SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348 – 8549 / https://doi.org/10.14445/23488549/IJECE-V9I5P101

Volume 9 Issue 5, 1-7, May 2022

Review Article

Comparison of Nano Q Plus Operating System with other Operating Systems in Wireless Sensor Networks

Rajshree[1], Abhiruchi Passi[2]

[1,2]*Department- FET, Manav Rachna International Institute of Research and Studies, India*

Received: 03 March 2022 Revised: 25 April 2022 Accepted: 06 May 2022 Published: 29 May 2022

Abstract *- Wireless sensor network epitomizes a new generation of real-time embedded systems with considerably different communication constraints from the traditional network system. Basically, the wireless sensor network comprises computing, communication, and sensing elements, allowing the administrator to sense an object in the events in a specified environment. In wireless sensor networks, sensor nodes are specially constructed to collect and deliver information from the base station to the receiver. These sensor nodes are hardware devices with small embedded systems to communicate wirelessly amongst the networks. To keep up with the specific application for which the WSN is designed, the hardware components should be assembled to make the WSN work effectively and correctly without any hardware constraints. Each node needs an operating system to fill the gap between hardware and the corresponding applications. In this paper, we establish the Nano Q plus operating system and represent its performance using various parameters such as energy consumption, routing formation, latency, and propagation delay and compare these performance parameters with other operating systems like Tiny OS MANTIS, etc. on behalf of other operating systems, the Nano Q plus operating system is very easy to manage by the application programmer, can build a large-scale sensor, is flexible, lightweight, dynamic, and has a low power sensor network operating system.*

Keywords *- Wireless Sensor Network (WSN), Operating systems, Nano Q plus operating system, The performance of operating systems, Comparison.*

1. Introduction

Wireless sensor networks recently attained a technology that enables it to develop a low-cost sensor network for various applications such as tracking endangered species across significant remote habitats, tagging small animals, health, home, robots, etc., [5]. ETRI, Korea's largest government-funded research facility in information technology and communications, has developed Nano Q plus operating system, which has a tiny embedded operating system and reconfigurable and scalable properties to fulfill the need for sensor networking application programmers in wireless sensor networks. On behalf of other operating systems, the Nano Q plus operating system is very easy to manage by the application programmer, can build a large-scale sensor, is flexible, lightweight, and dynamic, and has a low-power sensor network operating system [6].

In this paper, we establish the Nano Q plus operating system and represent its performance in various parameters such as energy consumption, routing formation, latency, and propagation delay. We compare these performances and features with other operating systems like Tiny OS MANTIS.

2. Key Elements

There are some vital elements for designing the Nano Q plus operating systems.

Operating system: Compared to available operating systems, the wireless sensor network's operating system should be less complex. Sensor networking application programmers should be concerned about application logic rather than low-level hardware issues such as networking, scheduling, and preempting. Many operating systems have been developed for WSN, like Tiny OS, MANTIS, and Nano Q plus [1-4].

2.1. Performance

Performance is the primary concern for WSN to achieve regard in many applications. The performance of WSN may be affected by the hardware constraints like battery power, small memory, and communication between the networks. The AA battery used in WSN must be operated for up to a few months for efficient power management. Transferring data between the networks must be done as fast as possible. Furthermore, the size of the program image should be small, i.e., less than 10KB.

2.2. MAC layer issues

The WSNs are deployed and work in an unknown environment, so MAC protocols must watch node density, topology, and network size. MAC protocols should minimize the amount of data transferred from one node to another. the latency of the network depends upon the single-hop and multi-hop routing transmission. It tends to delay data from receiver to sender, so MAC protocols should minimize latency. MAC protocols should ensure better usage of bandwidth or capacity.

2.3. Energy efficiency

To ensure the long life and lifetime of the sensor nodes, kernel schedulers and wireless communications modules should manage energy consumption even with less durable energy resources. So, the information on the amount of energy must be provided by the operating system.

2.4. Hardware/software issues

In wireless sensor networks, sensor nodes are specially constructed to collect and deliver information from the base station to the receiver. These sensor nodes are hardware devices with a small embedded system to communicate wirelessly amongst the networks. To keep up with the specific application for which the WSN is designed, the hardware components should be assembled to make the WSN work effectively and correctly without any hardware constraints. Each node needs an operating system to fill the gap between hardware and applications.

