ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

Visit: www.ijmrsetm.com

Volume 2, Issue 3, March 2015

A Comprehensive Survey on IoT-based Operating Systems

Umang

Assistant Professor, Department of Information Technology, Kumaun University, SSJ Campus, Almora, India

ABSTRACT: The Internet of Things (IoT) plays a crucial role in developing and delivering solutions to various situations in real life and consists of a broad spectrum of devices ranging from sensors powered by microcontrollers to devices powered by processors equivalent to those found in smartphones. Much research has been conducted recently on device IoT-based operating systems such as standard UNIX, Windows and current real-time operating systems. This study is a comprehensive survey of operating systems available for IoT-based devices. First, the paper introduces the various operating systems available currently for the IoT environment and the different parameters created to make these operating systems viable for IoT. Secondly, this study focuses on distinguishing the present Operating Systems. Finally, more focus will be on the following operating systems which are as follows: Contiki, Tiny OS, RIOT, Mbed OS, and FreeRTOS.

I. INTRODUCTION

The Internet of Things (IoT) based devices is designed as a crucial aspect of human life, with IoT devices having dynamic and heterogeneous nature used in many circumstances like intelligent vehicles, remote sensing, smart car parking, smartphones, electronic appliances, controlling and monitoring systems etc. IoT-integrated systems provide comprehensive services in a connected network for exchanging information. However, to accomplish the purpose of the various application scenarios, nodes must be cheap and have a tiny form factor coupled with the ability to function long enough on battery supply. These requirements lead to very constrained computing power and available memory development. Furthermore, nodes are very limited in communication, resorting to radio transmissions as frequently as possible to conserve energy. Otherwise, they use energy-efficient wireless communication technologies, which typically offer little bandwidth and a minimal payload per packet.

To match such constraints, specialized proprietary protocols have been designed and used. Standard Internet protocols, such as TCP and IP, were initially deemed inappropriate in this context [1]. However, as various distributed embedded systems have emerged recently, power-line communications and spontaneous wireless networks are now expected to interconnect heterogeneous devices [2]. The use of IP on these devices is increasingly considered the cheapest alternative in the long run, and contrary to IPv4, IPv6 is viewed as a viable and desirable solution to power IoT devices due to its larger address space, more appropriate packet header design and convenient features that enable bootstrapping and neighbour discovery.

The IoT devices are integrated into different objects using software working dynamically with the use of wireless sensor networks, RFID technologies, and the internet [3], [4]. Moreover, the operating systems for the IoT environments are developed; they use very few kilobytes of RAM, with low power consumption and also, and they are specifically designed and optimized for the best performance for a particular set of microprocessor-based platforms beyond which such OS becomes irrelevant in its application [5].

These IoT operating systems do not compromise communication, networking, or security features, compared to routine OS like Windows, Mac etc. Still, some come built-in with many pre-installed, pre-integrated applications, drivers and other network protocols. Moreover, these OS deploy uniquely designed security features to enrich the IoT infrastructure and avoid compromising the OS's stability and usability.

OS in these IoT devices functions as are source manager that manages CPU time and secondary storage such as hard disk, memory and network throughput. An IoT OS is intended to operate within the limitations of the Internet of Things, including size, memory, and energy processing capability. These discrete features of IoT are required for

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

Visit: www.ijmrsetm.com

Volume 2, Issue 3, March 2015

portable, efficient, flexible and lightweight systems with small memory tracks. Therefore, various operating systems, such as Windows 8.1, ARM, Linux etc., are seriously competing to design IoT-based Operating Systems.

Primarily, this study aims to provide a comparative survey amongst the terms, including architectural design, scheduling, programming language model, memory management and probability, hardware support, and a few inclusive drawbacks [6].

II. REQUIREMENTS FOR IOT OS

2.1 Architecture

The architecture of an operating system composes of the kernel and specifies the services to the user. Based on these parameters, architecture can be categorized as follows:

a. Monolithic- Used for multilayer application systems. These can handle higher-order complex computation and have higher processing speed, thus a better throughput. All processes run in kernel space.
b. Microkernel-This architecture provides only the major functionalities like scheduling, inter-process communication and synchronization that run on kernel space. Other functionalities of OS run in threads. Here processes run both in kernel space and user space. They provide high flexibility in processing due to plug-in availability and allow an OS to be designed over the base system.
c. VM architecture- This system is virtual over the actual running system. Due to virtualization, this architecture is slower, but they provide a high level of portability, flexibility and extensibility.
d. Modular architecture enables dynamically adding and replacing components into the kernel at runtime. Each module represents a separate functionality. So, modules can be plugged in and out as per the requirement.
e. Layered architecture—Multi-layered architecture is designed for a specific requirement and is not flexible regarding functionality. But this kind of operating system is easy to operate and handle.

