Comparing Operating Systems in Wireless Sensor Networks: Performance and Efficiency

Luis P. Garcia[1], Maria T. Hernandez[2] & Alejandro V. Lopez[3]
[1]Professor, Department of Electrical Engineering, University of São Paulo, São Paulo, Brazil
[2]Assistant Professor, Department of Electronics Engineering, University of Rio de Janeiro, Brazil
[3]Research Associate, Department of Electrical Engineering, University of Campinas, Brazil

ABSTRACT

Remote Sensor Networks (WSNs) have been the subject of escalated explore over the span of the latest years. WSNs comprise of a substantial number of sensor hubs, and are utilized for different applications, for example, building checking, condition control, untamed life living space observing, woods fire recognition, industry mechanization, military, security, modern process checking and control, front line reconnaissance, activity control, home robotization, and human services. Working framework (OS) bolster for WSNs assumes a focal part in building versatile appropriated applications that are effective and solid. WSNs confront parcel of issues that don't emerge in different kinds of remote systems and figuring situations. Restricted computational assets, control requirements, low unwavering quality and higher thickness of sensor hubs (bits) are some fundamental issues that should be considered while planning or choosing another working framework for execution assessment of remote sensor hubs. We give an arrangement structure that overviews the best in class in WSN Operating Systems (OS). The reason for this paper is to group the current working frameworks as per vital OS includes and propose proper OSs for various classifications of WSN applications, mapping the application necessities and OS highlights. This characterization helps in understanding the differentiating contrasts among existing working frameworks, and establishes a framework to plan a perfect WSN OS. At that point, we look at some current OSes for WSNs, including TinyOS, Contiki, and LiteOS.

Keywords: WSNs, Wireless Networks, Ad-hoc arranges, WSN Operating Systems, TinyOS..

I. INTRODUCTION

WSNs are made out of substantial quantities of modest organized gadgets that speak with each other. Working structures are at the center of the sensor center point building. To the extent

Wireless Sensor Networks,require these things in working framework models: Extremely little impression, to a great degree low framework overhead and amazingly low power utilization. When planning or choosing working frameworks for modest organized sensors, we will likely strip down memory size and framework overhead on the grounds that commonplace sensor gadgets are outfitted with 8-bit microcontrollers, code memory on the request of 100KB and RAM is under 20KB.

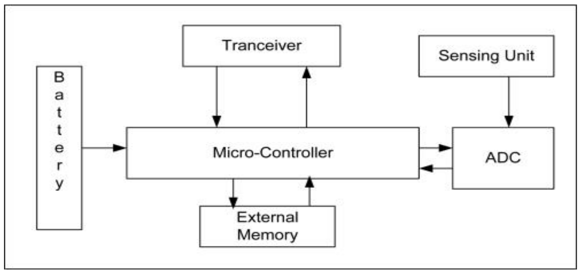

Figure 1: block diagram of a typical node

A remote sensor center point is made out of a scaled down scale controller, handset, clock, memory and easy to cutting edge converter. Figure 1 exhibits the square diagram of an normal center. The basic and most essential resources are essentialness (which is routinely given by a battery) and to a great degree compelled essential memory, with every now and again allows securing only two or three kilobytes. The scaled down scale controller used as a piece of a remote sensor center point works at low repeat appeared differently in relation to standard contemporary taking care of units. These benefit compelled sensors are a noteworthy case of a System on Chip (SoC).Dense organization of sensor hubs in the detecting field and circulated handling through multi-jump correspondence among sensor hubs is required to accomplish high caliber and adaptation to internal failure in WSNs. Application zones for sensors are developing and new applications for sensor systems are rising quickly.

The highlights of OS are Architecture, Programming Model, Scheduling, Memory Management and Protection, Communication Protocols, Resource Sharing, and Support for Real-Time Applications, in both ongoing and non-constant WSN OSs. Prominent and new OSs for WSN are TinyOS, Contiki, MANTIS Operating System, Nano-RK, SOS, EYES/PEEROS, CORMOS, SenOS and LiteOS. We examine some OSes for WSNs, including TinyOS, Contiki, and LiteOS with Architecture, Scheduling, Memory Management and Protection, Communication Protocols, and Resource Sharing.tinyos

TinyOS is an open source, adaptable, segment based, and application-particular working framework intended for sensor systems. TinyOS can bolster simultaneous projects with low memory prerequisites. The OS has an impression that fits in 400 bytes. The TinyOS segment library incorporates arrange conventions, circulated administrations, sensor drivers, and information obtaining devices.

