIPS32, MSP430, PIC32, RISC-V and x86 [25]

3.4.4 Apache Mynewt

Mynewt is an open source OS with Apache License 2.0, developed by the Apache Software Foundation [1]. The OS features a flexible and powerful Bluetooth Low Energy stack (BLE 5) implementation called NimBLE which provides the option to choose HOST only or CONTROLLER only or FULL stack. Mynewt facilitates cross-platform migration as it supports a great number of hardware platforms. It is designed to be hardware agnostic and can work with Cortex M0-M4 micro controllers, MIPS and RISC-V. It is designed as a pre-emptive, multi-tasking real time operating system kernel. Mynewt's Hardware Abstraction Layer (HAL) abstracts the MCU's peripheral functions, allowing developers to easily write cross-platform code [26]. All these features make Mynewt suitable for IoT boards with low computing power and memory resources.

3.4.5 Zephyr

Zephyr is an open source OS maintained by the Linux Foundation. Zephyr works on two OS design philosophies- a microkernel for less constrained IoT devices and a nanokernel for constrained devices. It supports multithreading with cooperative, priority-based, Earliest Deadline First (EDF), non-preemptive and preemptive scheduling [13]. The Zephyr OS is based on a small-footprint kernel designed for use on resource-constrained and embedded systems ranging from a simple embedded environmental sensors and LED wearables to sophisticated embedded controllers, smart watches, and IoT grids. The Zephyr kernel supports multiple architectures, including ARM Cortex-M, Intel x86, ARC, NIOS II, Tensilica Xtensa and RISC-V 32 which makes the OS compatible with over 200 development boards [27]. C and C++ are the languages used to develop applications in Zephyr. Zephyr provides a complete network stack for communication and includes multiple protocols. The applications can be developed, built and tested using the native posix port.

Table 2 Overview of IoT Operating Systems

IoT Operating System	Provider	License	Processor/CPU
Mbed OS	ARM	Apache 2.0	ARM Cortex-M
Contiki OS	Thingsquare	3-clause BSD	ARM Cortex-M, MSP430, AVR, x86,…
RIOT OS	FU Berlin	LGPLv2.1	ARM Cortex-M, MSP430, ARM7, AVR, x86, Cortex-M23, ESP8266, ESP32, MIPS32, MSP430,…….

Apache Mynewt	Apache Foundation	Apache 2.0	Cortex-M, RISC-V
Zephyr	Linux Foundation	Apache 2.0	ARM Cortex-M, x86, ARC, NIOS II,...
Android Things	Google	Apache 2.0/GPLv2	High-end processors such as x86, ARM Cortex-A (32/64)
Windows 10 IoT	Microsoft Corporation	Commercial	ARM Cortex-A7, Snapdragon 400, ARM Cortex-A53
Embedded Linux	Multiple providers	GPL, GPLv2,…..	ARM Cortex-A8, ARM Cortex A-53, ARM Cortex A7 ARM Cortex-A9, Intel Core i3,i5,i7,.....

3.4.6 Android Things

Android Things is an IoT operating system developed and maintained by developed by Google. Google announced its IoT OS Brillo at Google I/O 2015, which was rebranded to Android Things. As the name indicates, it is a simplified and trimmed down variant of Android, designed to run on low-power IoT devices. It supports development in both C and C++ programming language. It is built on top of monolithic kernel and provides completely fair scheduler [13]. The Peripheral I/O APIs allow apps to communicate with sensors and actuators using industry standard protocols and interfaces. The interfaces supported by Android Things are GPIO, PWM, I^2C, SPI, UART. Apps for IoT devices can be built using existing Android development tools, APIs, and resources along with new APIs that provide low level I/O and libraries for common components like temperature sensors, display controllers, and more [28]. Android Things provides a GUI interface, but its high memory requirements make it unsuitable for low-end, constrained IoT devices; rather, it is designed for high-end devices. Currently, Android Things OS supports two development boards - Raspberry Pi 3 Model B and NXP i.MX7D.

