International Journal of Smart Sensor and Adhoc Network

Volume 2 | Issue 4 Article 13

July 2012

# DESIGN ISSUES AND CLASSIFICATION OF WSNS OPERATING SYSTEMS

ANIL KUMAR SHARMA
*Department of I.T. and Computer Application, Dr. C.V.Raman University, Kota, Bilaspur, Chhattisgarh, India*, sharmaanil.mail@gmail.com

SURENDRA KUMAR PATEL
*Department of I.T. and Computer Application, Dr. C.V.Raman University, Kota, Bilaspur, Chhattisgarh, India*, surendrapatelit2004@gmail.com

GUPTESHWAR GUPTA
*Department of Mathematics & Information Technology, Govt. N.P.G. College of Science, Raipur, Chhattisgarh, India*, gupteshwar_gupta@yahoo.co.in

# DESIGN ISSUES AND CLASSIFICATION OF WSNS OPERATING SYSTEMS

**ANIL KUMAR SHARMA[1], SURENDRA KUMAR PATEL[2], & GUPTESHWAR GUPTA[3]**

[1&2]Department of I.T. and Computer Application, Dr. C.V.Raman University, Kota, Bilaspur, Chhattisgarh, India
[3]Department of Mathematics & Information Technology, Govt. N.P.G. College of Science, Raipur, Chhattisgarh, India
E-mail: sharmaanil.mail@gmail.com, surendrapatelit2004@gmail.com, gupteshwar_gupta@yahoo.co.in

**Abstract:-** Wireless Sensor Networks is an emerging area of research. Wireless Sensor networks (WSNs) face lot of problems that do not arise in other types of wireless networks and computing environments. Limited computational resources, power constraints, low reliability and higher density of sensor nodes (motes) are just some basic problems that have to be considered when designing or selecting a new operating system in order to evaluate the performance of wireless sensor nodes (motes). In this paper we focused on design issues, challenges and classification of operating systems for WSNs.

***Keywords:*** *WSN, OS, Design, Scheduling, Protocols*

## I. INTRODUCTION

WSNs are composed of large numbers of tiny networked devices that communicate untethered. Operating systems are at the heart of the sensor node architecture. In terms of Wireless Sensor Networks we need these things in operating system architectures: Extremely small footprint, extremely low system overhead and extremely low power consumption. When designing or selecting operating systems for tiny networked sensors, our goal is to strip down memory size and system overhead because typical sensor devices [1, 2, 3] are equipped with 8-bit microcontrollers, code memory on the order of 100KB and RAM is less than 20KB. After evaluating various research papers at present (till Dec, 2008) we have identified 37 operating systems running on different wireless sensor nodes (motes).

Tiny sensor nodes/motes collectively form Wireless Sensor Network (WSN). A WSN typically consists of hundreds or thousands of sensor nodes. These nodes have the capability to communicate with each other using multi-hop communication. Wireless sensor networks hold great promise as an enabling technology for a variety of applications, including data collection and event detection. [8]

The basic functionality of an operating system for tiny sensors/motes is to hide the low-level details of the sensors by providing a clear interface to the external world. Processor management, memory management, device management, scheduling policies, multi-threading, and multitasking are some services to be provided by an operating system. In addition to these services the operating system should also provide services like support for dynamic loading and unloading of modules, providing proper concurrency mechanisms, Application Programming Interface (API) to access underlying hardware, and enforce proper power management policies. In WSN core kernel of the operating system sits at each individual node. On top of it, middleware and applications run as interacting modules across nodes. Due to the significance of an operating system for WSNs with this paper we explore the challenges and design issues that may affect the design of an operating system for WSNs. We first discuss the issues and challenges involved in designing an OS for WSNs. Secondly we discuss on these features of an OS for tiny networked sensors: architecture, reprogramming, execution model, scheduling, power management, simulation support, and portability.

## II. PROBLEM FORMATION

After analyzing several recently deployed WSNs and operating systems for tiny networked sensors we identify that three OS features – virtual memory, preemptive priority scheduling, and OS safety – will significantly improve WSN systems. However, it is impossible to implement these features with traditional OS design techniques, due to the current situation of embedded H/W platforms. Moreover, we envision that the H/W constraints will still exist for a long time for energy and cost efficient miniaturized computing devices.