2. 3. Programming/Development Model

The programming model decides how an application developer can model the program. Typical programming models can be split into event-driven and multithreaded schemes for IoT operating systems directly influencing concurrency handlers and memory design. Now it depends on the IoT device in which this OS will be installed so that they can make it flexible and extensible and achieve the device's purpose. Therefore, OS should be programmed to be redesigned and upgraded. A Software Development Kit (SDK) present in the OS helps in modelling libraries and improvising the operating system interface, memory, and power model.

2.4. Scheduling

Scheduling strategy is the main factor that determines system performance and is directly proportional to the capabilities of an OS. The scheduling algorithm depends on the latency (response time, turnaround time), performance, energy efficiency, real-time capacities, fairness and waiting time. The scheduling algorithms are priority and non-priority schedulers accompanied by preemptive and non-preemptive nature. Preemptive performs the highest priority tasks topping all running tasks, whereas non-preemptive start a new task post-completion of the current studies [7].

2.5. Memory Management

Memory management offers an idea of managing memory allocation, de-allocation, caching, logical and physical address mapping, memory security and virtual memory as devices are limited, so an OS must have low memory and processing requirements. IoT devices typically provide a few kilobytes of memory, millions of times less than connected machines (smartphones, laptops, tablets etc.). The amount of memory management requirement relies on the type of application and the underlying platform support. The distribution of memory may be static or dynamic. If memory is fixed, it is static memory, and when it is required to have a flexible nature, it is called dynamic memory. Memory distribution is more accessible through the static method, but the active strategy can provide flexibility in runtime memory acquisition.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

Visit: www.ijmrsetm.com

Volume 2, Issue 3, March 2015

2.6. Energy Efficiency

Energy efficiency becomes crucial for battery-powered IoT systems and should be considered when developing an IoT OS. Most IoT systems are resource-bound in nature [8]. Therefore, batteries or several other limited energy sources are used to operate them. Scenarios for IoT implementation are varied, challenging and sometimes very distant. Humongous IoT network size requires IoT OS to run the IoT equipment for many years to be power efficient and to route power in a new direction in case a device goes down. In addition, OS should be powerful enough to handle power failure to save the current state.

2.7. Security

When it comes to the IoT ecosystem, security is a significant concern. OS at various levels in this ecosystem needs a high level of security to make it a robust system. Security enables encryption of user data and reliable communication in heterogeneous devices present at different layers of this ecosystem. Security can be in terms of data restriction, user control to access the system in terms of authorization and authentication, and physical security in case of foreign attacks. The device is not only concerned with security but also cover system requirement that manages the device in case it is in unsafe hands to ensure its integrity and avoid loss of data. Another security concern comes into play at the communication level to prevent data leaks during inter-device or device internet communication.

III. OPERATING SYSTEM FOR IOT DEVICES

Contiki OS was designed by Adam Dunkels and further improved [9], [10], [11]; Contiki OS is an open-source, flexible and lightweight IoT operating system. The programming model of the Contiki operating system is based on proto threads for efficient operation in a resource-constrained environment. In addition, the Contiki operating system features Cooja, a network simulator which simulates Contiki nodes [12]. These Contiki nodes are of three types: Emulated, Cooja, and Java.

Proto thread is a combination of event-driven programming systems and multithreaded ones. It is fortified with robust and energy-efficient web communication facilities that connect low-cost, power-restricted small microcontrollers to the internet and run on various power-restricted appliances. Contiki uses cooperative or preemptive based scheduling for the processes. Contiki is intended to use well-known and well-tested IPv4, IPv6, and HTTP standards. The small memory demands create Contiki suitable for systems with low energy constraints in C language. Contiki operates on a broad spectrum of small platforms, varying from 8051-powered -on-a-chip systems through the MSP430 and the AVR to various ARM devices.