3.4.7 Windows 10 IoT

Microsoft ventured into embedded systems domain with Windows CE in 1996. Microsoft has now withdrawn support for Windows CE and has moved to Windows 10 IoT. The operating system includes the stability and user-friendliness of Windows family of products. Windows 10 IoT comes in three flavours: IoT Enterprise, IoT Core and Server IoT. Licensing of the OS is done through OEM channels. Where reliability and safety are important, Windows OS is preferred over free or open source ones. Windows 10 IoT finds applications in aerospace, automotive, healthcare and industrial systems. Microsoft has built several other products around IoT. Windows 10 IoT brings Artificial Intelligence (AI) and Machine Learning (ML) to smart devices with Windows ML and support from Azure IoT Edge [29]. Windows 10 IoT supports boards like AAEON Up Squared, DragonBoard 410c, NXP i.MX 7, NXP i.MX 8M/8M Mini and Raspberry Pi 2/3B.

3.4.8 Embedded Linux

Embedded Linux does not refer to an individual OS; it is a categorization. Rather than discussing each Linux based operating system separately, all of those products have been discussed under this head. Linux is a very versatile environment suitable for IoT development and is very adaptable in nature. Some examples of embedded linux distributions are Raspbian, Yocto, Ubuntu Core, RTLinux, OSMC, Arch Linux ARM, Gentoo and openSUSE. Embedded Linux's popularity can be attributed to its core characteristics: reliability, configurability and low system requirements. Linux works only on embedded systems with at least a 32-bit address space [30]. Contributions from developers all over the world are making Linux stronger by time. Commercially licenced and "closed" operating systems are not recommended if modifications are to be made to match the requirements of the deployment environment. The flexibility of Linux, combined with consistency in performance, architectural tiers, virtualization and cloud support makes Linux distros a popular choice among IoT developers [12].

4. DISCUSSION

Section 3 of this paper presented an overview of the popular options available as IoT OS. It can be observed that most of the system software products mentioned in Section 3 are open source projects. To compare open source solutions with commercial software, the factors like technical specifications (such as support for hardware, functionality, reliability, performance, network connectivity and standards), software license of the product, project governance, founding members, commercial and community support, and the userbase has to be considered.

Contiki, Zephyr, Mynewt, and RIOT introduced in this paper are open source system software without integrated cloud services [1]. Contiki and RIOT are both mature software projects with an active developer community.

Google's Android Things in combination with Android operating system, Google cloud infrastructure, and other cloud services can be used in high-performance devices with x86 architecture. The only downside is that the OS will exclusively support Google product portfolio and will not let the developers to test software products or services from other vendors.

The other leading environment is ARM mbed OS. It has the backing of Arm Holdings and targets low-performance MCU based devices. Mbed has greater importance considering the share of ARM products in the market.

Embedded Linux is also a favourite among developers. Some Linux variants are delivered as free operating systems, while others are maintained as open source projects.

Windows 10 IoT is a commercial solution and is a widely adopted OS 52% of Edge nodes or Gateways use Windows operating system [31]. So, developers adopt Windows OS on IoT devices to ensure better communication between all devices on the network. Considering wide range of customizable services and software offered by Microsoft, Windows 10 IoT is expected to improve its position in the coming years.

To provide perspective on the usage and adoption of IoT OSs, the IoT Developer Survey 2019, undertaken by the Eclipse Foundation can be taken as reference. Linux held on to its top position among operating systems. The survey does not disclose the share of Linux in total IoT usage, but shows Windows holding second position with 20 percent. Other than Linux, the other popular choices include Windows, mbed OS, RIOT OS, Contiki and others. The share of non-Linux operating systems is illustrated through a graph (Figure 5) [31].