## III. DESIGN ISSUES AND CHALLENGES

WSN operates at two levels. One is at the network level and the other is at node level.[6] Network level interests are connectivity, routing, communication channel characteristics, protocols etc and node level interests are hardware, radio, CPU, sensors and limited energy. At a higher level OS for WSN can also be classified as node-level (local) and network-level (distributed). The important issues related to node-level are limited resource management; concurrency handling, power management and

memory management where as issues related to both are inter-node communication, failure handling, heterogeneity and scalability. This section discusses the important issues (of both node and network-level) to be considered while designing an operating system for WSN. These issues discuss the challenges and motivate the design requirements of an operating system needed for WSN [5][7].

- Restricted Computational Resources
    - Power
    - Memory
    - Bandwidth Battery Power
    - Processing
- Portability
- Customizability
- Multitasking
- Network Dynamics
- Distributed Nature
    - Inter-Node Communication
    - Failure Handling and Disconnection
    - Scalability

## IV. DESIGN CHARACTERISTICS

The following are the important design characteristics to be considered while designing an operating system for Tiny Networked sensors [2].

### *A. Flexible Architecture*

Architecture of the kernel influences the way it provides services. Two things that are affected by the OS architecture are: Run-time reconfigurability of the services, and Size of the core kernel.

Facility of adding kernel services or updating them depends entirely on the architecture of the operating system. Size of the core kernel is another factor that depends on the architecture. If the architecture allows to bundle all the required services together into a single system image, then size of the core kernel increases. All the services that constitute core kernel may not be required all the time for the applications running. On the other hand such architecture can also support building application specific single system image kernel that binds only the required services for an application. Even though this reduces the size of the kernel, it does not allow running multiple applications. Moreover such architecture makes entire image to be replaced if there are any changes to the kernel or application.If the architecture allows gluing services at run-time, this reduces the core size of the kernel, that is provides flexibility in updating or replacing the corresponding service, which is modified or changed without replacing the entire image of the kernel.

### *B. Efficient Execution Model*

The execution model provides the abstraction of computational unit and defines services like synchronization, communication, and scheduling. These abstractions are used by the programmer for developing applications. Communication service defines the way the computational units communicate. They communicate to exchange data, delegation of functionalities and signaling. While communicating there can be data that is shared. Accessing shared data requires proper synchronization mechanisms to avoid race conditions. At a given instance application might be required to perform concurrency intensive tasks. The context switching among the tasks is required in order to avoid blocking of the tasks from execution. Flexible computational unit aids in having flexible architecture for the system. Scheduling of computational units is crucial in the case of mission critical applications, where execution of them after their deadlines will lead to catastrophic situations.

At a given instance application might be doing multiple jobs. This requires proper scheduling of the processor to execute those jobs. Scheduling defines the order in which the computational unit has to gain access to processor.

### *C. Clear Application Programming Interface (API)*

APIs play vital role in providing clear separation between the low level node functionalities and the application program. Operating system should provide comprehensive set of APIs to interact with system and it's I/O. This helps user in flexibly developing applications without considering low level functionalities of the sensor node hardware. The system API may include

- Networking API
- Sensor data reading API
- Memory manipulation APIs
- Power management API
- Task management APIs

These APIs allow the application developer to build applications and use the available resources efficiently. APIs related to memory access are important to reconfigure the software running on the sensor node dynamically. APIs related to posting of events/tasks and setting the delays associated with the tasks gives the programmer flexibility in scheduling them.

### *D. Reprogramming*

Reprogramming is a mandatory feature for OS and it simplifies the management of software in sensor nodes. It is the process of dynamically updating the software running on the sensor nodes. Reprogramming got much attention in WSN because of the inaccessibility of the sensor nodes after deployment and due to the presence of large number of them in the network. Without reprogramming, it is difficult to add, modify or delete the software from the running system in WSN.