3.1. TinyOS

Developed for wireless sensor networks [13], [14] by Tiny OS Alliance, Tiny OS is an open-source and component-based embedded operating system. The design of this operating system includes low-powered wireless systems such as personal area networks (PANs), universal computing, intelligent buildings and smart meters. It supports C programming language. This is strongly related to the Tiny OS component-based model, so accessing hardware will become relatively easy. The architecture of the Tiny OS has been made secure over the years with the implementation of TinySec [15] and various types of embedded security layers [16], [17], [18], [19]. Tiny OS version2.1has supportive TOS threads, i.e. if the CPU is not in

OS	Min RAM	Min ROM	C Support	C++ Support	Multi-Threading	MCU w/o MMU	Modularity	Real-Time
Contiki	<2kB	<30kB	○	✘	○	✓	○	○
Tiny OS	<1kB	<4kB	✘	✘	○	✓	✘	✘
Linux	~1MB	~1MB	✓	✓	✓	✓	○	○
RIOT	~1.5kB	~5kB	✓	✓	✓	✓	✓	✓

TABLE I. KEY CHARACTERISTICS OF CONTIKI, TINYOS, LINUX, AND RIOT. (✓) FULL SUPPORT, (○) PARTIAL SUPPORT, (✘) NO SUPPORT. THE TABLE COMPARES THE OS IN MINIMUM REQUIREMENTS IN TERMS OF RAM AND ROM USAGE FOR A BASIC APPLICATION, SUPPORT FOR PROGRAMMING LANGUAGES, MULTI-THREADING, MCUS WITHOUT MEMORY MANAGEMENT UNIT (MMU), MODULARITY, AND REAL-TIME BEHAVIOR.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

Visit: www.ijmrsetm.com

Volume 2, Issue 3, March 2015

use, then it is the responsibility of an application to provide the CPU explicitly. The scheduler uses preemptive FIFO scheduling to execute threads and schedules as a high-priority thread.

3.2. RIOT OS

Developed by INRIA, HAW Hamburg and FU Berlin initially, the operating system of RIOT provides the developer with a friendly environment. RIOT usually supports most microcontroller architectures (16-bit, 32-bit, 8-bit) and low-power IoT devices.

To achieve minimum memory usage, the system is designed in a modular way. Thus, the system's configuration can be customized to meet the particular specification. The size of the kernel itself is minimized, thus requiring only a few hundred bytes of RAM and program storage. Dependencies between the modules are reduced to an absolute minimum. Developed using C and C++, the SDKs available for developing applications in RIOT OS are gcc, Valgrind and gdb. Moreover, the SDK framework supports application programming in C and C++.

3.3. RIOT

RIOT provides multithreading with a preemptive, priority-based and tickless scheduler. RIOT's scheduler works without periodic events and can be considered a tickless scheduler. RIOT will switch to the idle thread whenever there are no pending tasks. The only function of the idle line is to determine the most profound possible sleep mode, depending on the peripheral devices in use. In this manner, it is guaranteed to maximize the time spent in sleep mode, thus, minimizing the energy consumption of the whole system.

Furthermore, only external or kernel-generated interrupts wake the system from an idle state. In addition, all kernel functions are kept as small as possible, which allows the kernel to run even on systems with a low clock speed. Finally, the scheduler is designed to minimize the occurrences of thread switching, hence, reducing the overhead by context switching.

For the reduction of the inherent drawbacks like code stack usage, inter-process messaging and thread management overhead, multithreading is designed. Furthermore, to achieve minimum memory usage, the system is designed in a modular way. Thus, the system's configuration can be customized to meet the particular specification. Furthermore, the size of the kernel itself is minimized, thus requiring only a few hundred bytes of RAM

Table Comparison of operating system in terms of design aspects

Operating system	Architecture	Scheduler	Programming model	Programming language	Real time support	IoT devices	OS type
TinyOS	Monolithic	Non-preemptive FIFO	Event-driven concurrency	NesC	No	Low	Non-Linux
Contiki	Modular	Preemptive FIFO	Multithreading and event driven	C	Yes	Low	Non-Linux
RIOT	Microkernel	Preemptive priority based	Multithreading	C	Yes	Low	Non-Linux
FreeRTOS	Microkernel	Preemptive priority based and cooperative scheduler	Multiple threads, mutexes, semaphores	C and assembly functions	Yes	Low	Non-Linux

And program storage. Dependencies between the modules are reduced to an absolute minimum. RIOT OS supports all effective communication and networking protocols, including IPv6, 6LoWPAN, RPL, CoAP, UDP, TCP, CBOR, CCN-lite, Open WSN, and UBJSON.

