Operating Systems for Wireless Sensor Networks: A Survey Technical Report

Adi Mallikarjuna Reddy V AVU Phani Kumar, D Janakiram, and G Ashok Kumar
Distributed and Object Systems Lab
Department of Computer Science and Engineering
Indian Institute of Technology Madras, Chennai, India - 600036
{adi, phani, d.janakiram, ashok}@cs.iitm.ernet.in

May 3, 2007

Abstract

The design of operating system for Wireless Sensor Network (WSN) deviates from traditional operating system design due to significant and specific characteristics like constrained resources, high dynamics and inaccessible deployment. We provide a classification framework that surveys the state of the art in WSN operating systems (OS). The purpose of this survey is two-fold one is to classify existing operating systems according to important OS features and and the other is to suggest suitable OSs for different categories of WSN applications mapping the application requirements and OS features.

Architecture, Execution Model, Reprogramming, Scheduling and Power Management are the important OS features that are chosen to classify the existing WSN operating systems. The classification helps in understanding the contrasting differences of existing operating systems and lays the foundation for designing an ideal operating system. To help the application developer in choosing the right OS, based on the application requirement, we also classified existing WSN applications. This classification gives insight in choosing the best suitable operating systems that fits for a category of application.

1 Introduction

A wireless sensor node is a good example for a System on Chip (SoC) that has communication, computation, sensing and storage capabilities. These miniaturized nodes have stringent constraints in terms of available resources like processing power, battery power, program memory, available bandwidth. Figure 1 shows schematic diagram of sensor node components. Basically, each node comprises of a micro-controller, power source, Radio Frequency (RF) transceiver, external memory, and sensors. These sensor nodes collectively form a Wireless Sensor Network (WSN), which are used in wide variety of applications now a days. 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. Typical applications of these WSN include but not limited to monitoring, tracking, and controlling.

The basic functionality of an operating system is to hide the low-level details of the sensor node by providing a clear interface to the external world. Processor management, memory management, device management, scheduling policies, multi-threading, and multitasking are some of the low level services to be provided by an operating system. In addition to the services mentioned above, 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. Though some of

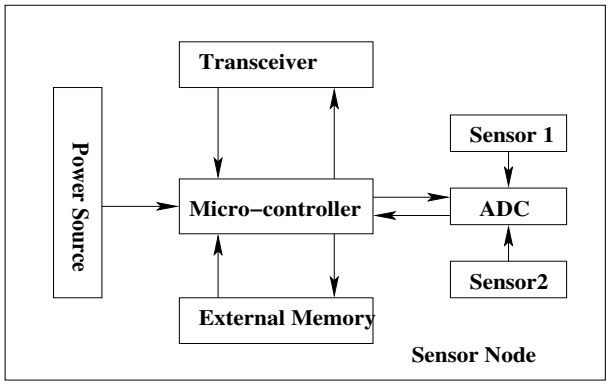

Figure 1: Sensor Node Architecture

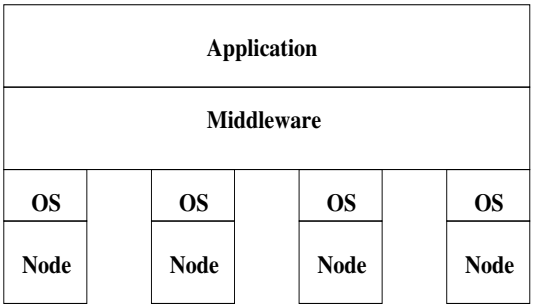

Figure 2: Software Layers

these are similar to the services provided by traditional operating systems, the realization of those services in WSN is a non-trivial problem, due to the constraints on the resource capabilities. Hence a suitable operating system is required for WSN to provide these functionalities to facilitate the user in writing applications easily with little knowledge of the low-level hardware details. Figure 2 depicts, where operating system stands in the software layers of the WSN. Middleware and application layers are distributed across the nodes. 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 and the availability of a significant body of literature on it, a detailed survey becomes necessary and useful at this stage. Although there are papers that surveys the characteristics, applications, and communication protocols of WSNs [20][79] [85] [21], prior to this there are no studies that survey the operating systems of WSN to the best of authors knowledge.

Our work is a study of issues and challenges, describing and comparing the different approaches. We discuss the challenges and design issues that may affect the design of an operating system for WSNs. We provide a *classification framework* that compares the existing operating systems according to the core OS features identified. The core OS features that constitute *classification framework* are architecture, reprogramming, execution model and scheduling. Miscellaneous features like power management, simulation support, and portability have also been considered. These features are used to compare and evaluate the existing operating systems. Applications of WSN are also investigated to categorise and highlight the important characteristics of them. We evaluate and match the important requirements of applications with that of operating systems. This provides an outlook on desired features of an ideal operating system for WSN applications.

The scope of this survey is limited to the study of existing WSN OSs which are well known and/or having interesting features to be considered as a WSN operating system.

The rest of this paper is organized as follows. In Section 2, we discuss the issues and challenges involved in designing an OS for WSNs. Section 3 describes design requirements of an operating system for WSN. A classification framework and a comprehensive survey of existing operating systems against this framework is presented in Sections 4, 5 respectively. Evaluation of different operating systems is tabulated in section 6. Section 7 explores the applications of WSN and highlights the important characteristics of these applications. In Section 8, a summary of open issues are discussed. We conclude with final remarks in Section 9.

2 Design Issues and Challenges

WSN operates at two levels [64]. One is at the network level and the other is at node level. 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.

2.1 Restricted Resources

A typical sensor node shown in figure 1 is constrained by the resources available to it. It is constrained by limited battery power, processing capability, memory and bandwidth. [56] evaluates the existing hardware platforms and presents a nice comparison report.

2.1.1 Battery Power

Power consumption is crucial to the life span of WSN based applications. Most of the applications in WSN are long lived ranging from days to years [67]. So a typical node with a limited power supply has to live mostly for months to years. Unlike conventional systems where the power is not at all a constraint factor in building the system, sensor nodes have to consider power as one of the available resources like processor and memory.

The main source of power consumption is communication when compared to computation and sensing [42]. The energy cost of transmitting a bit of data over RF channel is equivalent to executing thousands of instructions by the processor on the node. Reading and writing to flash storage also consumes significant amount of energy. Hence, run time environment should also take into account the flash energy consumption overhead while loading and unloading of modules into program memory.

It is the responsibility of the operating system to provide necessary mechanisms in order to consume the power in optimized way to prolong the life of the WSN. Periodic sleeping of sensor nodes is one of the mechanisms to conserve power. Sensor nodes operate in three sleep modes, idle, power down and power save, in order to conserve the energy. In idle mode the processor alone shuts off, power down mode shuts off everything except the watch dog timer and interrupt logic necessary to wake up the node, power save mode is similar to power down mode except that it keeps the timer running [41] [42].

The operating system has to provide a clean and concise abstraction for power management. The power management routines provided by the OS are used in defining sleep rates and duty cycle periods desired by the application. These can be changed at run time based on application

Standard	Data rate	Range
CC1000	39kbps	300m
802.15.4 (PAN)	256kbps	100m
Bluetooth	up to 3Mbps	1m to 100m
802.11 (Wi-Fi)	54Mbps	45m to 90m

Table 1: Wireless Standards

requirements [25]. The power-aware strategy should be used to design various policies of the operating system. For example the scheduling policy should be power-aware and schedule the tasks to processor.

2.1.2 Processing Power

Sensor nodes will have a processing power in the order of a few MIPS [42]. Computation intensive operations should be properly scheduled; otherwise high priority tasks get delayed/starved. Computation models like event driven will follow run-to-completion model [25]. This takes more processor time if the task is running for long time and preventing other jobs to wait for longer time irrespective of their priorities. Hence operating system should properly schedule the processor according to the priority of jobs.

2.1.3 Memory

The current generation of micro-controllers family such as Mica [84], its successors and some micro-controllers (e.g. nymph, EYES etc) specific to various research projects have nearly 128kbytes of program memory. One of the main constraints for the developer is this available program memory and operating system developed for WSN should fit within this memory. The system software such as operating system, virtual machine, middleware, and application algorithms have to fit into this memory. Optimal usage of this memory should start from lower level (i.e. Operating System). Sensor nodes also have non volatile external data storage mechanism (e.g. EEPROM or flash memory). This is similar to secondary storage in traditional systems. Every program module is stored here, before it is actually loaded into program memory for execution.