Code is distributed over the air using code dissemination protocols. These protocols deal with the splitting and compressing the code to be sent for updating the software on the nodes. Communication in these protocols is either single-hop or multi-hop. In single-hop method the nodes are directly connected to the base station either through wired or wireless and then reprogrammed. In multi-hop communication method, the code is sent hop-by-hop in the network. After the reception of code at the node, it has to either add or update the existing software running on it. This requires efficient memory management mechanisms. For reprogramming to be successful at any time in the running system the code should be relocatable. Relocatable code is position independent that can be run in any location of the memory. This is an important requirement for reprogramming as the modified code has to be loaded and run in any part of the available free memory. The underlying execution-environment plays a vital role in facilitating reconfigurability. Operating system should allocate memory dynamically to facilitate loading of software components at run time. It should also provide inter component communication which helps in dynamically linking the components.

*E. Resource Management*

One of the fundamental tasks of an operating system is to manage the system resources efficiently. Resources available in a typical sensor node are processor, program memory, battery, and sensors etc. Efficient use of processor involves using a scheduler with optimal scheduling policy. Usage of memory involves memory protection, dynamic memory allocation, etc. Battery should be treated as a special resource. Sleep modes help in power management of battery. Managing sensors include controlling sensing rate. It is the responsibility of the operating system to follow necessary mechanisms in order to consume the power in optimized way in turn prolonging the life of the WSN [2].

*F. Real-time Nature*

This is the optional design characteristic and is application specific. Real-time applications of WSN can be classified into periodic and a periodic, critical and non critical. The classical example for the periodic task is monitoring application, where the data is read from the environment or habitat in a periodic manner. Target tracking or fire explosion are the examples for a periodic tasks. These examples again can be classified into critical and non-critical tasks. This classification is based on whether the execution of the tasks is in stipulated time or not. Satisfying real-time constraints is also one of the key requirements for critical applications in WSN. For example in applications like fire detection in nuclear reactors, preventive action should be taken within hard deadlines. Real-time constraints of the applications can be satisfied with the help of a real-time scheduler by following with proper scheduling policy.

## V. CLASSIFICATIONS FOR WSN OPERATING SYSTEMS

Architecture, execution model, reprogramming, scheduling, and power management are the important design features that forms basis for our classification. Below is an overview of the design features:

*A. Architecture*

Architecture of the kernel influences the way it provides services. There are mainly three kinds of architectures in the literature. They are:

- Monolithic

Application + Necessary OS components = Single system image

- Modular

Application and OS is built as a set of interacting modules

- Virtual Machine

Application as a set of static and dynamic components = Network wide single system image

There is a trade-off between performance and the flexibility depending on the architecture that is chosen. Monolithic kernel always forms a single system image for the node. This is not preferred if there are frequent changes in the requirements of the application, which might cause the reconfiguration of existing software on the node. Modular architecture fits well if there is a requirement for reconfiguration. It simplifies the problems of code maintenance and modification. But there is a overhead in loading and unloading modules dynamically if the modules are position dependent. This overhead also includes allocation of contiguous memory for a single module. Virtual machine architecture considers the whole network of nodes as single entity. This gives flexibility in developing applications. As application is composed of instructions specific to virtual machine, reprogramming is easy.

*B. Execution Model*

The kernel transitions and DVM are major sources of overhead impacting the execution speed of an application. [4] Execution model drives the performance of the operating system for WSN. Execution model or programming model used by most of the embedded systems is event-based. But thread-based programming model can also be used at some cost. There exists trade-offs among these two. The other execution models that are quite often used are state based, object based and data centric. State based approach is similar to FSM and offers many advantages like concurrency, reactivity and reconfigurability. Every application is composed of states and responds to events which results in state transitions based on different inputs.

Reconfigurability can be achieved simply by changing the state transition table associated with the application.

*C. Reprogramming*

Reconfigurability got much attention in WSNs because of the inaccessibility of the sensor nodes after being deployed in large number. In WSN literature, reconfigurability is the ability to add/ delete/ modify the software module running on the node. This functionality is useful when there is a need to tune the application software according to the user requirements and make the application adaptable. For example, at run time it might require to change filter condition of an aggregation mechanism or add a fault tolerant service to make routing protocol robust. Reconfigurability also enables to deploy heterogeneous nodes i.e. different nodes running different software modules. Reconfigurability is accomplished using code distribution protocols.

