Review on open source operating systems for internet of things

To cite this article: Zhengmin Wang et al 2017 J. Phys.: Conf. Ser. 887 012044

View the article online for updates and enhancements.
Review on open source operating systems for internet of things

Zhengmin Wang, Wei Li and Huiliang Dong

Information Center, China tobacco ZheJiang industrial CO.,LTD., room 217, building No.1, No.77 Zhongshan South Road, Hangzhou, Zhejiang, China, Email: wangzm@zjtobacco.com

Abstract. Internet of Things (IoT) is an environment in which everywhere and every device became smart in a smart world. Internet of Things is growing vastly; it is an integrated system of uniquely identifiable communicating devices which exchange information in a connected network to provide extensive services. IoT devices have very limited memory, computational power, and power supply. Traditional operating systems (OS) have no way to meet the needs of IoT systems. In this paper, we thus analyze the challenges of IoT OS and survey applicable open source OSs.

1. Introduction

In recent few years, the interest in Internet of Things (IoT) is growing from industrial field and academic field for its applications. The development of several technologies such as sensor technology, network communication technology and cloud computing made it possible to achieve the vision of IoT. We are moving towards an automated era, where we have smart planet, smart cities, smart homes, all equipped with intelligent IoT capable of carrying out tasks by themselves. IoT is an integrated system of uniquely identifiable communicating devices which exchange information in a connected network to provide extensive services.

There are a large number of embedded devices in IoT. Compared with the traditional embedded devices, IoT devices generally have scarcity of energy and memory resources. They are generally small and battery-operated. Furthermore, they should be safe and reliable. IoT devices need to support a variety of communication protocols and protocol conversion between the layers and have the ability of cloud computing in application layer. In software aspect, the software of IOT devices is more complex than traditional embedded device software, and it also puts forward higher requirement for embedded operating system. In order to meet this requirement, IoT operating system came into being.

This paper is organized as follows. First, we analyze the structure of IoT and challenges of IoT OS. Then, we survey the main open source operating systems for IoT and we emphasize to compare the network stacks in them. Finally we forecast the trend of IoT operating systems.

2. Structure of IoT

The Internet of things (IoT) is the inter-networking of physical devices, buildings, and other items embedded with electronics, software, sensors, actuators, and network connectivity which enable these objects to collect and exchange data. The classical structure of one IoT system is showed as the following Fig.1. Embedded system is the core of the IoT and it is responsible for managing all the sensors, actuators, drives, and need to connect the network. For a large number of networking needs, the embedded operating system due to the closed features cannot be directly applied to the Internet of things equipment, mobile iOS and Android operating system due to the larger, high power
consumption and other reasons cannot be very good fit networking terminal, therefore the operating system of IoT needs a new round of development and innovation.

![Classical structure of IoT system](image)

**Figure 1.** Classical structure of IoT system

### 3. Challenges of IoT OS

The design of an operating system includes many aspects, such as architecture, scheduling algorithm, memory management, network protocol, security measures, etc. In this section, we just analyze three very important aspects of OS for IoT devices: network connectivity, support for heterogeneous hardware, memory management.

#### 3.1. Network Connectivity

In the foreseeable future, it is expected that trillions of devices will be connected to the Internet. Internet connectivity is a fundamental requirement of IoT devices. IoT system must support low power, highly reliable and internet-enabled communication stack. The stack must be flexible, so that with minimal changes it can be configured to meet the demands of a wide range of IoT applications. The support to IPv6 is mandatory in IoT systems to have unique identities in tremendously large networks. A standardized approach, using latest developments in the IEEE 802.15.4 and IETF working groups, has been proposed \[1, 2\]. Maria etc. describe in detail power-efficient IEEE 802.15.4-2006 PHY layer, the power saving and reliable IEEE 802.15.4e MAC layer, the IETF 6LoWPAN adaptation layer enabling universal Internet connectivity, the IETF ROLL routing protocol enabling availability, and finally the IETF CoAP enabling seamless transport and support of Internet applications\[1\]. The IoT OS should cater for multiple network stacks and for continuous network stack evolution.

#### 3.2. Support for Heterogeneous Hardware

There are various kinds of Internet of things nodes in the market, such as Arduino \[3\], OpenMote nodes \[4\], and TelosB nodes \[5\]. IoT devices are based on various microcontroller architectures and families, such as 8-bit (e.g., Intel 8051/52), 16-bit (e.g., TI MSP430), 32-bit (e.g., ARM7). They have different sizes of RAM and ROM and they can be equipped with a wide variety of communication technologies. Thus the IoT OS should support this heterogeneity in hardware architectures.