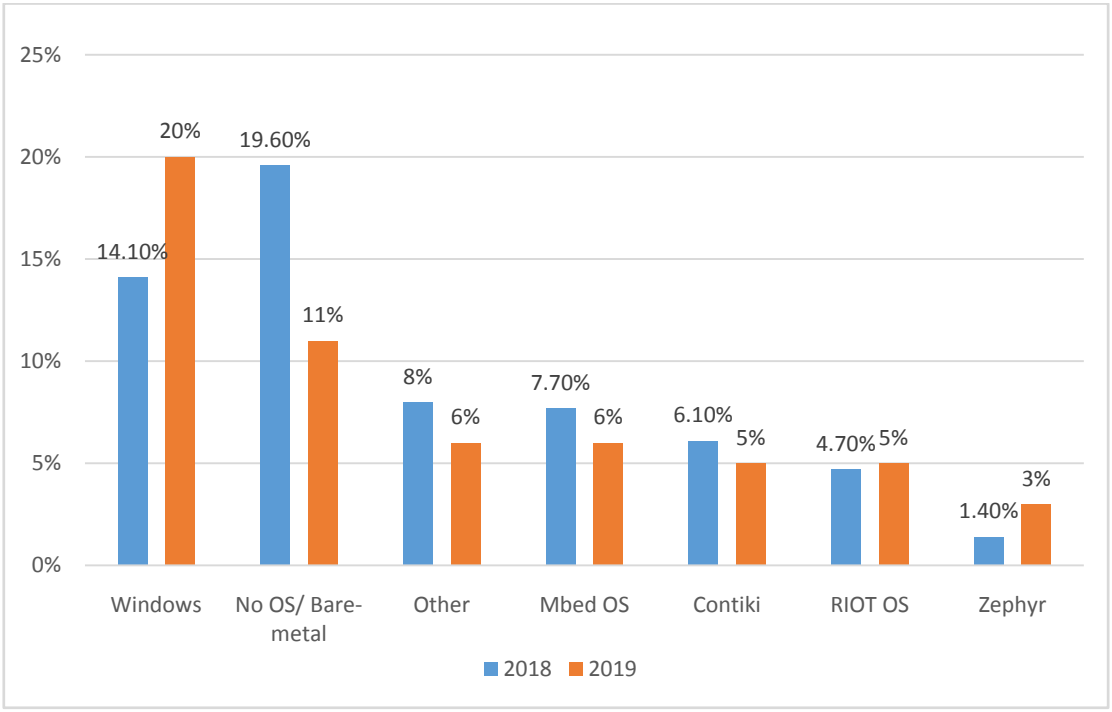

Figure 5 Market share of non-Linux IoT operating systems

5. CONCLUSION

IoT operating systems are used by hobbyists, developers and researchers. There are several options available in the market as both free and commercial distributions. Open source development and cross platform applications are experiencing great growth in the field of IoT. The availability of low cost development boards makes IoT more pervasive, open and community driven. This has encouraged industry leaders like Google, Microsoft, Canonical, Intel, ARM, MATLAB and Apache to venture into developing system software or a complete software suite for IoT solutions. Innovations like NVIDIA Jetson, ThingSpeak [32] and TensorFlow Lite indicates that the IoT industry is working towards a better coupling with evolving technologies like cloud computing, data analytics, machine learning and GPU computing.

Single Board Computers are not just tools for prototype development; they are adopted in several parts of the world as low-cost alternatives to desktop computers. The introduction of simple and intuitive operating systems like Raspbian, KODI distributions and Android has a great role to play in that. So, there is a need to create more general purpose, customer oriented operating systems targeting SBCs. IoT solutions can be developed for any domain. But in general, there is a convergence towards creating a connected world as proposed by concepts like Internet of Everything and Campus of Things. In the years to come IoT will grow into a mainstream technology and there will be standardization in IoT operating systems and development environments.

6. ACKNOWLEDGMENTS

The author would like to acknowledge the support provided by the academic institution Naipunnya Institute of Management and Information Technology in completing this paper. The information and insights derived from community discussion forums and developer groups are also thankfully acknowledged.

7. REFERENCES

[1] Amiri-Kordestani, Mahdi, and Hadj Bourdoucen. "A survey on embedded open source system software for the internet of things." Free and Open Source Software Conference. Vol. 2017. 2017.

[2] “TensorFlow Lite,” TensorFlow. [Online]. Available: https://www.tensorflow.org/lite

[3] K. J. Singh and D. S. Kapoor, "Create Your Own Internet of Things: A survey of IoT platforms.," in IEEE Consumer Electronics Magazine, vol. 6, no. 2, pp. 57-68, April 2017

[4] “loop(),” Arduino Reference. [Online]. Available: https://www.arduino.cc/reference/en/language/structure/sketch/loop/

[5] Pritish Sachdeva and Shrutik Katchii, "A Review Paper on Raspberry Pi", International Journal of Current Engineering and Technology, Vol.4, No.6 ,pp. 3818-3819, 2014.