2.1.4 Bandwidth

A typical sensor node uses RF channel to communicate with other sensor nodes in the network. ZigBee is the emerging standard to define the communication protocol stack based on the existing physical and data-link layers of IEEE 802.15.4 Personal Area Network (PAN) standard. Data rate supported by PANs is 256kbps. Whereas Bluetooth standard supports data rate up to 3Mbps. CC1000 is another standard that has been widely used in sensor networks. Its data rate is around 39kbps. Rarely used wireless standard in WSN is IEEE 802.11 (Wi-Fi), whose data rate is almost 54Mbps. PAN consumes low-power and now a days it is widely used in sensor nodes. This operates in different modes conserving power. Bluetooth RF transceiver consumes more power to switch between modes [18] [75].

As per as authors knowledge 8MHz clock speed processor, 10k of RAM and 128k of flash memory sensor nodes are available currently. Nodes may last at most 945 days with two AA batteries [17]. Table 1 summarizes the wireless communication standards available for WSN.

2.2 Portability

The hardware platforms in WSN are evolving day-by-day. Portability is an important issue to be considered as every one is working on their customized hardware platforms. Portability is one of the

main concerns for the developer to make the software work on different hardware platforms. The operating system should be written in such a way that it is easily portable to different hardware platforms with minimal changes.

2.3 Customisability

Applications in WSN are spread over different disciplines. Specific applications of WSN include but not limited to monitoring environment, surveillance, target tracking etc. Survey of some of these applications can be found in [67] [85]. Most of the software platforms developed for WSN are application specific. Different applications demand different requirements from operating system. These requirements may be reconfigurability[1], real-time guarantees. The design of OS should be in such a way that it should be easily customizable and extensible to various applications.

2.4 Multitasking

At a given point of time, nodes in the WSN could be doing more than one task. For example, consider a typical application where in the sensed data from the environment is collected, aggregated based on some filtering conditions, encrypted/decrypted and passed it towards the sink node through other nodes. In this application the sensor node has to do the following tasks at a given point of time:

1. sense the data
2. collect data from other neighborhood sensor nodes
3. aggregate the data based on the certain conditions provided
4. encrypt/decrypt the data before processing/forwarding
5. route the data to the sink node

A sensor node may be doing more than one task/operation, listed above, at a given point of time. Some of these are concurrent operations and should be handled carefully; Otherwise, the important tasks/operations to be performed are ignored. Due to *run-to-completion* processing of tasks in WSN, the other tasks to be performed are delayed. It is not feasible to buffer all the incoming or outgoing messages due to low capacity of buffer. Physical parallelism provided by micro-controllers is limited, and the context switch overhead involved in switching the task is very important. Asynchronous behavior of task handlers in operating system is feasibly good to handle the concurrent operations. Operating system should handle these situations by providing a good execution model and good mechanism to switch between the tasks easily.

2.5 Network Dynamics

Mobility, failure of communication channels/nodes constitutes the dynamics in WSN. Topologies are more prone to changes due to these dynamics which may result in network partitions. Link failures and the interferences in the RF communication channel deviates the behavior of the WSN from its normal operation.

Operating system should adapt the application according to the context of different dynamics of the environment. This helps in providing transparency from network dynamics to the application.

[1] reconfigurability and reprogramming are used interchangeably although a minor difference exists

2.6 Distributed Nature

There is a clear distinction between the services that should be supported by middleware and OS in traditional systems. This is masked in WSN due to cross layer interaction support which is a prominent feature for these kind of systems.

Sensor nodes in WSN are loosely coupled and some times deployed across a large geographical area. The scale of the network some times is in the order of thousands of sensor nodes. Each individual node has its own processing power, system software to run and the co-operation among the nodes happen through exchange of messages.

In a distributed environment, the goal of an operating system should be to manage the different nodes spread over the region and make them appear as a single virtual entity. This involves providing communicational transparency, failure transparency, heterogeneity and scalability support for the application.

2.6.1 Inter-Node Communication

Application developer composes his program with a single entity view of the system. This composition involves various interactions that appears to be among components on the same node. Physically these components are distributed across the network. The system has to optimally place these components so that the communication cost is minimized. Other issues related to communication such as low bandwidth, link failures and inaccessibility of the nodes should be masked from the application developer.

2.6.2 Failure Handling and Disconnection

Nodes in the network can fail or disconnect due to physical damage, harsh environmental interferences, energy depletion, node mobility and communication failures. This may result in topological changes. The failure or disconnection of nodes in the network should not affect the running applications. The operating system design should be robust to handle this issue.

2.6.3 Heterogeneity

Heterogeneity in the network arrives due to varying level of node capabilities. This causes different nodes to be present in the network with different capabilities. These capabilities can be in terms of memory, sensing modality or residual energy, software components residing at the node. Many of the practical sensor networks are heterogeneous [37] in their sensing capability. Due to this, they can incur low deployment cost. System should mask heterogeneity to the user by optimally distributing the load on the nodes according to their capabilities.

2.6.4 Scalability

Scalability here refers to the size of the network [81]. As the system is composed of large number of nodes, the system algorithms should work with an acceptable performance degradation with increase in the number of nodes.

In WSN there are subtle differences related to some issues in designing middleware [38] and operating system. Some of the above issues seems to look like middleware issues but virtual machine approaches [23] and distributed operating systems [23] [26] in WSN did concerned about them in designing.

The above issues should be considered while designing a operating system for sensor networks. Most of them are purely specific to sensor networks and the designer should pay more attention while designing an operating system for WSN.

3 Design Characteristics

The following are the important design characteristics to be considered while designing an operating system for WSN.

3.1 Flexible Architecture

Architecture of the kernel influences the way it provides services. Two things that are affected by the OS architecture are : (1) run-time reconfigurability of the services, and (2) 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 an 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 to run multiple applications. Moreover such an architecture makes entire image to be replaced if there are any changes to the kernel or application.

If the architecture allows to glue services at run-time, this reduces the core size of the kernel. Provides flexibility in updating or replacing the corresponding service, which is modified or changed without replacing the entire image of the kernel.

3.2 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 signalling. 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.

3.3 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 its 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
  - Send and receive operations
- Sensor data reading API
- Memory manipulation APIs

  - Load and store operations

- Power management API
  - Sleep, reading energy level
- Task management APIs [33]
  - Set delay, set priority, post

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.

3.4 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.

The code is distributed over the air using code dissemination protocols [77] [45] [28]. 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 an 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.

3.5 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.

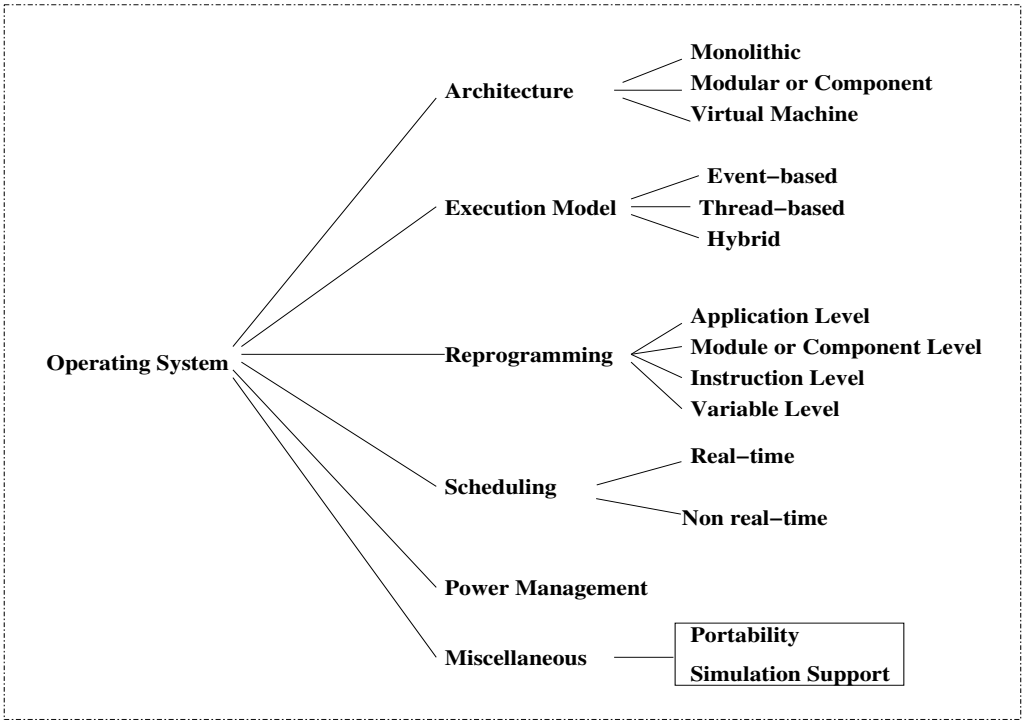

Figure 3: Classification Framework for WSN Operating Systems

3.6 Real-time Nature

This is the optional design characteristic and is application specific. Real-time applications of WSN can be classified into periodic and aperiodic, 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 aperiodic 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 [55][48][70] .