A. Architecture

TinyOS falls under the solid design class. TinyOS utilizes the part show and, as indicated by the necessities of an application, diverse segments are stuck together with the scheduler to create a static picture that keeps running on the bit. A part is a free computational substance that uncovered at least one interfaces. Segments have three computational reflections: orders, occasions, and errands. Instruments for between part communication are commands and events. Tasks are used to express intra-component concurrency.

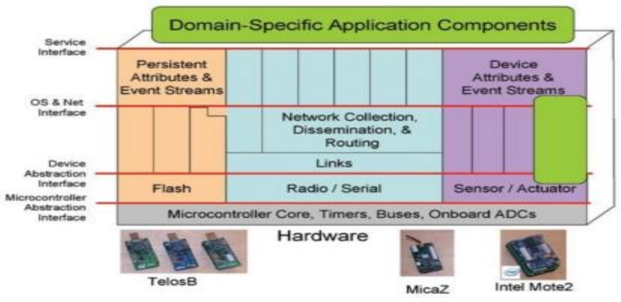
Domain-Specific Application Components
Service Interface
OS & Net Interface
Device Abstraction Interface
Microcontroller Abstraction Interface
Persistent Attributes & Event Streams
Network Collection, Dissemination, & Routing
Device Attributes & Event Streams
Links
Flash
Radio / Serial
Sensor / Actuator
Microcontroller Core, Timers, Buses, Onboard ADCs
Hardware
TelosB
MicaZ
Intel Mote2

***Figure 2*: TinyOS architecture.**

A charge is a demand to play out some administration, while the occasion flags the fulfillment of the administration. TinyOS gives a solitary shared stack and there is no partition between piece space and client space. Figure 2 shows the TinyOS architecture.

B. Scheduling

The center of the TinyOS execution show are undertakings that rushed to fruition in a FIFO way. Since TinyOS underpins just non preemptive planning, undertaking must obey rushed to culmination semantics. Undertakings hurried to fulfillment regarding other assignment yet they are not nuclear as for intrude on handlers, orders, and occasions they summon. Since TinyOS utilizes FIFO planning, inconveniences related with FIFO booking are additionally connected with the TinyOS scheduler. The sit tight time for an errand relies upon the assignment's landing time. FIFO booking can be uncalled for to last assignments particularly when short errands are holding up behind longer ones.

C. Memory management and protection

In sensor hubs, equipment based memory insurance isn't accessible and the assets are rare. Asset limitations require the utilization of hazardous, low level dialects like nesC. In TinyOS rendition 2.1, memory wellbeing is consolidated. The objectives for memory security as given in seem to be: trap all pointer and cluster blunders, give helpful diagnostics, and give recuperation procedures. Usage of memory-safe TinyOS misuses the idea of a Deputy. The Deputy is an asset to asset compiler that guarantees compose and memory security for C code. Code gathered by Deputy depends on a blend of arrange and run-time checks to guarantee memory wellbeing. Safe TinyOS is in reverse perfect with prior rendition of TinyOS. The Safe TinyOS instrument chain embeds registers with the application code to guarantee security at run-time. At the point when a check distinguishes that security is going to be abused, code embedded by Safe TinyOS takes therapeutic activities. TinyOS utilizes a static memory administration approach D. Correspondence PROTOCOL SUPPORT Earlier forms of TinyOS give two multi-jump conventions: dispersal and TYMO.

The spread convention dependably conveys information to each hub in the system. This convention empowers managers to reconfigure inquiries and to reinvent a system. The dispersal convention gives two interfaces : DisseminationValue and DisseminationUpdate. A maker calls DisseminationUpdate. The order DisseminationUpdate.change() ought to be called each time the makers needs to disperse another esteem. Then again, the DisseminationValue interface is accommodated the buyer.

The occasion DisseminationValue.changed() is flagged each time the spread esteem is changed. TYMO is the usage of the DYMO convention, a steering convention for portable specially appointed systems. In TYMO, bundle designs have changed and it has been actualized over the dynamic informing stack.

D. Communication protocol support

Earlier versions of TinyOS provide two multi-hop protocols: dissemination and TYMO. The dissemination protocol reliably delivers data to every node in the network. This protocol enables administrators to reconfigure queries and to reprogram a network. The dissemination protocol provides two interfaces : DisseminationValue and DisseminationUpdate. A producer calls DisseminationUpdate. The command DisseminationUpdate.change() should be called each time the producers wants to disseminate a new value. On the other hand, the DisseminationValue interface is provided for the consumer. The event DisseminationValue.changed() is signaled each time the dissemination value is changed. TYMO is the implementation of the DYMO protocol, a routing protocol for mobile ad hoc networks. In TYMO, packet formats have changed and it has been implemented on top of the active messaging stack.