2.5. Limited energy

The energy source in wireless sensor networks is a battery. The life of a wireless sensor network entirely depends on the battery, which has a limitation of energy.

3. Operating System

In wireless sensor networks, the data is forwarded through the number of nodes from the base station. These nodes are also known as motes. Motes are wireless devices that are multifunctional and energy-efficient. In other words, motes are tiny computers that work collectively to form a network. Motes are the key to environmental applications of wireless sensor networks. A group of motes gathers the signal from the environment to execute a particular target.

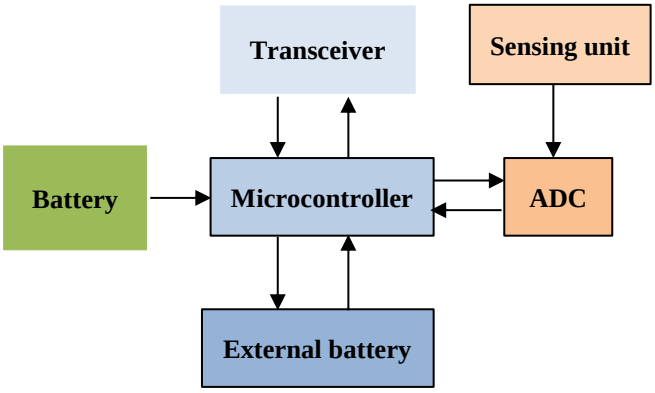

Fig. 1 Block diagram of Operating System

The wireless sensor network has hundreds or even thousands of sensor nodes. These nodes have four essential hardware components: microprocessor/microcontroller, battery, radio transceiver, and physical sensors. So, the hardware components of sensor nodes should be arranged to make WSNs work effectively and correctly. Each node needs an operating system to fill the gap between hardware and applications [8-12]. There are various operating systems, such as Tiny OS, Magnet OS, Mate, Mantis, OSPM, Eyes OS, Sen OS, and Emeralds.

3.1. Tiny OS

A tiny OS is a tiny micro-threaded operating system that allows application software to access hardware directly when needed. It provides modularized components with small storage overhead and processing and guarantees concurrent data flow amongst the hardware devices. Tiny OS is designed with three essential components, i.e., synthetic hardware, hardware abstraction, and high-level components. These components are mapped to specific hardware abstractions.

3.2. Mate

The primary function of mate is to make Tiny OS attainable to nonexpert programmers and provide efficient and quick programming to the entire sensor network. It is designed to work on top of Tiny OS as one of its components. Inmate, a program code is fabricated as a capsule, and these capsules have 24 instructions, and the length of each instruction is 1 byte. Each capsule has version and type information that can quickly deploy into the network to make code injection easy.

3.3. MANTIS

MANTIS is general single-based hardware and enables a flexible environment with a multithread operating system for fast deployment of applications. MANTIS contains a classical layered multi-threaded structure with a network protocol stack and device drivers. It uses preemptive scheduling with time slicing, standard programming language, and I/O synchronization via mutual exclusion. Furthermore, it uses standard C to manufacture the kernel and API.

3.4. Magnet OS

Magnet OS is specially designed for energy conversion and adaption for wireless sensor network applications. It is a distributed adaptive operating system that provides a general abstraction for applications to adapt independent resources, is scalable for an extensive network, changes in a stable manner, and is efficient concerning energy conversion.

3.5. OSPM

OSPM is also known as dynamic power management (DPM), based on a greedy algorithm designed by power management techniques. The primary function of OSPM is to switch the system to sleep mode as soon as it is idle. The deeper the sleep state, the less power consumption and the longer the wake-up time.

3.6. EYES OS

EYES OS works in a simple sequence, like returning a value, entering the sleep mode, and performing a computation. It is specially designed for small memory requirements and coding, where it is capable of distribution and reconfiguration. EYES OS has two main components: the local information component and the network component. the function of the network component is to provide information and retrieve the same from the network and transmit and receive information from other

networks. On the other hand, the function of the local information component is to provide a setting of parameters or variables in sensor nodes and the availability of resources and their status.

3.7. Sen OS

Sen OS is based on a finite state machine (FSM) and quickly realizes concurrency and reconfiguration. It consists of a call-back library of call functions, a state transition table, and a kernel. The state transition table is application-dependent and can be modified or reloaded at the runtime sensor node. A call function and kernel are stored in flash ROM and are statically built-in sensor nodes.