[6] "Bringing the Power of AI to Millions of Devices," *NVIDIA*.[Online].Available:https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-nano/

[7] "Top 15 Best IoT Operating System For Your IoT Devices in 2020", UbuntuPIT, 2020. [Online]. Available: https://www.ubuntupit.com/best-iot-operating-system-for-your-iot-devices/

[8] a. vikasG, "IoT Operating Systems", Devopedia, 2020. [Online]. Available: https://devopedia.org/iot-operating-systems

[9] D. Guinard, "Operating Systems for IoT Embedded Systems – Web of Things", Webofthings.org, 2020. [Online].Available:https://webofthings.org/2016/12/12/iot-os-embedded/

[10] A. Prabhu. S, G. Prabhu and P. R, "A STUDY OF OPERATING SYSTEM FOR EMBEDDED SYSTEMS", International Journal of Latest Trends in Engineering and Technology, no., pp. 54-58, 2016. Available:https://www.ijltet.org/journal/148299172610.pdf

[11] Operating Systems. [Online]. Available: https://homepage.cs.uri.edu/faculty/wolfe/book/Readings/Reading07.htm

[12] "IoT Operating Systems," Arrow.com, 10-Sep-2018. [Online].Available:https://www.arrow.com/en/research-and-events/articles/iot-operating-systems

[13] Y. B. Zikria, S. W. Kim, O. Hahm, M. K. Afzal, and M. Y. Aalsalem, "Internet of Things (IoT) Operating Systems Management: Opportunities, Challenges, and Solution," Sensors, vol. 19, no. 8, p. 1793, 2019

[14] S. Sungjoo and A. Jerraya, "Introduction to Hardware Abstraction Layers for SoC," in Embedded Software for SoC, Boston, MA: Springer, 2003, pp. 179–186.

[15] "Hardware Abstraction: Definition & Purpose", Study.com[Online].Available:https://study.com/academy/lesson/hardware-abstraction-definition-purpose.html

[16] B Kumar, "The Role of Sleep Mode in Embedded Systems", eeweb, 2020. [Online]. Available: https://www.eeweb.com/profile/kumarb/articles/the-role-of-sleep-mode-in-embedded-systems

[17] Schramm, Norbert, and Anita Sabo. "Concurrent programming method for embedded systems." 9th International Symposium of Hungarian Researchers on Computational Intelligence and Informatics. Vol. 41. 2008.

[18] Wael Alabdulaly," Memory Management techniques and Processes Scheduling", International Journal of Scientific & Engineering Research", Volume 7, Issue 4, pp. 1182-1184, 2016

[19] "Embedded Operating Systems for the IoT," cs.virginia.edu. [Online]. Available: https://www.cs.virginia.edu/~bjc8c/class/cs6501-f18/

[20] "Main Page," SELinux Wiki. [Online]. Available: http://www.selinuxproject.org/page/Main_Page

[21] Ogawa, Keigo, et al. "IoT Device Virtualization for Efficient Resource Utilization in Smart City IoT Platform." 2019 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops). IEEE, 2019.

[22] Jabeen, Qamar, et al. "A survey: Embedded systems supporting by different operating systems", International Journal of Scientific Research in Science, Engineering and Technology ,Vol.2, Issue 2,pp. 664-673, 2016.

[23] Milinković, Aleksandar, Stevan Milinković, and Ljubomir Lazić. "Choosing the right RTOS for IoT platform." Proceedings of the international scientific professional symposium Infoteh, Jahorina. 2015.

[24] Qutqut, Mahmoud H., et al. "Comprehensive survey of the IoT open-source OSs." IET Wireless Sensor Systems 8.6 (2018): 323-339.

[25] "The friendly Operating System for the Internet of Things. Learn more.," RIOT. [Online]. Available: https://www.riot-os.org/#usage

[26] "Apache Mynewt," Apache Mynewt. [Online]. Available: https://mynewt.apache.org/

[27] "Supported Boards," Supported Boards - Zephyr Project Documentation, 14-Feb-2020. [Online]. Available: https://docs.zephyrproject.org/latest/boards/index.html

[28] "Android Things 1.0 Features and APIs Android Developers," Android Developers. [Online]. Available: https://developer.android.com/things/versions/things-1.0

[29] Terry Warwick, "Overview of Windows 10 IoT Core - Windows IoT," Overview of Windows 10 IoT Core - Windows IoT | Microsoft Docs. [Online]. Available: https://docs.microsoft.com/en-us/windows/iot-core/windows-iot-core

[30] Embedded Linux. [Online]. Available: https://www.itu.dk/research/rces/emli.html

[31] Iot.eclipse.org, 2020. [Online]. Available: https://iot.eclipse.org/resources/iot-developer-survey/iot-developer-survey-2019.pdf

[32] "ThingSpeak for IoT Projects," IoT Analytics - ThingSpeak Internet of Things. [Online]. Available: https://thingspeak.com/