3.4. Mbed OS

The Real-Time Operating System (RTOS) Mbed OS is created for restricted IoT systems by Advanced RISC Machine (ARM)in collaboration with its technological partners; mbed OS is developed for 32-bit ARM Cortex-M microcontrollers [20]. The whole OS is written using C and C++ language. It provides a preemptive scheduler and is based on a monolithic kernel.

ISSN: 2395-7639

International Journal of Multidisciplinary Research in Science, Engineering, Technology & Management (IJMRSETM)

Visit: www.ijmrsetm.com

Volume 2, Issue 3, March 2015

The software development kit (SDK) form bed OS provides the software framework for the developers to develop various microcontroller firmware to be run on IoT devices. These SDK comprises core libraries with the following components: Networking, Test scripts, Microcontroller peripheral drivers, RTOS and runtime environment, Build Tools, and Debug Scripts. The applications for mbed OS can only be developed online using its native online code editor cum compiler known as mbed online integrated development environments (IDEs). While writing code can only be done through a web browser, its compilation is done by the ARMCC C/C++ compiler in the cloud. On the connectivity front, the mbed OS supports the following connectivity technologies: Bluetooth Low Energy, Wi-Fi, Zigbee IP, Zigbee LAN, Cellular, Ethernet, and 6LoWPAN.

3.5. FreeRTOS

Developed by Real Time Engineers Ltd., the FreeRTOS is developed for platforms such as ARM7, ARM9, ARM Cortex-M3, ARMCortex-M4, ARMCortex-A, RM4x, TMS570,
Cortex-R4, AtmelAVR, AVR32, HCS12, AlteraNiosII,

Table Comparison of Operating System in terms of Memory and Operational Support

Operating system	RAM (KB)	ROM (KB)	Processor (bits)	Memory allocation	Communication network protocol	Simulation support	Multimedia support
TinyOS	10	4–8	8	Static	TCP, UDP, IPv6, 6LoWPAN, RPL, CoAP, Hydrogen routing Protocol, HTTP	TOSSSIM	Limited codecs support
Contiki	2	40	16–32	Dynamic	MQTT, uIP, Rime stack, UDP, TCP, HTTP, 6LoWPAN, RPL, IPv6, CoAP	Cooja	Limited codecs support
RIOT	1.5	5	8–16–32	Dynamic	CoAP, TCP, HTTP, 6LoWPAN, RPL, IPv6, IPv4, IETF	Cooja	Full support
LiteOS	4	128	32	Dynamic	LTE, NB-IoT, Zigbee, 6LoWPAN	AVRORA	No support
FreeRTOS	10	12	32	Dynamic	TCP/IP, CoAP, 6LoWPAN	POSIX	Limited support

MicroBlaze, CortusAPS1,CortusAPS3, CortusAPS3R,CortusAPS5, CortusFPF3, CortusFPS6, CortusFPS8, FujitsuMB91460series, FujitsuMB96340series, Coldfire, V850, 78K0R, RenesasH8/S, MSP430,8052, X86, RX, SuperH, PIC, AtmelSAM3, AtmelSAM4, AtmelSAM7, AtmelSAM9. Written mostly in C with the addition of a few assembly functions, this open-source OS is licensed under Modified GPL [21], [22].

The application development part for FreeRTOS is handled through multiple threads, software timers and semaphores, along with a tickless mode for low consumption of resources by running the various applications.

IV. CONCLUSION

The IoT is indeed very challenging concerning the design of a suitable generic operating system, which must deal with diverse requirements of heterogeneous hardware platforms and application scenarios, provide an adaptive IP network stack, and offer a standard developer-friendly API. OS facilitates the development and subsistence of IoT. The paper discusses various IoT OS based on resource constraints according to the hardware requirement. A comparative overview of open-source IoT OS has been done on aspects like kernel, scheduler, memory management, performance, simulator, security, and power. From the above survey, it can be seen that all the OS for the IoT environment is well equipped with all the foremost networking and communication protocols and security features

and optimized for efficient usage of computing power in our source constraint environment. According to resources, the different OS has security systems to overcome attacks. Tiny OS has a TinySec library to provide message authentication, integrity and confidentiality semantic security. Contiki has ContikiSec Transport Layer Security (TLS) in 3 modes authentication, confidentiality, and integrity. RIOT has Cyber-Physical Ecosystem (CPS) to interact, monitor and control intelligent objects. FreeRTOS has Wolf SSL for security, authentication, integrity, and confidentiality.