3.8. Contiki

Contiki is a hybrid system that separates essential system support from reprogrammable and dynamically loaded services. These services are called processes that communicate with each other through kernel by posting events. The kernel allows applications and device drivers to communicate with the hardware directly. The kernel does not provide any hardware abstraction that makes it easy to program and replace services.

4. Nano Q plus Operating System

ETRI is Korea's largest government-funded research facility in information technology and communication. It has developed the Nano Q plus operating system, a tiny embedded operating system with reconfigurable and scalable properties to fulfill the need for sensor networking application programmers in wireless sensor networks. On behalf of other operating systems, the Nano Q plus operating system is very easy to manage by the application programmer, can build a large-scale sensor, is flexible, lightweight, and dynamic, and has a low-power sensor network operating system [13].

Two main concepts design Nano Q plus first is a layered design concept, and the second is a modular concept.

4.1. Modular Concept

The architecture of the modular concept consists of three components and four key modules. the three components are Nano Q plus operating system, hardware, and application. The Nano Q plus part represents a platform for sensor networking programmers by offering APIs to develop convenient wireless sensor applications. The application part consists of more than one module interacting with the systems API with the Nano Q plus part. For wireless communication, the hardware part consists of a sensor/actuator, RF modules, and MCU using ATMEGA 128.