4 Classification Framework for WSN Operating Systems

We have chosen architecture, execution model, reprogramming, scheduling, and power management as the important design features that forms basis for our classification framework. Different types of each feature is depicted in the figure 3. Based on each feature an operating system is classified. We call this as classification framework rather than classification, because the classification is according to multiple features rather than a single feature of operating system. Miscellaneous features shown in the figure are explained for each operating system. These features are simulation support and portability. Below is an overview of the design features.

1. **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

Events	Threads
Computation is handled by event handlers	Computation is divided between threads
Mostly run to completion event handlers	Can be preempted
No stack overhead as there can be only one event handler running at a time	Context switch overhead
Used when application requires efficiency	Used when application requires flexibility
Allows high concurrency	It is not for concurrency intensive operations

Table 2: Events vs Threads.

- 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 [49].

2. **Execution Model:** 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 is an active debate in the community of embedded systems on using one of these execution models [82] [62] [40]. The comparison between the two can be found in Table 2. 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.

3. **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 such as [28] [77] [45] [53].

   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.

4. **Scheduling:** Real-time systems can be classified into periodic and aperiodic, 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 aperiodic 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 [55][48][70] .

5. **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.

6. **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.

Operating systems that exist in the literature are evaluated against the classification framework in the following sub sections.

5 OS Classification

This section classifies the existing operating systems based on the framework presented in section 4. Each feature along with its categories are taken and the operating systems that falls under each category are explained.

5.1 Architecture

Operating systems which fall under monolithic are TinyOS [42] and MagnetOS [23]. TinyOS uses component model at compile time and single static image at run time. Matè [49] built on top of TinyOS and CVM (Contiki VM) built on top of Contiki follows virtual machine architecture. MantisOS [25], Contiki [31], Sensor Operating System (SOS) [39], Bertha [54] and CORMOS [86] uses modular approach. Contiki provides services to be used by the processes. SOS uses modular approach at kernel as well as at application level. The application in SOS is built using set of interacting modules.

Monolithic	Modular	VM
TinyOS	SOS	VMSTAR
MagnetOS	Contiki	Matè
	MantisOS	MagnetOS
	CORMOS	ContkiVM
	Bertha	
	kOS	

Table 3: Architecture of Different OSs

Event-based	Thread-based	Hybrid	Others
TinyOS	MantisOS	Contiki (Event+Thread)	SenOS
SOS		kOS (Event+Object)	Nano-RK
CORMOS			
EYES			
PEEROS			

Table 4: Execution Model of Different OSs

5.2 Execution Model

5.2.1 Event-based

1. **TinyOS :** TinyOS is an event driven operating system which provides a programming framework for embedded systems. It has component-based execution model implemented in nesC which has a very low memory foot print. TinyOS concurrency model is based on commands, asynchronous events, deferred computation called tasks and split phase interfaces. The function invocation (as command) and its completion (as event) are separated into two phases in interfaces provided by TinyOS. Application user has to write the handler which should be invoked on triggering of an event. Commands and Event handlers may post a task, which is executed by the TinyOS FIFO scheduler. These tasks are non preemptive and run to completion. However tasks can be preempted by events but not by other tasks. Data race conflicts that arise due to preemption can be solved using atomic sections.The communication architecture of TinyOS uses the concept of Active Messages (AM) [83]. These are small packets of size 36 bytes and a one byte handler ID. A node after receiving active message, dispatches it to corresponding handlers that are registered. TinyOS event driven model has obvious disadvantages like low programming flexibility, non-preemption that are associated with event model.

2. **SOS :** SOS is developed in C and follows event-driven programming model. An application in SOS is implemented as one or more interacting modules. Modules are position independent binaries that implement a task or function. These are similar to services in Contiki [31] and tasks in TinyOS. Modules in SOS have well defined entry and exit point. Entry into the module is done only through one of the following methods. One is messages delivered from the scheduler and the other is calls to functions registered by the modules for external use. Each module will have handler function for message handling. Dynamic linker in the kernel enables the dynamic binary modules loading. SOS uses priority queues for scheduling messages as opposed to FIFO queue in TinyOS.

3. **EYES/PEEROS :** EYES [29] started with the motivation of meeting the goals like small size, power awareness, distribution and reconfigurability. It adapted event-driven execution model in order to achieve small size of code and limited available energy. Basic computational entity is task, which is a piece of code that runs to completion. Efficient utilization of distributed resources in the network are handled by resource management and remote procedure call mechanisms. It provides two abstraction layers for the programmer. Each of these layers provide some set of APIs for the developer. First level (called API level 0), *Sensor Network Layer*, provides APIs related to sensor data readings, information concerning available resources. It also provides APIs pertaining to transmitting data, gathering network information. Second level (API level 1), *Distributed Systems Layer*, it provides transparent access to the remote sensor nodes.

   Extension of EYES OS with real-time scheduling, memory management, resource management is PEEROS. PEEROS [60] offers real time guarantees for the applications. Task is the functional abstraction of PEEROS. Each task has a priority associated with it. PEEROS offers priority based multitasking for fast response to events. Priority based multitasking is achieved by scheduler that follows EDFI [47] algorithm. Communication and storage management are accomplished by messaging system and module manager respectively.

4. **CORMOS :** The basic abstractions that CORMOS [86] provides are *events*, *handlers*, and *paths*. Events can be for local as well as remote actions. Events are explicitly created by the handlers if they want to explicitly execute them. Handlers are functions that perform all the processing at each node. Each handler is invoked when they are scheduled by the event scheduler. CORMOS uses an earliest-deadline-first scheduler that makes it easy for applications to schedule events and hides from applications the existence of timer-based, asynchronous interfaces. Currently CORMOS scheduler puts the CPU to sleep when there is no pending computation. Handlers are atomic and run-to completion and should never block.

5.2.2 Thread-based

1. **MantisOS :** MantisOS is thread-driven operating system model for sensor networks. Thread is a simple computational entity which has its own state. Thread-driven model gives flexibility in writing applications as the developer is not concerned about task size which is mandatory in event-driven model. Execution of an application involves spawning multiple threads. Network stack and scheduler are also implemented as threads just like an application. Apart from these threads there is idle thread which runs when all other threads are blocked. Idle thread invokes the required power management routines. To maintain threads, kernel maintains a thread table that consists of thread priority, pointer to thread handler and other information about the thread. Scheduling between the threads is done by means of scheduler that follows priority based scheduling algorithm with round-robin semantics. Race conditions are avoided by using binary and counting semaphores.

   MantisOS suffers from the overheads of context switching and the memory allocated(in the form of stack) per each thread. This overhead is significant in resource constrained systems like WSN.

5.2.3 Hybrid

1. **Contiki :** Contiki combines the advantages of both events and threads. It is primarily an event driven model but supports multi-threading as an optional application level library. Application can link this library if it needs multi-threading. Events are classified as asynchronous and synchronous in Contiki. Synchronous events are scheduled immediately and

asynchronous events are scheduled later. Polling mechanism is used to avoid race conditions. In Contiki everything (communication, device drivers, sensors data handling) is implemented as service. Each of the service has interface and implementation. Application is aware of only interfaces. The service implementation can be changed at run time. This is done by stub library which is linked with the application for accessing services.

5.2.4 Others

1. **SenOS :** SenOS [44] is a finite state machine based OS. The main components of SenOS are *event queue*, *callback library*, and *a state transition table*. SenOS kernel polls event queue and takes corresponding actions which results in state transitions. Callback library provides a set of libraries to the application programmer for hardware access, communication and event handling. Each transition table defines an application running in WSN. The concurrency among the applications is provided by switching transition tables for execution.

2. **Nano-RK :** Nano-RK [33] is a reservation-based, energy-aware real-time operating system for wireless sensor networks. Computational units, tasks are associated with priorities, and higher-priority task always preempts the lower priority task. For time sensitive tasks in applications, it implements rate-monotonic scheduling algorithm for the tasks, so that the deadlines of the tasks are honored. Application can define their resource requirements and deadlines to meet. It also provides socket abstractions for communication. This OS is an extension of Resource-Kernel (RK) [66] paradigm to energy-limited resource constrained devices like WSN.