REFERENCES

[1] S. P. Kumar, “Sensor networks: Evolution, opportunities, and challenges,” Proceedings of the IEEE, vol. 91, no. 8, pp. 1247–1256, 2003. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/ epic03/wrapper.htm?arnumber=1219475
[2] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelffl, Eds., Vision and challenges for realizing the Internet of Things. Cluster of European Research Projects on the Internet of Things, European Commision, 2010.
[3] Hermann Kopetz. "Internet of things." In Real-time systems, pp. 307-323. Springer US, 2011.
[4] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic and Marimuthu Palaniswami. "Internet of Things (IoT): A vision, architectural elements, and future directions." *Future Generation Computer Systems* 29, no. 7 (2013): 1645-1660.
[5] http://micrium.com/iot/iot-rtos/
[6] P. Gaur and M. Tahiliani, "Operating Systems for IoT Devices: A Critical Survey", 2015 IEEE Region 10Symposium, 2015.
[7] P. Dutta and A. Dunkels, Operating systems and network protocols for wireless sensor networks, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, Vol. 370, No. 1958, pp. 68–84, 2012.
[8] Adam Dunkels, Oliver Schmidt, Niclas Finne, Joakim Eriksson, Fredrik Österlind, Nicolas Tsiftes and Mathilde Durvy. "The Contiki OS: The Operating System for the Internet of Things." (2011).
[9] Adam Dunkels, Bjorn Gronvall, and Thiemo Voigt. "Contiki- a lightweight and flexible operating system for tiny networked sensors." In *Local Computer Networks, 2004. 29th Annual IEEE International Conference on*, pp. 455-462. IEEE, 2004.
[10] http://www.contiki-os.org/
[11] Anuj Sehgal. "Using the Contiki Cooja Simulator " (2013).
[12] http://en.wikipedia.org/wiki/TinyOS
[13] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer and David Culler. "TinyOS: An operating system for sensor networks." In *Ambient intelligence*, pp. 115-148. Springer Berlin Heidelberg, 2005.
[14] Chris Karlof, Naveen Sastry, and David Wagner. "TinySec: a link layer security architecture for wireless sensor networks." In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pp. 162-175. ACM, 2004.
[15] David J. Malan, Matt Welsh, and Michael D. Smith. "A public-key infrastructure for key distribution in TinyOS based on elliptic curve cryptography." In *Sensor and Ad Hoc Communications and Networks, 2004. IEEE SECON 2004. 2004 First Annual IEEE Communications Society Conference on*, pp. 71-80. IEEE, 2004.
[16] Arijit Ukil, Jaydip Sen, and Sripad Koilakonda. "Embedded security for Internet of Things." In *Emerging Trends and Applications in Computer Science (NCETACS), 2011 2nd National Conference on*, pp. 1-6. IEEE, 2011.
[17] Sachin Babar, Antonietta Stango, Neeli Prasad, Jaydip Sen, and Ramjee Prasad. "Proposed embedded security framework for internet of things (IoT)." In *Wireless Communication, Vehicular Technology, Information Theory and Aerospace & Electronic Systems Technology (Wireless VITAE), 2011 2nd International Conference on*, pp. 1-5. IEEE, 2011.
[18] Kresimir Grgic, Drago Zagar, and Visnja Krizanovic. "Security in IPv6-based wireless sensor network—Precision agriculture example." In *Telecommunications (ConTEL), 2013 12th International Conference on*, pp. 79-86. IEEE, 2013.
[19] https://mbed.org/technology/os/
[20] http://en.wikipedia.org/wiki/FreeRTOS
[21] http://www.freertos.org/
[22]J. Gubbi, R. Buyya, S. Marusic and M. Palaniswami," Internet of Things (IoT): A vision, architectural elements, and future directions", Future Generation Computer Systems, vol. 29, no. 7, pp. 1645-1660, 2013.