E. Resource sharing

TinyOS utilizes two systems for overseeing shared assets: Virtualization and Completion Events. A virtualized asset shows up as an autonomous occurrence. i.e., the application utilizes it free of different applications. Assets that can't be virtualized are taken care of through finishing occasions.

The GenericComm correspondence heap of TinyOS is shared among various strings and it can't be virtualized. GenericComm can just send one bundle at any given moment, send tasks of different strings fall flat amid this time. Such shared assets are taken care of through fulfillment occasions that illuminate holding up strings about the finishing of a particular task.

II. CONTIKI

Contiki, is a lightweight open source OS written in C for WSN sensor hubs. Contiki is a profoundly compact OS and it is work around an occasion driven portion. Contiki gives preemptive multitasking that can be utilized at the individual procedure level. A commonplace Contiki design expends 2 kilobytes of RAM and 40 kilobytes of ROM.A full Contiki establishment incorporates highlights like: multitasking portion, preemptive multithreading, proto-strings, TCP/IP organizing, IPv6, a Graphical User Interface, a web program, an individual web server, a basic telnet customer, a screensaver, and virtual system processing.

A. Architecture

The Contiki OS follows the modular architecture. At the kernel level it follows the event driven model, but it provides optional threading facilities to individual processes. The Contiki kernel comprises of a lightweight event scheduler that dispatches events to running processes. Process execution is triggered by events dispatched by the kernel to the processes or by a polling mechanism. The polling mechanism is used to avoid race conditions. Any scheduled event will run to completion, however, event handlers can use internal mechanisms for preemption.

Two sorts of occasions EW upheld by Contiki OS: no concurrent occasions and synchronous occasions. The distinction between the two is that synchronous occasions are dispatched quickly to the objective procedure that makes it be planned. Then again nonconcurring occasions are more similar to conceded strategy calls that are en-lined and dispatched later to the objective procedure.

The surveying instrument utilized as a part of Contiki can be viewed as high-need occasions that are planned in the middle of each offbeat occasion. At the point when a survey is booked, all procedures that actualize a survey handler are brought arranged by their need. Figure 3 shows the block diagram of the Contiki OS architecture.

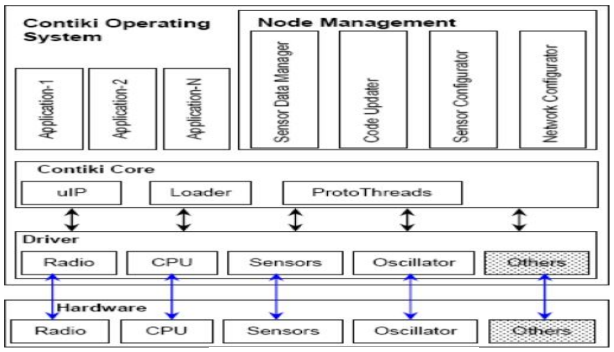

Figure 3:- Contiki OS architecture

B. Scheduling

The surveying system utilized as a part of Contiki can be viewed as high-need occasions that are booked in the middle of each offbeat occasion. At the point when a survey is booked, all procedures that execute a survey handler are brought arranged by their need.

C. Memory management and protection

Contiki bolsters dynamic memory administration and furthermore underpins dynamic connecting of the projects. To make preparations for memory fracture Contiki utilizes a Managed Memory Allocator. The essential duty of the oversaw memory allocator is to keep the allotted memory free from fracture by compacting the memory when squares are free. Hence, a program utilizing the memory allocator module can't make sure that assigned memory remains set up.

For dynamic memory administration, Contiki additionally gives memory square administration capacities . This library gives basic yet capable memory administration capacities for squares of settled length. A memory piece is statically proclaimed utilizing the MEMB() large scale. Memory squares are apportioned from the proclaimed memory by the memb_alloc() work, and are de-designated utilizing the memb_free() work. It is important here that Contiki does not give any memory insurance instrument between various applications.

D. Communication. protocol support

Contiki underpins a rich arrangement of correspondence conventions. In Contiki, an application can utilize the two renditions of IP i.e., IPv4 and IPv6. Contiki gives a usage of uIP, a TCP/IP convention stack for little 8 bit smaller scale controllers. uIP does not require its companions to have an entire convention stack, yet it can speak with peers running a comparable lightweight stack. The uIP execution has the base arrangement of highlights required for a full TCP/IP stack. uIP is composed in C, it can just help one system interface, and it underpins TCP, UDP, ICMP, and IP conventions.