TinyOS purely uses event-based model whereas MantisOS uses thread-based model. Contiki on the other hand follows hybrid model, using event-based model at kernel level and providing threading as application library. MantisOS uses subset of POSIX threads. SOS has never taken a strong position on this issue since their focus is on reconfigurability, they use a concurrency model similar to TinyOS event model without the context switching overhead of MantisOS. Most of the implementations on mote-class sensor nodes with thread-based systems concluded that this model is unfit for them because of the RAM concerns.

5.3 Reprogramming

Figure 4 shows different levels of granularity that each OS can support in reprogramming. The order (from top to bottom) shown in figure is from higher to lower level of granularity. This classification mainly focuses on the finer granularity that a OS can support. As shown in figure flexibility in terms of changing application behaviour decreases, where as the updating cost increases when we go from top to bottom here.

5.3.1 Application Level

1. **TinyOS :** The behaviour of the TinyOS program can be changed through hard code transition or by modifying source code, recompile with TinyOS and placing the new image on the mote. It uses XNP [28] as code dissemination protocol. Later on TinyOS is supported by Deluge [45] and MOAP [77] for multi-hop network reprogramming. Reprogramming causes high communication overhead in TinyOS as the entire system image has to be re-flashed even for a small change. This is due to monolithic architecture of TinyOS.

   As such there is no memory protection that is offered by TinyOS. No virtual memory concept or memory management exists in TinyOS and the code is physically address binded.

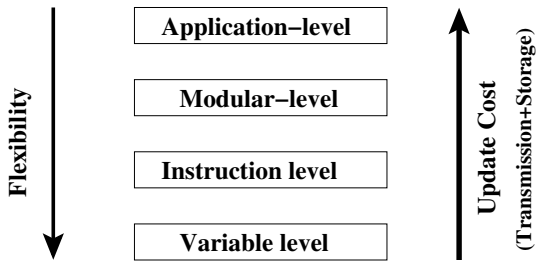

Figure 4: Reprogramming Flexibility and Update cost w.r.t Granularity

2. **SenOS :** Reconfigurability of applications is achieved by dynamically changing the transition table of that particular application. SenOS monitor plays vital role in replacing the transition table.

5.3.2 Modular or Component Level

1. **SOS :** The primary motivation and goal behind the development of SOS is reconfigurability which is the ability to add, modify and delete the software modules at run-time [71]. It supports dynamic native code updates through the loadable module systems(module granularity). Modules can be loaded and removed on the fly, with any necessary updates made to the function control blocks. Re-programmability facility in SOS middle-ground between the TinyOS with Deluge and Matè as it does reprogramming at modular level. SOS includes publish-subscribe mechanism that is similar to MOAP to distributed modules into the network. Depending on the user's needs it can use Deluge, MOAP or any other code distribution protocols. Since it is not required to have same modules or image running on the individual sensor nodes, one can use any sort of distribution protocols such as single-hop or multi-hop. Dynamic loading facility in SOS introduces significant problems related to memory allocation and message handling of modules. Lack of proper memory management policies might cause problems like insufficient memory for loading a module. Problems related to message handling are message to a non existent module or delivery of a message to improper handler.

2. **MantisOS :** Reprogramming facility is provided by MantisOS as a library which is built in to the kernel. Application uses this library calls to write new code and it is installed to run the updated code. This requires a software reset. MantisOS notion of dynamic reprogramming is currently limited to remote login and changing of variables/parameters. MantisOS provides a system call to actually implement the reprogramming capability. Support for reprogramming entire OS is in progress.

3. **Contiki :** The service implementation can be changed at run time. This is done by stub library which is linked with the application for accessing services. Dynamic loading and unloading of services can be flexibly done in Contiki. Instead of re-flashing the entire system image like in TinyOS and MantisOS, Contiki allows to reprogram only the required application service. Contiki does not follow proper memory management techniques which might cause an overhead while reprogramming. It assumes that code is position dependent. This requires the code to be loaded in to the same location of memory. This causes memory allocation problems if the code size increases.

4. **CORMOS :** Runtime system core of CORMOS consists of scheduler, memory allocator and module registry. Memory manager maintains a static table for allocating and de-allocating events. Size of the table is fixed at compile-time. A module may load or unload other modules

Real-time	Non Real-time
Nano-RK	TinyOS
CORMOS	SOS
PEEROS	Contiki
Nano-QPlus	MantisOS
DCOS	EYES
t-kernel	SenOS
	OSSTAR
	MagnetOS
	kOS
	T2

Table 5: Real-time and Non real-time OS classification

as well as register handlers during execution. This allows a designer to activate or deactivate a particular routing protocol or reliable protocol on demand as a module. Unlike TinyOS, CORMOS combines communication with processing in its base run-time system. Application in CORMOS is composed of interacting modules. Currently, modules that constitute the user application and core system extensions are loaded during system initialization within the main function. Dynamic reconfigurability of software modules at run-time is still under progress. Communication protocols used are not power-efficient.

5.3.3 Instruction Level/Variable Level

1. Matè [49] is used to support easy reprogramming at instruction level on top of TinyOS. It runs as a component on top of TinyOS. The code fits in terms of capsules (24 instructions). Once a capsule is installed on a mote in the network, it virally propagates through out the network. Every instruction is executed as a TinyOS task. Matè uses Trickle [53] as code dissemination protocol. This is built on top of Deluge by proposing suppression mechanisms for control and data messages, and methods for periodically broadcasting advertisements to increase reliability.

2. Dynamically extensible Virtual Machine (DVM) [22] implemented on top of SOS, performs execution of scripts, tuning parameters apart from modular updates and replacing entire image.

5.4 Scheduling

Classification of OSs into real-time and non real-time is shown in table 5. Execution model presented in 5.2 gives the scheduling algorithm details of each individual systems. Real-time scheduling algorithms can be used at the cost of performance.

5.5 Power Management

System wide power management includes controlling of micro-controller (processor) and radio as well.

1. **TinyOS :** TinyOS provides API in order to conserve and manage power properly. It manages radio [68] as well as processor. Application should call *HPLPowerManagement.Enable()* in its *StdControl.Init()*. This enables the processor to sleep whenever possible upon the next clock after the following conditions are met. (i) The radio is off (call *CC1000RadioC.StdControl.stop()*). (ii) All clock interrupts are disabled. (iii) Enable the SPI interrupt. and (iv) task queue is

API	Functionality
query_energy()	Query residual battery energy
set_energy_mode()	Set energy savings mode (future)
get_energy_mode()	Get energy savings mode (future)
tx_power_set()	Change radio transmitter power
powerdown()	Power the system down for t seconds

Table 6: Power Management APIs of Nano-RK

empty. In addition to the radio, each service has to be stopped. Further, in the Stop() function, each service should call *HPLPowerManagement.adjustPower()* [52].

2. **MantisOS :** MantisOS achieves energy-efficiency using power-efficient scheduler which makes the processor to sleep after all active threads called MantiOS *sleep()* function. This function resembles UNIX sleep() function. First the application should explicitly activate power save mode with *mos_enable_power_mgt()*, before it actually calls sleep function. Power-aware scheduling is implemented by the idle thread which may detect patterns in CPU utilization and adjust kernel parameters to conserve energy. *dev_mode()* provided by the OS can be used to set a device to idle or off depending on the application requirements to conserve power. The mode has to be set back again in order to use the device again.

3. **SenOS :** SenOS provides power management as an application layer protocol for sensor network management. It uses Dynamic Power Management (DPM) [73] algorithm, which determines sleep-rate transition dynamically at run-time. As DPM can be expressed as finite state machine model.

4. **EYES/PEEROS :** One among the design criteria of EYES/PEEROS is support of low power mode. They started this at lowest possible layer i.e hardware. EYES sensor node provides several power-saving modes. The transceiver TR101 used by this node has low power consumption. When there are no more external events to take process, the node enters the power-saving mode. External events will wake the node to its normal operation.

5. **SOS :** The power management has not been considered explicitly in SOS. In the recent version of SOS kernel [74], it includes DS2438 kernel module to explicitly monitor the battery on the node. It provides APIs to access the battery voltage, battery current, DS2438 chip temperature.

6. **Contiki :** Even though Contiki does not provide explicit power management abstractions, it allows the application programmers to implement such mechanisms. One such provision is exposing its internal queue state to the applications. Applications can decide to power down the system when there are no events to be scheduled. The processor wakes up in response to an external event. Which is handled by an external poll handler.