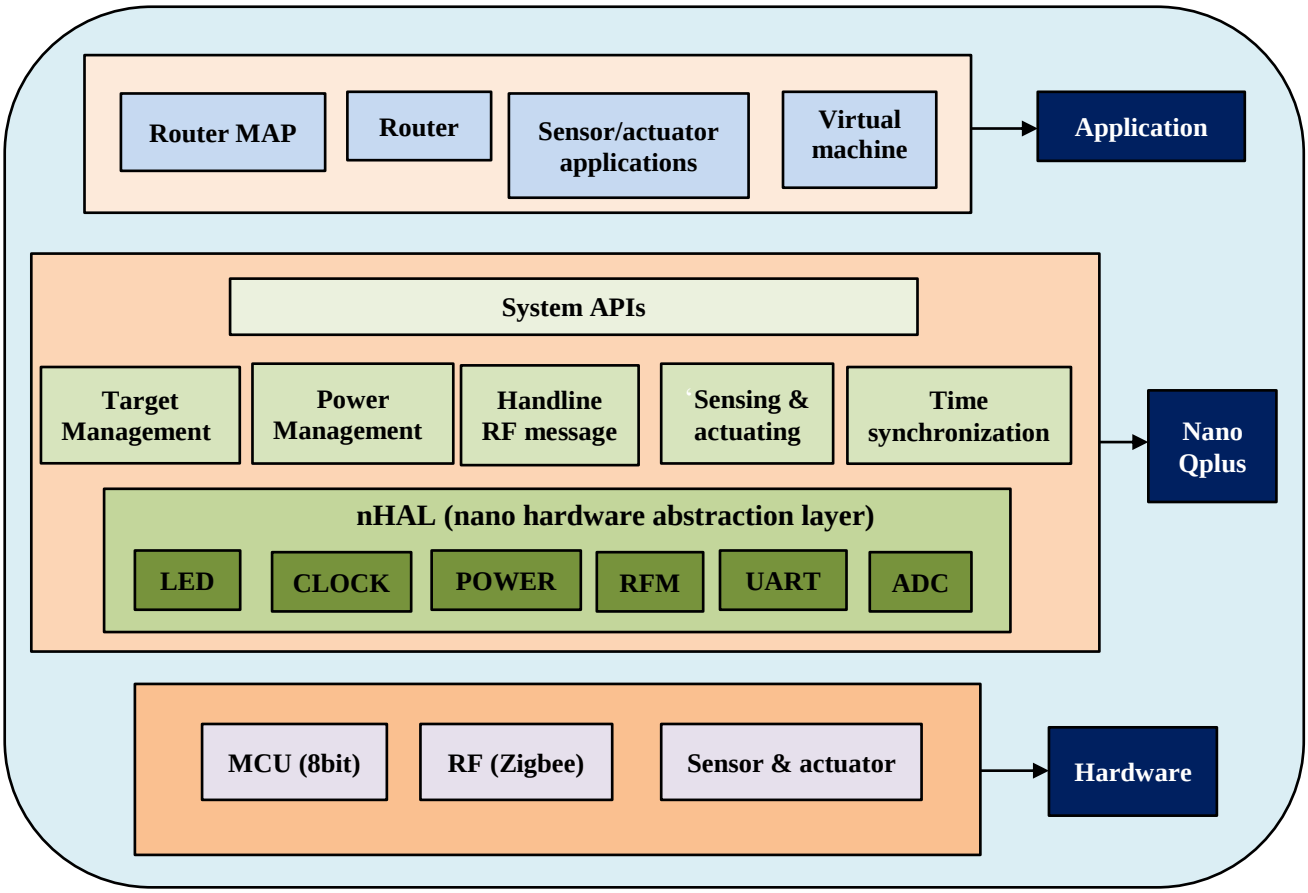

Fig. 2 Shows Nano Q plus operating system technology

4.2. KTey Modules

- nHAL: nHAL is a device driver module for abstracting the hardware part and is composed of various components like ADC, CLOCK, POWER, LED, and RFM. It also provides hardware independence.
- Task management: the task management function is similar to the task scheduler, which is based on a Linux scheduler and mainly focuses on resource limitation and energy efficiency. The task scheduler decides the run order of the task that describes a piece of the node that needs to be executed. Nano Q plus adopt a variety of task schedulers, for example, preemptive round-robin, non-preemptive FIFO, preemptive round-robin with energy efficiency, timed FIFO, and preemptive RR scheduler.
- Power management: In power management, each node in WSN has six operation modes with ATMEGA 128, and the transceiver turns off/on in CC2420 to achieve low power consumption. Power management operates an adaptive operation for a long time and monitors current capacity. To know how long the node will survive in WSN, it has three low, middle, and full levels.
- RF message handling: This part increases the energy efficiency of wireless communication between sensor nodes.

4.3. Layer Design Concept

The layer design concept consists of four layers, i.e., the application layer, network layer, MAC layer, and physical layer. The application layer converts the information into a clear form and gives a large amount of software for various wireless sensor network applications. It is also responsible for controlling traffic. The primary function of the network layer is to provide routing between the sensor nodes. The physical layer is composed of chip coin CC2420 radio for IEEE802.15.4. The MAC layer is responsible for RF channel access with the CSMA-CA algorithm [14].

5. Comparison

The Nano Q plus operating system has been compared with other operating systems such as Tiny OS, MANTIS, Mate, Eyes OS, Sen OS, Magnet OS, OSPM, and Contiki based on their performance and features. The comparison of Nano Qplus and other operating systems based on their performance is shown in table 1, and their comparison based on features is shown in table 2.

In this paper, we introduce a graphing mechanism in figure 5 that shows the variation between the network lifetime of wireless sensor networks based on scheduling algorithms. The tiny OS supports a non-preemptive first-in, first-out (FIFO) scheduling algorithm; therefore, it does not support real-time applications. The wait time for a task depends on the task's arrival time. Contiki OS supports a handler priority scheduling algorithm where events are fired to the target application as they arrive. MANTIS supports preemptive priority scheduling where the target length of time slice configuration is set up to 10 milliseconds, and when there is no thread in the queue, the system goes to sleep mode. Lite OS supports robin-robin and priority-based scheduling, which requires an unavailable task resource; the task enables interrupts and goes to sleep mode. Nano Q plus supports Linux scheduling; therefore, it enhances the network's lifetime, where the thread programming model handles various issues that arise in sensor network applications.