Reprogramming can be done at different granularities ranging from tuning a variable to changing the entire image of software on the node. Application level reprogramming replaces the entire application image. Modular level or component level reprogramming replaces/updates the module or component of an application. Instruction level and variable level gives the flexibility in changing instructions and tuning parameters of the application respectively.

*D. Scheduling*

Real-time systems can be classified into periodic and a periodic, critical and non critical. Most of the applications of WSN can be classified into the mentioned categories. The classical example for the periodic task is monitoring application, where the data is read from the environment or habitat in a periodic manner. Target tracking or fire explosion are the examples for a periodic tasks. These examples again can be classified into critical and non-critical tasks. This classification is based on whether the execution of the tasks is in stipulated time or not. Satisfying real-time constraints is also one of the key requirements for critical applications in WSN. For example in applications like fire detection in nuclear reactors, preventive action should be taken within hard deadlines. Real-time constraints of the applications can be satisfied with the help of a real-time scheduler by following a proper scheduling policy.

*E. Power Management*

Power management interfaces provided by an operating system can be used to enforce an optimal way of utilizing energy. Conserving power involves accessing/controlling components on the sensor node. Power management interfaces are used to control/access these components. The components which expose power management interfaces are processor, radio and battery. The components that can be controlled to conserve power are processor and radio.

*F. Miscellaneous*

- Simulation Support

The applications can be tested on the simulation environment provided by the operating system, before they are actually deployed in the network. The same code that is tested on simulation environment should be able to run on sensor nodes.

- Portability

Portability of the operating system to different hardware platforms is important in order to cope up with the upcoming hardware platforms.

## VI. CONCLUSION

We have presented design issues, challenges, characteristics and classification of operating systems for WSN's. The objectives of this paper were to provide background knowledge of problem formation on different operating systems for Tiny Networked Sensors. Important points of the realization phase such as the WSN operating system architecture, execution model and reprogramming are discussed.

The survey explores the existing design approaches as well as new requirements to be considered in the future applications of OS for wireless sensor networks. Finally, this paper includes helps the researchers in understanding various aspects while building WSN system software in particular to OS. The issues presented here Motivates the design principles to be considered while designing an OS for WSN.

## REFERENCES

[1] Antônio Augusto Fröhlich and Lucas Francisco Wanner: "Operating System Support for Wireless Sensor Networks". Journal of Computer Science 4 (4): 272-281, 2008. ISSN 1549-3636

[2] Adi Mallikarjuna Reddy, V AVU Phani Kumar, D Janakiram and G Ashok Kumar: "Operating Systems for Wireless Sensor Networks: A Survey-Technical Report", IIT Madras, Chennai, India, May 3, 2007.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister: "A System Architecture for Networked Sensors", ASPLOS-IX 11/00 Cambridge, MA, USA 2000.

[4] Lin Gu , John A. Stankovic "t-kernel: Providing Reliable OS Support to Wireless Sensor Networks", SenSys'06, November 1–3, 2006, Boulder, Colorado, USA.

[5] Gregory J. Pottie, "Wireless sensor networks," in IEEE Information Theory Workshop Proceedings, June 1998.

[6] Kayvan Atefi,, Mohammad Sadeghi,, Arash Atefi "Real-Time Scheduling Strategy for Wireless Sensor Networks O.S", International Journal of Distributed and Parallel Systems (IJDPS) Vol.2, No.6, November 2011.

[7] Lalit Saraswat,Pankaj Singh Yadav "A Comparative Analysis of Wireless Sensor Network Operating Systems", Proceedings of the 5th National Conference; INDIACom-

2011 Computing For Nation Development, March 10 – 11, 2011 Bharati Vidyapeeth‟s Institute of Computer Applications and Management, New Delhi

[8] Prabal Dutta,Mike Grimmer, Anish Arora, Steven Bibyk, David Culler “Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events”

◈◈◈