7. **Nano-RK** Nano-RK provides wide variety of APIs. Power Management APIs provided by it are shown in table 6

5.6 Miscellaneous

5.6.1 Simulation Support

1. **TinyOS :** TOSSIM [51] simulates TinyOS applications for sensor network. The same code can be used for both simulation as well as for testbed deployment. TOSSIM simulates the

TinyOS network stack at the bit level, allowing experimentation with low-level protocols in addition to top-level application systems. It scales extremely well for network inactive applications and moderately well for network intensive applications. It is not designed to simulate heterogeneous sensor networks. TOSSIM also has a GUI tool, TinyViz, which can visualize and interact with running simulations. Using a simple plug-in model, users can develop new visualizations and interfaces for TinyViz.

EMTOS [37] allows to run nesC/TinyOS application as a single module in EMSTAR [36]. EMTOS wrapper library for works in a similar way to TOSSIM. In case of TOSSIM *pc* platform is used to build TOSSIM, here the *emstar* module is used to build EMTOS.

2. **SOS :** SOS simulation framework which is similar to TOSSIM, can be used to simulate SOS applications on a PC. SOS is also supported by both Avrora [80] and emStar [36] frameworks. SOS has an added advantage of debugging code with gdb.

3. **MantisOS :** Mantis provides multi-model prototyping environment which extends beyond simulation framework for development of network management and visualization applications within the MANTIS sensor network. It enables developer to test the written code on virtual sensor nodes as well as later on original sensor nodes. It allows to integrate virtual environment with the real deployment network. It also permits a virtual node to leverage other APIs outside MANTIS APIs.

4. **Contiki :** Earlier Contiki supported basic simulation environment. It has been ported onto FreeBSD to run as a user-level process. Each node in the simulation environment is represented by process started on its own. These nodes are well connected by simulated network layer [30]. But, this lacks in synchronization among processes as they were running individually resulting in non-deterministic simulation results. There was no support for heterogeneous networks in this simulator. Cooja [35] is advanced simulation environment for Contiki OS.

5. **SenOS :** State-machine based approach may not be advantageous for complex applications where they cannot be expressed as states or they result in innumerable number of states (state explosion problem). It gives less flexibility in terms of programming. Reprogramming in SenOS cannot support changes other than state transitions.

5.6.2 Platform Support

1. TinyOS is gaining its importance in the WSN applications, and has been ported to different platforms. The list of platforms supported as of now are shown in table 7. Other platforms (commercial and non-commercial) supported are Eyes (University of Twente,Netherlands), TMote Sky (moteIV), MicaZ (XBow) and iMote (Intel).

2. Different hardware platforms supported by SOS are shown in table 7.

3. The list of platforms supported by MantisOS as of now are shown in table 7.

4. The list of platforms supported by Contiki as of now are shown in table 7.

5. Nano-RK is implemented on Firefly node built at CMU.

6. EYES and PEEROS are implemented on EYES [34] sensor node.

TinyOS	SOS	Contiki	MantisOS
Telos	Cricket	avr MCU	Mica2
Mica2Dot	imote2	MSP430 MCU	MicaZ
Mica2	Mica2	x86	Telos
Mica	Micaz	6502	Mantis nymph
TMote Sky	tmote		
Eyes	XYZ		
MicaZ	Protosb		
iMote	avrora		
	cyclops		
	emu		

Table 7: Hardware Platforms Supported by different OS

5.7 Other Operating Systems

5.7.1 t-Kernel

t-kernel has started with motivation to achieve three goals: (i) OS Protection, (ii) Virtual Memory, and (iii) Preemptive scheduling. Virtual memory is provided by using flash and no memory management hardware. t-kernel supports priority based preemptive scheduling and protects OS kernel code for safety from bugs in application code.

5.7.2 DCOS

DCOS [43] is a light weight Data Centric Operating System for embedded devices like WSN. The main features of this OS are real-time scheduling, dynamic configurations and the data-centric approach. DCOS differs from the other OS in providing the above features. The main components of the kernel include real-time scheduler, data manager, dynamically loadable modules, and the other support like dynamic memory allocation, supporting various hardware devices. It uses EDFI scheduler [47] similar to Nano-RK. Data-centric architecture of it helps in reconfiguration functionality. Functional building blocks are centrally co-ordinated using this architecture and rearranging these connections among them gives new functionality. Dynamically Loadable Modules (DLMs) is a separate task from the kernel and compiles separately. DLMs are relocatable and can be loaded and executed. It also supports dynamic allocation of memory to the modules and for various hardware devices.

5.7.3 MagnetOS

The main motivation of MagnetOS [23] is to provide unifying single system abstraction and adaptability. Unifying single system abstraction implies, to consider the whole system as a single unified Java Virtual Machine (JVM) to the application. This is achieved by providing a Single System Image (SSI) to the application which consists of static and dynamic components. The advantages of SSI are easy application development and adaptability. Application development is easy because the developer need not concern about the details of object location, communication issues, monitoring mechanisms and resource constraints of nodes. Adaptability is accomplished by Magnet OS virtual machine. Netpull and Netcentre are the algorithms for deciding optimal placement and migration of application components according to the available resources of the nodes.

OS	Web URL
TinyOS	http://www.tinyos.net
SOS	http://nesl.ee.ucla.edu/projects/sos
Contiki	http://www.sics.se/∼adam/contiki
t-Kernel	http://www.cs.virginia.edu/∼lg6e/t-kernel
MantisOS	http://mantis.cs.colorado.edu
EYES	http://www.eyes.eu.org/
PEEROS	http://www.eyes.eu.org/
VMSTAR	http://senses.ucdavis.edu
OSSTAR	http://senses.ucdavis.edu
MagnetOS	http://www.cs.cornell.edu/People/egs/magnetos/
Nano-QPlus	http://www.etri.re.kr
kOS	http://www.ee.ucl.ac.uk/secoas/
CORMOS	http://www.ics.forth.gr/carv/scalable/cormos.html
SenOS	http://redwood.snu.ac.kr/
DCOS	http://www.eyes.eu.org/

Table 8: OS Research Projects in WSN.

5.7.4 kOS

KOS [26] views Sensor network similar to biological -like system of automaton. The desirable characteristics that exist common to both are scalability, robustness, simplicity and self organization. KOS implements run to completion model and asynchronous messaging service similar to TinyOS. System functionality in KOS is abstracted in to objects and methods.

Other OSs in progress are Jallad [46], OSSTAR [63] and T2 [50]. Jallad OS is application specific and is exclusively for space applications.

The above comparison of existing OSs shows the importance of programming model, kernel structure and the granularities of reprogramming issues in designing a OS for embedded and general purpose nature of WSN. The existing runtime system infrastructures specified above does not provide clear and concise separation of the application and the underlying platform. They lack in real-time scheduling of events, interface to manage battery resource efficiently, and reconfigurability of system on its own to the changing dynamics. It also reveals us the fact that there is a thin layer between operating system and middleware and we foresee that they should be built in some structured way to exploit all the benefits of underlying network of sensor nodes.

6 Evaluation of Operating Systems

Some of the research projects that are completed and ongoing are listed in table 8. After looking at the various operating systems that exists in the literature for WSN, it is high time to compare and evaluate the existing operating systems. Summary of the existing operating systems and the supported features by them is shown in table 9.

7 Applications of WSN

7.1 Categories of Applications

WSNs have established their presence in wide variety of applications, and researchers are trying to experiment more. The broader classification of these applications depends on the nature of the

S.No	Project	Execution Model	Priority-based Scheduling	Real-time guarantee	Dynamic Reprogramming	Memory Management	Low Po Mod
1	TinyOS	Component-based	×	×	×	×	√
2	SOS	Module-based	√	×	√	×	×
3	Contiki	Hybrid[2]	×	×	√	×	√
4	MantisOS	Thread-based	×	×	√	×	√
5	EYES	Event-based	√	×	×	×	√
6	PEEROS	Event-based	√	√	×	√	√
7	SenOS	State-based	×	×	√	×	√
8	VMSTAR	VM-based	×	×	√	×	×
9	OSSTAR	Hybrid[3]	×	×	√	√	×
10	Nano-QPlus	Thread-based	√	√	×	×	√
11	CORMOS	Event-based	×	√	√	√	×
12	MagnetOS	VM-based	×	×	×	×	√
13	Nano-RK	Task-based	√	√	√	×	√
14	kOS	Object-based	×	×	×	×	√
15	DCOS	Data-Centric	×	√	√	×	√
16	T2	Component-based	×	×	×	×	√
17	t-Kernel	Task-based	√	×	√	√	√

Table 9: Summary of Operating Systems

purpose they are used for. Based on this, they can be categorised into the following categories: (i) Monitoring, (ii) Detecting/Classifying/Tracking and (iii) Controlling. These applications are used in different domains. They are:

1. **Environmental Monitoring:** This domain of applications can be broadly categorised into indoor and outdoor monitoring [78].