Contiki gives another lightweight layered convention stack, called Rime, for organize based correspondence. Rime gives single jump unicast, single bounce communicate, and multi-bounce correspondence bolster. Rime underpins both best exertion and dependable transmission. In multi-jump correspondence, Rime enables applications to run their own particular steering conventions. Contiki does not bolster multicast. Along these lines Contiki does not give any usage of gathering administration conventions, for example, the Internet Group Management Protocol (ICMP), or Multicast Listener Discovery (MLD) convention. Contiki gives an execution of RPL (IPv6 steering convention for low power and lossy systems by the name ContikiRPL . ContikiRPL works over low power remote connections and lossy electrical cable connections.

E. Resource sharing

Since occasions hurried to finish and Contiki does not permit interfere with handlers to post new occasions, Contiki gives serialized access to all assets.

III. LITEOS

LiteOS is a Unix-like OS for WSN, provide system programmers with a thread-based programming mode, although it provides support to register event handlers using callbacks, a hierarchical file system, support for object-oriented programming in the form of LiteC++, and a Unix-like shell. The footprint of LiteOS is small enough to run on MicaZ nodes having an 8 MHz CPU, 128 bytes of program flash, and 4 Kbytes of RAM.

A. Architecture

LiteOS takes after a particular engineering outline. LiteOS is divided into three subsystems: LiteShell, LiteFS, and the Kernel. LiteShell is a Unix-like shell that offers help for shell orders for record administration, process administration, investigating, and gadgets. A fascinating part of the LiteOS is its LiteShell that lives on a base station or a PC. This influences to help more mind boggling orders as the base station has rich assets. The LiteShell must be utilized when there is a client show on a base station. Some nearby handling is done on the client charge.

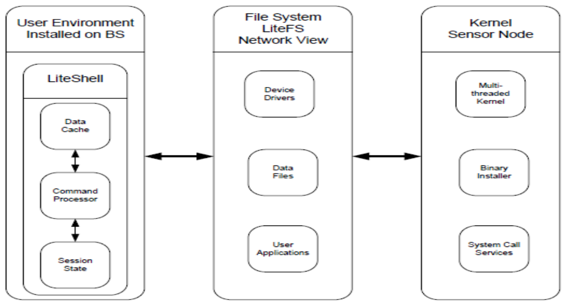

Figure 4:-LiteOS Architecture

B. Scheduling

LiteOS takes after a secluded engineering plan. LiteOS is apportioned into three subsystems: LiteShell, LiteFS, and the Kernel. LiteShell is a Unix-like shell that offers help for shell charges for record administration, process administration, investigating, and gadgets. An intriguing part of the LiteOS is its LiteShell that dwells on a base station or a PC. This influences to help more mind boggling orders as the base station has rich assets. The LiteShell must be utilized when there is a client show on a base station. Some neighborhood handling is done on the client order.

C. Memory management and protection

Inside the kernel, LiteOS underpins dynamic memory assignment using C-like malloc and free capacities. Client applications can utilize these APIs to allot and de-dispense memory at run-time. Dynamic memory develops the other way of the LiteOS stack. The dynamic memory is distributed from the unused territory between the finish of the portion factors and the beginning of the client application memory pieces. This permits modifying the span of dynamic memory as required by the application.

The LiteOS bit assembles independently from the application, along these lines the address space isn't shared between the piece and the application. So also, every client application has its different address space. Procedures and Kernel memory wellbeing is upheld through partitioned address spaces.

D. Communication protocol support

LiteOS gives correspondence bolster as documents. LiteOS makes a record comparing to every gadget on the sensor hub. So also, it makes a document comparing to the radio interface. At whatever point there is a few information that should be sent, the information is put into the radio document and is a short time later remotely transmitted. In a similar way, at whatever point a few information lands at the hub it is put in the radio document and is conveyed to the corresponding application using the port number present in the data

At the network layer LiteOS supports geographical forwarding. Each node contains a table that can only hold 12 entries. This routing protocol is supported in LiteOS version 0.3.

E. Resource sharing

LiteOS propose the utilization of APIs accommodated synchronization at whatever point a string needs to get to assets that are shared by various strings. The LiteOS documentation does not give any detail on how framework assets are shared among numerous executing strings.