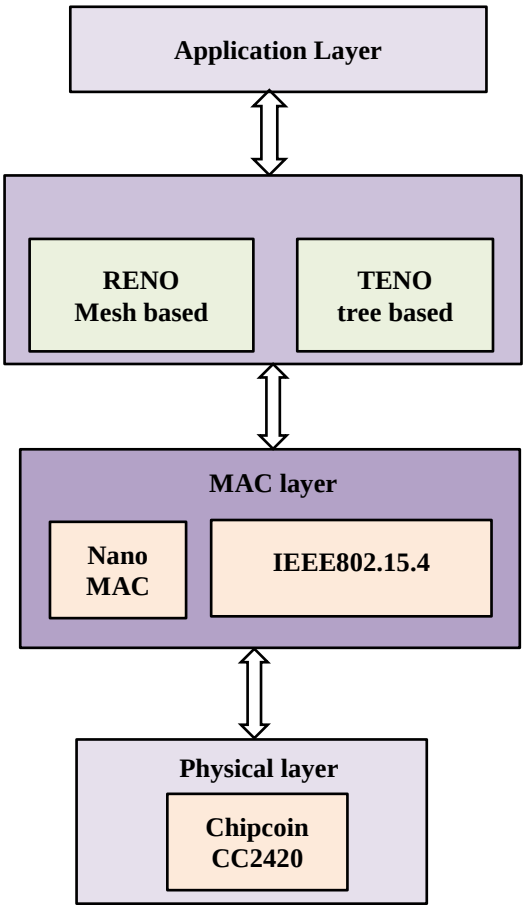

Fig. 3 Nano Q plus network stack

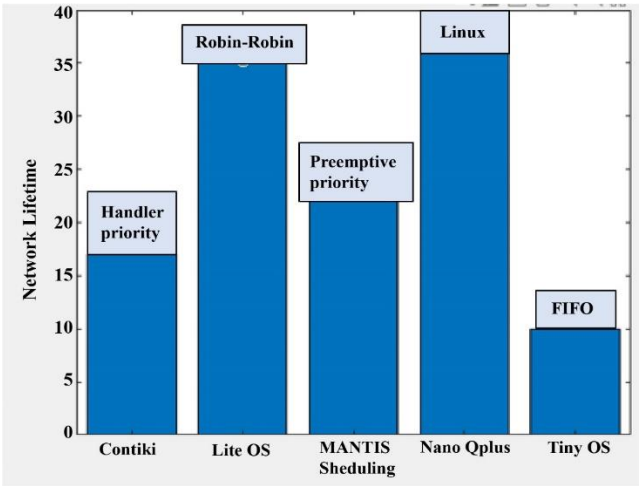

Fig. 4 The graph between network lifetime and scheduling of operating systems

6. Conclusion

In wireless sensor networks, sensor nodes are specially constructed to collect and deliver information from the base station to the receiver. These sensor nodes are hardware components with a small embedded system to communicate wirelessly amongst the networks. To keep up with the specific application for which the WSN is designed, the hardware components should be assembled to make the WSN work effectively and correctly without any hardware constraints. Each node needs an operating system to fill the gap between hardware and applications. In this paper, Nano Q plus operating system has been compared with other operating systems such as Tiny OS, MANTIS, Mate, Eyes OS, Sen OS, Magnet OS, OSPM, and Contiki based on their performance and features. The comparison of Nano Qplus and other operating systems based on their performance is shown in table 1, and their comparison based on features is shown in table 2. Compared to other operating systems, the Nano Q plus operating system is very easy to manage by the application programmer, can build large-scale sensors, is flexible, lightweight, and dynamic, and has a low-power sensor network operating system.

References

[1] John Heidemann et al., "Building Efficient Wireless Sensor Networks with Low-Level Naming," *Proceedings of ACM Symposium on Operating Systems Principle*, pp. 146-159, 2001. *Crossref,* https://doi.org/10.1145/502034.502049

[2] Hill Jason et al., "The Platforms Enabling Wireless Sensor Networks," *Communications of the ACM,* vol. 47, no. 6, pp. 41-46, 2004. *Crossref,* https://doi.org/10.1145/990680.990705

[3] Seongsoo Hong, and Tae-Hyung Kim, "Designing a State-Driven Operating System for Dynamically Reconfigurable Sensor Networks," *Proceedings of the 2003 System on Chip (Soc) Design Conference,* pp. 40-42, 2003.

[4] H Abrach et al., "MANTIS: System Support for Multimodal Networks of in-Situ Sensors," *Proceedings of the 2nd Workshop on Sensor Networks and Applications (WSNA'03),* San Diego, CA, pp. 50-59, 2003. *Crossref,* https://doi.org/10.1145/941350.941358

[5] Niels Reijers, and Koen Langendoen, "Efficient Code Distribution in Wireless Sensor Networks," *Proceedings of the 2nd Workshop on Sensor Networks and Applications (WSNA'03),* San Diego, CA, pp. 60-67, 2003. *Crossref,* https://doi.org/10.1145/941350.941359

[6] Young-Sam Shin Kwangyong Lee, Heeseok Choi, and Seungmin Park, "A Design and Implementation of a Multi-Hop Wireless Sensor Network Based on Nano-Qplus Platform," *In the 20th International Technical Conference on Circuits/Systems, Computers and Communication*, 2005.