   (a) Indoor monitoring applications typically include monitoring buildings and offices. These applications involve sensing temperature, light, air streams, air pollution modalities. Other important indoor applications are fire and smoke detection. Civil engineering research shown that WSNs can also be used to monitor the civil structures like bridges and buildings for vibrations.

   (b) Interesting outdoor applications of WSN are chemical hazardous detection, habitat monitoring, high way traffic monitoring, earth quake detection, volcano detection, forecasting weather phenomenon, river monitoring, flooding prediction. Sensor nodes have found their applicability in agriculture also. They can be used to detect early frost damage, climatic conditions and improve crop management and reduce costs. Soil moisture monitoring system is one more prominent application of WSN in agriculture.

   Major projects that focused on this domain are: (i) GreatDuckIsland [57], (ii) Vineyard monitoring [27] (iii) CORIE system [24],(iv) Sensor Scope [12], (v) ALERT [1], (vi) SECOAS [11], (vii) Seismic/acoustic monitoring system [10], (viii) Camalie Net [3], (ix) MASNET [9], (x) FIRE project [6], (xi) Spirit [13]

2. **Health:** Advances in WSN opened up new opportunities in the health care systems. This domain of applications typically involve monitoring patients and alerting doctors. Here sensors measure the recent actions of the patients and remind the doctors about the behaviour of the patient. Another application might be tracking patients, doctors, and drug usage in the hospitals. [76] gives an overview of potential applications of WSN in health care and presents

S.No	Category	Real-time Guarantees	Dynamic Reprogramming	Power Management	Scalability	Suitable OS
1.	Environmental Monitoring					
	(a). Indoor	×	×	×	×	Any operating system
	(b). Outdoor	×	√	√	√	SOS, Contiki, MantisOS SenOS, DCOS Nano-RK, t-kernel
2.	Health	√	×	×	×	PEEROS, Nano-RK DCOS, CORMOS
3.	Military	√	√	√	√	Nano-RK DCOS
4.	Industry	√	√	√	√	Nano-RK DCOS

Table 10: Application Characteristics and Suitable OSs

the issues and challenges involved in those applications. Survey of medical applications of WSN can be found in [61].

Major projects that focused on this domain are: (i) CodeBlue [4] [58], (ii) SSIM [69], (iii) WHMS [14] [59], (iv) Wise-Pac [16]. More comprehensive list of applications in the are of body sensor networks can be found in [2].

3. **Military Applications:** Sensor nodes are well suited to the need of military applications. Interesting of them are information collection, enemy tracking, battlefield surveillance, target classification, perimeter security, border patrol.

   Major projects that focused on this domain are: (i) ExScal [5]

4. **Industry:** Efficient production, minimizing cost, and ensuring safety is crucial to any industry and it requires comprehensive monitoring and control of industrial processes and equipment. MachineTalker [8] is one example of this kind that made possible to monitor the fluid level in multiple fuel storage tanks in petroleum refining and storage facilities. WiSA [15] is an academic project that focused to monitor and control functions in the production cycle to enhance the performance while reducing maintenance of them. [19] [72] presents potential applications of WSNs in industrial automation applications. General Electric (GE) uses WSN to monitor their industrial equipment [7].

More detailed survey of WSN applications can be found in [20] [78] [85].

7.2 Characteristics of Applications

The different kinds of applications used in different domains mentioned above exposes us lot many characteristics. They also tell us what kind of underlying system software should be present in order to satisfy those characteristics. The important characteristics/requirements of these applications summarised from the above applications are:

1. Most of the applications require real-time guarantees on the data. This is especially true in the case of health and military applications. (Real-time)

2. Most of the applications of WSN might last for longer days, and the requirements of the application may change over a period of time. Run-time environment should allow to reconfigure the application at run-time by changing its code. (Reprogramming)

3. Applications like monitoring are intended to run for longer periods of time, as the power source available is sacred and nevertheless it is difficult to replace the battery as WSNs are deployed in inaccessible terrains and deserts. It is mandatory to conserve the usage of power by appropriate means at the operating system level. (Power Management)

4. Since these applications run on hundred to thousands of sensor nodes, the run-time environment should be able to scale to number of nodes without degrading the applications performance. (Scalability)

These characteristics of applications are evaluated against the categories of applications in table 10. ($\times$) indicates not required and ($\surd$) indicates required. The possible OS that can be used is shown against each category of applications in the table.

The characteristics shown in table 10 influences the features of the OS. Architecture and the memory management of OS decides reprogramming functionality. Execution model and the scheduling mechanism decides the real-time guarantees real-time property.

8 Future Research Directions

Miniatured sensor nodes that form WSN are part of the embedded systems category. But they differ in characteristics like resource constraints, energy constraints, and nature of deployment. Several embedded operating systems do exist in the literature. OS design for WSNs poses significant challenges as compared to the embedded systems. They demand a novel and compact OS design. Sensor network OSs mainly targeted at achieving goals like *low footprint OS*, *flexible execution model*, *reprogramming*, *efficient power management* and *real-time support*.

The features of OS in WSN are from developer, application and system perspectives. The idea of providing programming model (event model, thread model), power management (power aware policies) and reprogramming (dynamic loading of modules) support are from developer, system and application perspectives respectively.

Developer Perspective: Execution model provided by operating system serves the purpose of application developer by providing a suitable computational abstractions. Thread model is flexible for application developer because he/she need not concern about the implementation details regarding indefinite blocking of tasks, busy-wait polling, however at the cost of system overhead for context switching and stack memory.

Application Perspective: Application requirements in WSN are prone to change. It is due to long running nature of the network. Reprogramming gives the flexibility in changing application requirements at run time. Another feature that is adopted by many OSs is event driven model which is also from application perspective. Most of the applications that run on sensor networks are event driven and hence system should support concurrency, run-to-completion semantics that are apparent from event driven applications.

System (WSN) Perspective: Another feature that is mostly provided by sensor OS are power management policies. These are necessary from system perspective because power is an important resource to be efficiently exploited in sensor networks. Some of the policies that are employed are sleeping mechanism, power aware scheduling and controlling of sensing rate. The following subsections discuss the characteristics to be present in an ideal WSN operating system.

8.1 Execution Model

Event-based execution model is best suited for the asynchronous nature systems. Event driven nature of applications has led to the adaptation of even-based execution model for some of the WSN operating systems [42] [39] [31] [86] [60]. Even though this best suits for these systems,

it is not suitable when the events to be processed take longer time since the events are run-to-completion in nature. For example when a node is processing, it can not handle the incoming packets. These incoming packets have to be buffered for processing, which may lead to buffer overflow as the node has little or no memory. It is some times reasonable to interrupt the executing task in order to process other tasks. This is where the application of thread-based execution makes feasible. This model provides the flexibility in pre-emptying the tasks while they are executing. MantisOS [25] has adopted the thread-based execution model, where as the Contiki [31] provided it as an application library for the user, giving flexibility in choosing between the execution models. Contiki uses protothreads [32], which are lightweight and does not require stack for each thread. Event-based and thread based are the most prominent execution models used in sensor network operating systems. The decision of choosing either of these models depend on the application requirements like flexibility, efficiency and features like concurrency, multitasking.

8.2 Reprogramming and Memory Management

Reprogramming requires dynamic loading and unloading of software components. This may be at different levels ranging from a component to a data variable. Component can be entire OS image, module, and service. Modular [4] level reprogramming seems to be advantageous when compared to lower level reprogramming granularities like a data variable. This is because the binary image of the module with proper interfaces and relocatable code can be easily linked with other system modules. On the other hand reprogramming requires proper memory management policies like contiguous memory allocation, deallocating memory, and paging. Due to lack of these policies reprogramming might pose restrictions like constant module size and non-relocation of code. For these policies to be enforced by the application developer proper APIs should be provided by OS. This might result in access violation, unsafe memory protection etc. Even though these policies are required for relaxing run time memory restrictions, there is a trade off between performance and flexibility.

8.2.1 Architecture

The granularity of reprogramming often depends on architecture of the OS. System architecture should be of component oriented which gives flexibility in terms of adaptability, modularity, extendibility, maintainability and re-usability [65]. Application development would be easier as application is simply the composition of interacting components.