IV. COMPARATIVE ANALYSIS

We summarize our survey of OSs for WSNs. Table 1 summarizes the previous discussions based on the common features: OS Architecture, Scheduling, Memory Management and Protection, and Communication Protocol Support, Resource Sharing. Some OSs provide support for priority scheduling while many others do not even provide support for this. It can likewise been seen that early OSs for WSNs stressed on utilizing an occasion driven programming worldview. Be that as it may, as software engineers are more acquainted with threading based programming worldview, contemporary OSs for WSNs bolster the threading based programming model.

Table 1 : Comparative Study

Features / OS	TinyOS	Contiki	LiteOS
Architecture	Monolithic (Primarily Event Driven, Support For TOS Threads)	Modular (Protothreads and Events)	Moular (Threads and Events)
Scheduling	FIFO	Events are fired as they occurs. Interrupts execute with respect to priority.	Priority Based Round Robin Scheduling
Memory Management and Protection	Static Memory Management with Memory Protection.	Dynamic Memory Management and Linking. No Process Address Space Protection.	Dynamic Memory Management and it provides Memory Protection to Processes.
Communication Protocol Support	Active Message.	*u*IP and Rime.	File Based Communication.
Resource Sharing	Virtualization and Completion Events.	Serialized Access.	Through Synchronization Primitives.

V. CONCLUSION

THIS PAPER HELPS TO UNDERSTAND THE CHARACTERISTICS OF POPULAR OSs FOR WSN IN PARTICULAR AND EMBEDDED DEVICES IN GENERAL. DESIGN STRATEGIES FOR VARIOUS COMPONENTS OF AN OS FOR WSN HAVE BEEN EXPLAINED AND INVESTIGATED, ALONG WITH THEIR RELATIVE PROS AND CONS. TARGET APPLICATION AREAS OF DIFFERENT WSN OSs HAVE BEEN POINTED OUT. WE BELIEVE THAT THE PROS AND CONS OF DIFFERENT DESIGN STRATEGIES PRESENTED HERE WILL MOTIVATE RESEARCHERS TO DESIGN MORE ROBUST OSs FOR WSNs. MOREOVER, THIS COMPARISON WILL HELP THE APPLICATION AND NETWORK DESIGNER TO SELECT AN APPROPRIATE OS FOR THEIR WSN APPLICATIONS

REFERENCES.

[1] Akyildiz, I.F.; Su, W.; Sankarasubramaniam, Y.; Cayirci, E. Wireless Sensor Networks: A Survey. Comput. Netw. 2002, 38, 393-422.

[2] Reddy, V.; Kumar, P.; Janakiram, D.; Kumar, G.A. Operating Systems for Wireless Sensor Networks: A Survey. Int. J. Sens. Netw. 2009, 5, 236-255.

[3] Levis, P.; Madden, S.; Polastre, J.; Szewczyk, R.; Whitehouse, K.; Woo, A.; Gay, D.; Hill, J.; Welsh, M.; Brewer, E.; Culler, D. Tinyos: An Operating System for Sensor Networks; Available online: http://dx.doi.org/10.1007/3-540-27139-2_7 (accessed on 17 April 2011).

[4] Contiki Documentation; Available online: http://www.sics.se/~adam/contiki/docs/ (accessed on accessed on 17 April 2011).

[5] Casado, L.; Tsigas, P. ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System. Lect. Note. Comput. Sci. 2009, 5838, 133-147.

[6] Cao, Q.; Abdelzaher, T.; Stankovic, J.; He, T. The LiteOS Operating System: Towards Unix Like Abstraction for Wireless Sensor Networks. In Proceedings of the 7th International Conference on Information Processing in Sensor Networks (IPSN 2008), St. Louis, MO, USA, 22–24 April 2008.

[7] Dunkels, A.; Gronvall, B.; Voigt, T. Contiki a Lightweight and Flexible Operating System for Tiny Networked Sensors. In Proceedings of the 9th Annual IEEE International Conference on Local Computer Networks, Washington, DC, USA, October 2004; pp. 455-462.

[8] Dwivedi, A.K.; Tiwari, M.K.; Vyas, O.P. Operating Systems for Tiny Networked Sensors: A Survey. IJRTE 2009, 1, 152-157.

[9] Montenegro, G.; Kushalnagar, N.; Hui, J.; Culler, D. Transmission of Ipv6 Packets over IEEE 802.15.4 Networks, RFC 4944; Available online: http://tools.ietf.org/html/rfc4944 (accessed on 17 April 2011).

[10] Muhammad Omer Farooq and Thomas Kunz, Operating Systems for Wireless Sensor Networks: A Survey , Sensors 2011, 11, 5900-5930; doi:10.3390/s110605900, ISSN 1424-8220 , www.mdpi.com/journal/sensors, Published: 31 May 2011