[7] D. Ian Chakeres, and M. Elizabeth Belding-Royer, "AODV Routing Protocol Implementation Design," *24th International Conference on Distributed Computing Systems Workshop*, pp. 698-703, 2004. *Crossref,* https://doi.org/10.1109/ICDCSW.2004.1284108

[8] Philip Levis et al., "The Emergence of Networking Abstractions and Techniques in TinyOS," *Proceedings of the First Symposium on Networked Systems Design and Implementation, USENIX Association*, pp. 1-14, 2004.

[9] Chih-Chieh Han et al., "A Dynamic Operating System for Sensor Networks*," Proceedings of the 3rd International Conference on Mobile Systems, Application, and Services (Mobisys' 05)*, Seattle, Washington, 2005.

[10] Bhatti Shah et al., "Mantis OS: An Embedded Multi-Threaded Operating System for Wireless Micro Sensor Platforms," *Mobile Networks and Applications*, vol. 10, no. 4, pp. 563-579, 2005. *Crossref,* https://doi.org/10.1007/S11036-005-1567-8

[11] Han Chih-Chieh et al., "A Dynamic Operating System for Sensor Nodes," *Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services–Mobisys,* 2005. *Crossref,* https://doi.org/10.1145/1067170.1067188

[12] Qing Cao et al., "The Liteos Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks," *IPSN 08, Proceedings of the 7th International Conference on Information Processing in Sensor Networks*, pp. 233–244, 2008**.** *Crossref,* https://doi.org/10.1109/IPSN.2008.54

[13] Dunkels Adam et al., "Protothreads: Simplifying Event-Driven Programming of Memory-Constrained Embedded Systems," *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (Sensys)*, pp. 29-42, 2006. *Crossref,* https://doi.org/10.1145/1182807.1182811

[14] Gay David, Philip Levis, and David Culler, "Software Design Patterns for Tinyos," *ACM Transactions on Embedded Computing Systems,* vol. 6, no. 4, pp. 22, 2007. *Crossref,* https://doi.org/10.1145/1274858.1274860

Annexure

Table 1. Comparison of Nano Qplus and other operating systems based on their performance

Performance	Nano Qplus	Tiny OS	Mate	Magnet OS	MANTIS	OSPM	Eyes OS	Sen OS	Contiki
Design	Modular and layer concept	Tiny micro-threaded operating system	Virtual machine	Adaption mechanism	Multi-threaded operating system	Dynamic power management	Efficient code distribution management	Finite state machine	Hybrid operating system
Components	Nano hall, task scheduler, power management, and RF handler	Hardware abstraction, synthetic hardware, and high-level software components	Subroutine and message send and receive timer	Both static and dynamic components	Kernel, scheduler, and network stack	Computer specification and power management	Local information and network component	A kernel, a state transition, and call back library	Core services and dynamically loadable services
Scheduling	Linux based	Simple FIFO		Robin robin	Preemptive scheduling		FIFO	FIFO	FIFO poll handler with priority scheduling
Protocol	MAC protocol	Unreliable data link protocol	Beacon less ad hoc protocol	Fixed ad hoc routing protocol	Network stack and MAC protocol			Power management protocol	
Network architecture	Cluster-based	Stack-based threaded	Stack-based	Distributed adaptive operating system	Single-board hardware architecture	Based on a greedy algorithm	Cluster-based		Monolithic binary image
Routing	Ad hoc multiple routing	Single hop routing	Hop by hp injection	Fixed ad hoc routing	Flooding routing				
Execution mode	Multi-threaded	Event-based	Event-based	VM based	Layered multi-threaded based	Event-based	Event-driven model	Event-based	Multi thread-based
Building blocks		Component interference and task	Byte-code interpreter		Comprehensive system APIs and system interaction		Application programming interface		Service layer and service interference
Memory allocation	Dynamic	Static	Dynamic	Dynamic	Dynamic	Dynamic	EEPROM		Dynamic

Table 2. Comparison of Nano Qplus and other operating systems based on their features

Features	Low power mode	Multimodal testing	Real-time guarantee	Dynamic module support	Memory management
Nano Qplus	Yes	Yes	Yes	No	No
Tiny OS	Yes	-	No	No	No
Mate	Yes	-	No	No	No
Magnet OS	-	-	No	No	No
MANTIS	No	-	No	Yes	No
OSPM	-	-	-	-	-
Eyes OS	-	-	No	No	Yes
Sen OS	Yes	-	No	Yes	No
Contiki	-	Yes	No	Yes	No