8.3 Power Management

Power management plays a vital role in prolonging node life time. In addition to sleep modes, the OS should also consider application defined duty cycle to conserve power. The OS should provide power management API so that application can use sleep functionality on any .

8.4 Real-Time Nature

Applications like industrial process control, contaminant detection etc requires real-time guarantees. As the environment in which sensor networks operate is unforeseen, the operating conditions are quite unpredictable. In the applications like nuclear reactor monitoring, environmental monitoring it is necessary to provide strict hard and soft deadlines respectively. This requires support from scheduler which employs scheduling policies like EDF, FCFS that ensure delivery deadlines. Programmer should have flexibility in choosing the scheduling policy based on the application requirements. Due to the characteristics of WSN like communication channel fading, unstable topology, quick energy depletion it would be hard to ensure hard real time guarantees.

[4] Module here refers to a position independent software component with proper interfaces

8.5 Context-Awareness:

The conditions under which the application executes varies periodically. The system might face abrupt changes in terms of available resources,environmental conditions, sudden failures etc. The run time environment should be context aware and adapt the application to the system dynamics. There is certainly a trade-off between performance and adaptability. Virtual machine approach can be a solution to provide context awareness but increases complexity of the system.

8.6 Support for Multiple Applications

Because of the potential applications of WSN, and growing interest on WSNs enforced to deploy multiple application simultaneously on the same network. It gives an advantage of reducing infrastructure deployment cost. But, it is an interesting research challenge. Till now, there are less or no OSs that facilitate running multiple applications.

Though other desirable characteristics like low memory foot print and portability are mandatory, the above are the important objectives to be considered while designing OS for a embedded WSN systems.

9 Conclusions

This survey helps the researchers in understanding various aspects like issues, design characteristics while building WSN system software in general and in particular to OS. The issues presented here motivates the design principles to be considered/followed while designing an OS for WSN. The proposed classification framework presents noteworthy features of a typical WSN OS and classifies existing operating systems according to them. The purpose of analysing different applications, their characteristics and suggesting an ideal OS for those applications will help an application developer to choose an operating system. The survey not only explores the existing design approaches of OSs for WSN, but also explores new requirements to be considered by keeping in mind the future applications of WSN.

References

[1] ALERT . http://www.alertsystems.org/.

[2] Body Sensor Networks. http://bsn-web.org/.

[3] Camalie Net. http://camalie.com/WirelessSensing/ WirelessSensors.htm.

[4] CodeBlue : Wireless Sensor Networks for Medical Care. http://www.eecs.harvard.edu/ mdw/proj/codeblue/.

[5] ExScal: Extreme Scale Wireless Sensor Networking. http://cast.cse.ohio-state.edu/exscal/.

[6] FIRE (Fire Information and Rescue Equipment) Project. http://fire.me.berkeley.edu/.

[7] GE Global Research Center. http://www.ge.com/research/grc_2_2_1.html.

[8] Machine Talnker. http://www.machinetalker.com.

[9] Mobile Actuator Sensor Networks. http://mechatronics.ece.usu.edu/mas-net/.

[10] Monitoring volcanic eruptions with a wireless sensor network. http://www.eecs.harvard.edu/ mdw/proj/volcano/.

[11] Self-Organising Collegiate Sensor Networks. http://www.secoas.org/.

[12] Sensor Scope Project. http://sensorscope.epfl.ch/.

[13] Spirit: Waste Management System Monitoring using Wireless Sensors Networks. http://mist.cs.wayne.edu/spirit/spirit.html.

[14] WHMS - Wearable Health Monitoring Systems. http://www.ece.uah.edu/ jovanov/whrms/.

[15] WiSA: Wireless Sensor and Actuator Networks for Measurement and Control. http://www.control.hut.fi/Research/WiSA/.

[16] Wise-Pac: Sensor Networks for Medical Applications. http://www2.enel.ucalgary.ca/People /Haslett/WCLM/CCHE/WebPage/AdHoc.html.

[17] Moteiv Corporation, 2007. http://www.moteiv.com.

[18] ZigBee Alliance, 2007. http://www.zigbee.org.

[19] A. S.-V. A. Bonivento, L.P. Carloni. Platform based design of wireless sensor networks for industrial applications. In *Proceedings of Design Automation and Test in Europe (DATE), Munich*, March 2006. http://www.gigascale.org/pubs/851.html.

[20] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer Networks*, 38(4):393–422, 2002.

[21] J. N. Al-Karaki and A. E. Kamal. Routing techniques in wireless sensor networks: a survey. *Wireless Communications, IEEE [see also IEEE Personal Communications]*, 11(6):6–28, 2004.

[22] R. Balani, S. Han, R. Kumar, I. Tsigokannis, and M. Srivastava. Multi-level software reconfiguration for sensor networks. In *Proceedings of the Sixth Annual ACM Conference on Embedded Software (EMSOFT-2006)*, Seoul, South Korea, October 22-25 2006.

[23] R. Barr, J. C. Bicket, D. S. Dantas, B. Du, T. W. D. Kim, B. Zhou, and E. G. Sirer. On the need for system-level support for ad hoc and sensor networks. *SIGOPS Oper. Syst. Rev.*, 36(2):1–5, 2002.

[24] P. Barrett. CORIE Project. http://www.ccalmr.ogi.edu/CORIE/.

[25] S. Bhatti, J. Carlson, H. Dai, J. Deng, J. Rose, A. Sheth, B. Shucker, C. Gruenwald, A. Torgerson, and R. Han. Mantis os: An embedded multithreaded operating system for wireless micro sensor platforms. *Mobile Networks and Applications*, 10(4):563–579, January 2005.

[26] M. Britton, V. Shum, L. Sacks, and H. Haddadi. A biologically inspired approach to designing wireless sensor networks, 2005.

[27] J. Burrell, T. Brooke, and R. Beckwith. Vineyard computing: Sensor networks in agricultural production. *IEEE Pervasive Computing*, 03(1):38–45, 2004.

[28] Crossbow Technology Inc. Mote in-network programming user reference, 2003.

[29] S. Dulman and P. Havinga. Operating system fundamentals for the EYES distributed sensor network. In *Proceedings of Progress 2002*, Utrecht, the Netherlands, October 2002.

[30] A. Dunkels, L. M. Feeney, B. Grnvall, and T. Voigt. An integrated approach to developing sensor network solutions. In *Proceedings of the Second International Workshop on Sensor and Actor Network Protocols and Applications*, Boston, Massachusetts, USA, 2004. Invited paper.

[31] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - a lightweight and flexible operating system for tiny networked sensors. In *LCN '04: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, pages 455–462, Washington, DC, USA, 2004. IEEE Computer Society.

[32] A. Dunkels, O. Schmidt, and T. Voigt. Using Protothreads for Sensor Node Programming. In *Proceedings of the REALWSN'05 Workshop on Real-World Wireless Sensor Networks*, Stockholm, Sweden, June 2005.

[33] A. Eswaran, A. Rowe, and R. Rajkumar. Nano-rk: An energy-aware resource-centric rtos for sensor networks. *rtss*, 0:256–265, 2005.

[34] EYES. Eyes sensor node. http://www.eyes.eu.org/sensnet.htm.

[35] F.Osterlind. A sensor network simulator for the contiki os. Technical Report Tech. Rep. T2006-05, Swedish Institute of Computer Science (SICS), Febraury 2006.

[36] L. Girod, J. Elson, A. Cerpa, T. Stathopoulos, N. Ramanathan, and D. Estrin. Emstar: a software environment for developing and deploying wireless sensor networks. In *Proceedings of the 2004 USENIX Technical Conference*, Boston, MA, 2004.

[37] L. Girod, T. Stathopoulos, N. Ramanathan, J. Elson, D. Estrin, E. Osterweil, and T. Schoellhammer. A system for simulation, emulation, and deployment of heterogeneous sensor networks. In *Proceedings of the Second ACM Conference on Embedded Networked Sensor Systems*, Baltimore, MD, 2004.

[38] S. Hadim and N. Mohamed. Middleware: Middleware challenges and approaches for wireless sensor networks. *IEEE Distributed Systems Online*, 7(3):1, 2006.

[39] C.-C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava. A dynamic operating system for sensor nodes. In *MobiSys '05: Proceedings of the 3rd international conference on Mobile systems, applications, and services*, pages 163–176, New York, NY, USA, 2005. ACM Press.

[40] J. Hellerstein and T. Roscoe. Events and threads. Lecturer Notes, November 2005.

[41] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. *SIGARCH Comput. Archit. News*, 28(5):93–104, 2000.