#### 3.3. Memory Management

Memory management is an important function of the operating system. IoT devices usually have small resources (KB level RAM and MB level Flash). One of the requirements for IoT OS is to fit within such memory constraints. IoT OS should provide a set of optimized libraries (potentially cross layer) providing common IoT functionality, and efficient data structures.
The use of sections to divide the text of the paper is optional and left as a decision for the author. Where the author wishes to divide the paper into sections the formatting shown in table 2 should be used.

4. Open Source OS for IoT

4.1. F Contiki

Contiki [6] is a flexible and portable OS.

For communication, Contiki provides powerful low-power Internet communication. Contiki supports fully standard IPv6 and IPv4, along with the recent low-power wireless standards: 6lowpan, RPL, CoAP. With Contiki’s ContikiMAC and sleepy routers, even wireless routers can be battery-operated.

Contiki has microIP (uIP) [7] and Rime stacks. Rime stack comprises a set of custom light-weight protocols. uIP stack is a light-weight stack which supports IPv6. uIP stack uses protocols especially developed for low-power and wireless memory-constrained real-time sensor devices. Another application layer protocol uXMPP (Micro eXtensible Messaging and Presence Protocol) [8] exists for event-driven Contiki for web services such as short messaging, publish subscribe, basic authentication mechanisms etc.

In wireless networks, nodes may need to relay messages from others to reach their destination. With Contiki, even relay nodes, so-called routers, can be battery-operated thanks to the ContikiMAC radio duty cycling mechanism which allows them to sleep between each relayed message. Some call these sleeping routers, we call it sleepy routers.

In situations when bandwidth is at a premium or where the full IPv6 networking stack is overkill, Contiki provides a tailored wireless networking stack called Rime. The Rime stack supports simple operations such as sending a message to all neighbors or to a specified neighbor, as well as more complex mechanisms such as network flooding and address-free multi-hop semi-reliable scalable data collection. Everything runs with sleepy routers to save power.

For memory, Contiki is designed to run in small amounts of memory. A typical system with full IPv6 networking with sleepy routers and RPL routing needs less than 10 k RAM and 30 k ROM. The OS supports dynamic memory allocation but there is no Memory Protection Unit (MPU). Contiki is therefore highly memory efficient and provides a set of mechanisms for memory allocation.

For heterogeneous hardware, Contiki runs on a range of different hardware platforms from 8-bit to 64-bit and is designed to be easy to port to new hardware.

At present, contiki has become the most important choice in the field of IoT.

4.2. TinyOS

Together with Contiki, TinyOS is the most prominent OS for WSN applications, targeting very constrained 8-bit and 16-bit platforms and is known for its sophisticated design [9, 10].

For communication, TinyOS has Berkeley Low-power Internet Stack (BLIP) [11] which consists of TCP, UDP, ICMPv6 and IPv6 and supports 6LoWPAN, RPL and CoAP. The OS uses Hydro routing protocol to ensure reliable unicast communication within an IPv6 sub network with lossy links.

For memory, TinyOS uses compile-time static memory allocation. There are no heaps, function pointers or virtual memory concepts of dynamic memory allocation. In version 2.1 onwards [12], Safe TinyOS has MPU.

For heterogeneous hardware, compared to Contiki, there are only a few hardware supported by TinyOS.

4.3. RIOT

RIOT [13] is developed separately in two parts [14], hardware-dependent and hardware-independent code with well-defined interface. This helps in independent configuration of hardware-specific code to exploit the underlying platform. The kernel and system libraries can remain unchanged in case hardware specific features are modified.

For memory, it doesn’t have MMU or Floating Point Unit. It has both memory allocation provisions: static and dynamic. It uses tickless scheduler to reduce power consumption. The system
enters in sleep mode by switching to idle thread whenever there are no pending tasks and remains in
the mode until woke up by an interrupt.

For communication, RIOT has modular Internet stack. Modular stack is more flexible than layered
stack as modules can be separately maintained. The protocols 6LoWPAN and RPL are provided with
support to TCP, UDP and IPv6.

For heterogeneous hardware, RIOT can run on 8-bit platforms (e.g. Arduino Mega 2560), 16-bit
platforms (e.g. MSP430), and on 32-bit platforms (e.g. ARM).