[42] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[43] T. Hofmeijer, S. Dulman, P. G. Jansen, and P. J. M. Havinga. Dcos, a real-time light-weight data centric operating system. In S. Sahni, editor, *IASTED Int. Conf. on Advances in Computer Science and Technology (ACST), St. Thomas, Virgin Islands, USA*, pages 259–264, Calgary, Canada, November 2004. ACTA Press.

[44] S. Hong and T. Kim. Senos: state-driven operating system architecture for dynamic sensor node reconfigurability. In *International Conference on Ubiquitous Computing*, pages 201–203, October 2003.

[45] J. W. Hui and D. Culler. The dynamic behavior of a data dissemination protocol for network programming at scale. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 81–94. ACM Press, 2004.

[46] A. Jallad and T. Vladimirova. Operating systems for wireless sensor networks in space. In *Proceedings of the 8th Military and Aerospace Applications of Programmable Logic Devices and Technologies International Conference (MAPLD'2003)*, pages P–1005, Washington DC, US, NASA, September 7-9 2005.

[47] P. G. Jansen, S. J. Mullender, P. J. M. Havinga, and J. Scholten. Lightweight edf scheduling with deadline inheritance. Technical Report TR-CTIT-03-23, University of Twente, Enschede, May 2003.

[48] J. P. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm: Exact characterization and average case behavior. In *IEEE Real-Time Systems Symposium*, pages 166–171, 1989.

[49] P. Levis and D. Culler. Mate: A tiny virtual machine for sensor networks. In *International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, USA*, Oct. 2002. To appear.

[50] P. Levis, D. Gay, V. Handziski, J.-H.Hauer, B.Greenstein, M.Turon, J.Hui, K.Klues, C.Sharp, R.Szewczyk, J.Polastre, P.Buonadonna, L.Nachman, G.Tolle, D.Culler, and A.Wolisz. T2: A second generation os for embedded sensor networks. Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universität Berlin, November 2005. http://www.tkn.tu-berlin.de/publications/papers/T2_TR.pdf.

[51] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proceedings of the 1st international conference on Embedded networked sensor systems*, pages 126–137, New York, NY, USA, 2003. ACM Press.

[52] P. Levis, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. A. Brewer, and D. E. Culler. The emergence of networking abstractions and techniques in tinyos. In *NSDI*, pages 1–14, 2004.

[53] P. Levis, N. Patel, D. Culler, and S. Shenkar. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor network. In *1st Symposium on Networked Systems Design and Implementation (NSDI'04)*, pages 15–28, San Francisco, CA,, March 2004.

[54] J. Lifton, D. Seetharam, M. Broxton, and J. A. Paradiso. Pushpin computing system overview: A platform for distributed, embedded, ubiquitous sensor networks. In *Pervasive*, pages 139–151, 2002.

[55] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[56] C. Lynch and F. O. Reilly. Processor Choice For Wireless Sensor Networks. In *Workshop on Real-World Wireless Sensor Networks (REALWSN'05)*, Stockholm, Sweden, June 2005.

[57] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson. Wireless sensor networks for habitat monitoring. In *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 88–97, New York, NY, USA, 2002. ACM Press.

[58] D. Malan, T. Fulford-Jones, M. Welsh, and S. Moulton. Codeblue: An ad hoc sensor network infrastructure for emergency medical care. In *International Workshop on Wearable and Implantable Body Sensor Networks*, April 2004.

[59] A. Milenkovi, C. Otto, and E. Jovanov. Wireless sensor networks for personal health monitoring: Issues aid an implementation. *Computer Communications*, 2006. http://www.ece.uah.edu/ jovanov/papers/index.html.

[60] Mulder, S. Dulman, L. van Hoesel, and P. Having. Peeros-system software for wireless sensor networks, August 2003.

[61] I. Noorzaie. Survey paper: Medical applications of wireless networks. http://www.cs.wustl.edu/j̃ain/cse574-06/medical_wireless.htm, April 2006.

[62] J. Ousterhout. Why threads are a bad idea (for most purposes). USENIX Winter Technical Conference, Jan 1996.

[63] R. Pandey, J. Kottalam, Y. Ramin, I. Wirjawan, and J. Kosh. Osstar: A scalable component-based operating system for sensor network. Technical report, University of California, Davis, Davis, California, 2005. in Preparation.

[64] H. Park, W. Liao, K. H. Tam, M. B. Srivastava, and L. He. A unified network and node level simulation framework for wireless sensor networks. Technical report, September 7 2003.

[65] N. Parlavantzas, G. Coulson, M. Clarke, and G. Blair. Towards a reflective component based middleware architecture, 2000.

[66] R. Rajkumar, K. Juvva, A. Molano, and S. Oikawa. Resource kernels: a resource-centric approach to real-time and multimedia systems. volume 3310, pages 150–164. SPIE, 1997.

[67] K. Römer and F. Mattern. The design space of wireless sensor networks. *IEEE Wireless Communications*, 11(6):54–61, Dec. 2004.

[68] R.Szewczyk. Tinyos 1.1 power management feature. http://www.tinyos.net/tinyos-1.x/doc/changes-1.1.html, September 2003.

[69] L. Schwiebert, S. K. Gupta, and J. Weinmann. Research challenges in wireless networks of biomedical sensors. In *MobiCom '01: Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 151–165, New York, NY, USA, 2001. ACM Press.

[70] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols: An approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[71] R. Shea, C.-C. Han, and R. Rengaswamy. Motivations Behind SOS. Technical Report SOS2000-1, University of California Los Angeles, Networked Embedded Systems Lab, Los Angeles, CA, February 2004.

[72] X. Shen, Z. Wang, and Y. Sun. Wireless sensor networks for industrial applications. In *Fifth World Congress on Intelligent Control and Automation (WCICA)*, volume 4, pages 3636–3640, June 2004.

[73] A. Sinha and A. Chandrakasan. Dynamic power management in wireless sensor networks. *IEEE Des. Test*, 18(2):62–74, 2001.

[74] Sos documentation. http://nesl.ee.ucla.edu/projects/sos-1.x/doxygen/doxygen/html/index.htm.

[75] W. Stallings. *Wireless Communications and Networks*. Prentice Hall Professional Technical Reference, 2001.

[76] J. Stankovic, Q. Cao, T. Doan, L. Fang, Z. He, R. Kiran, S. Lin, S. Son, R. Stoleru, and A. Wood. Wireless sensor networks for in-home healthcare: Potential and challenges. In *High Confidence Medical Device Software and Systems (HCMDSS) Workshop*, Philadelphia, PA, June 2005.

[77] T. Stathopoulos, J. Heidemann, and D. Estrin. A remote code update mechanism for wireless sensor networks. Technical Report CENS-TR-30, University of California, Los Angeles, Center for Embedded Networked Computing, November 2003.

[78] A. Th, L. J, and M. S. A survey of applications of wireless sensors and wireless sensor networks. In *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*, pages 719– 724, 2005.

[79] S. Tilak, N. B. Abu-Ghazaleh, and W. Heinzelman. A taxonomy of wireless micro-sensor network models. *SIGMOBILE Mob. Comput. Commun. Rev.*, 6(2):28–36, 2002.

[80] B. L. Titzer, D. K. Lee, and J. Palsberg. Avrora: scalable sensor network simulation with precise timing. In *IPSN '05: Proceedings of the 4th international symposium on Information processing in sensor networks*, page 67, Piscataway, NJ, USA, 2005. IEEE Press.

[81] Venkateswarlu R and Janakiram D. A simple model for evaluating the scalability in wireless sensor networks. In *International Conference on Intelligent Sensors, Sensor Networks and Information Processing Conference*, pages 97– 100, December 2005.

[82] R. von Behren Jeremy Condit and E. Brewer. Why events are a bad idea(for high-concurrency servers). In *10th Workshop on Hot Topics in Operating Systems (HotOS IX)*, May 2003.

[83] T. von Eicken, D. E. Culler, S. C. Goldstein, and K. E. Schauser. Active messages: a mechanism for integrated communication and computation. In *ISCA '92: Proceedings of the 19th annual international symposium on Computer architecture*, pages 256–266, New York, NY, USA, 1992. ACM Press.

[84] Xbow. Xbow. http://www.xbow.com/.

[85] N. Xu. A survey of sensor network applications. http://enl.usc.edu/ ningxu/papers/survey.pdf.

[86] J. Yannakopoulos and A. Bilas. Cormos: A communication-oriented runtime system for sensor networks. In *The Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, February 2005.