4.4. FreeRTOS
FreeRTOS [15] is written mostly in C with a few assembly functions. It uses Nabto [16] to provide
communication between peer IoT devices. It supports two privilege modes of operation: kernel mode
and user mode. The programming model is based on tasks and routines. Co-routines are light-weight
tasks having smaller shared stacks. Generally tasks are scheduled preemptively, each having a stack of
its own and co-routines follow cooperative scheduling amongst themselves. A task can preempt a co-
routine. Priority based preemptive scheduling of OS supports real-time systems.

For memory, FreeRTOS supports dynamic memory allocation with two heaps. Fast internal RAM
is used by task stack, while the other heap stores application data in slow external RAM. FreeRTOS
has FreeRTOS-MPU to protect the kernel from failures. It has optional tickless mode to save energy.

For communication, it supports FreeTCP/IP (based on uIP stack, still under development) and lwIP
networking stacks which are based on IPv6 and other less power consuming protocols like 6LoWPAN,
CoAP etc.

For heterogeneous hardware, a variety kinds of MCUs, such as Atmel, Freescale, ST, and TI and so
on, offer official RTOS ports for FreeRTOS.

4.5. MbedOS
MbedOS [17] was developed by the ARM, specifically for the operation of the ARM processor device
design and networking equipment. It includes all the features you need to develop a connected product
based on an ARM Cortex-M microcontroller, including security, connectivity, an RTOS, and drivers
for sensors and I/O devices.

For communication, mbedOS supports IPv6 and 6LoWAN, and takes SSL and TLS, the standard
protocols for securing communications on the internet, and allow the users to include them in their
mbed project with a simple API.

4.6. uClinux
The original uClinux [18] was a derivative of Linux 2.0 kernel intended for microcontrollers without
Memory Management Units (MMUs). However, the Linux/Microcontroller Project has grown both in
brand recognition and coverage of processor architectures. Today's uClinux as an operating system
includes Linux kernel releases for 2.0 2.4 and 2.6 as well as a collection of user applications, libraries
and tool chains. It has full TCP/IP stack which can be replaced with a light-weight stack like uIP and
lwIP [7]. uClinux has higher footprint compared to other OSs.

4.7. µC/OS
µC/OS [19] is a full-featured embedded operating system. Features support for TCP/IP, USB, CAN,
bus, and Modbus. Includes a robust file system, and graphical user interface

µC/OS has a compact, reliable, high-performance TCP/IP protocol stack designed for embedded
applications, featuring dual IPv4 and IPv6 support. It allows multiple interfaces and multiple IP
addresses per interface. It has multicast IPv6, ICMPv6, TCP and UDP. Some application layer
protocols like FTP, POP3 etc., are not compatible with IPv6. The Ethernet device controllers provide
hardware abstraction for easy porting to a variety of Ethernet devices.

4.8. Lite OS
Huawei LiteOS [20] is a Unix-like OS. It has Unix based programming environment. The OS consists
of three subsystems: LiteFS, LiteShell and the kernel.
For memory, LiteFS file system provides support to file and directory operations. The concurrent programming model is thread-based with some support to events via callback functions. Whenever an already registered event occurs, callback function is invoked. Callback functions reduce the number of active threads and improve system efficiency. The memory allocation to applications is done dynamically through system calls using malloc and free API functions. LiteOS has an event trace logger mechanism to provide system insight. Every running application has a buffer to register triggered events. When the buffer runs off the space, it is emptied in a file in external flash. This mechanism can help in protecting memory.

For communication, LiteOS supports plug-and-play routing stack, it does not have any built-in networking protocols.

4.9. Other OSs
In addition to the above, there are many other IoT OS, such as Android Things [21], OpenWSN [22], NutOS [23].

![Classical structure of IoT OS](image)

Figure 2. Classical structure of IoT OS

In this section, we list the predominant open source OSs targeting for IoT devices. There is a continuously growing long chain of OSs with a diverse set of features. According to analysis of existing open source OSs, they all have the classical structure as Figure 2.

5. Conclusion
IoT devices have some special features as compared to conventional computing devices. Therefore, the OS is to be designed according to the distinct requirements of IoT devices and target application areas. In this paper, we briefly introduce the challenges of IoT OS and briefly review the open source operating systems for IoT and make a simple comparison on network stack of the state-of-the-art OS. The paper would help researchers to understand the IoT and choose the appropriate OS according to their requirements.

6. References


