Real-Time Scheduling Strategy for Wireless Sensor Networks O.S

Kayvan Atefi[1], Mohammad Sadeghi[2], Arash Atefi[3]

[1]Faculty of Computer and Mathematical Sciences,UiTM,Shah Alam,Malaysia
k1.educational@gmail.com
[2]Faculty of Computer and Mathematical Sciences,UiTM,Shah Alam,Malaysia
m_sadeghi_3d@yahoo.com
[3]Faculty of Engineering, OloomTahghighatUniversity, Kermanshah, Iran
arash_atefi@yahoo.com

Abstract

Most of the tasks in wireless sensor networks (WSN) are requested to run in a real-time way. Neither EDF nor FIFO can ensure real-time scheduling in WSN. In this paper some characteristics and limitation of operating design in wireless sensor network are discussed. During this paper the researchers will discuss about message scheduling and scheduling strategy for sensor network O.S, for this reason the researchers analysed system architecture of TinyOS, FIFO scheduling TinyOS, task mechanism and event-driven mechanism. Main purpose of these articles is presenting a real-time scheduling strategy (RTS). RTS utilizes a pre-emptive way to ensure hard real-time scheduling. The experimental results prove that RTS has a good performance both in communication throughput and over-load.

Keywords

Wireless sensor networks, Real-Time Scheduling Strategy, Networks O.S

1. INTRODUCTION

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. 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. The realization of these 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 1 show, where operating system stands in the software layers of the WSN. Middleware and application layers are distributed across the nodes as interacting modules. Core kernel of the operating system sits at each individual node.

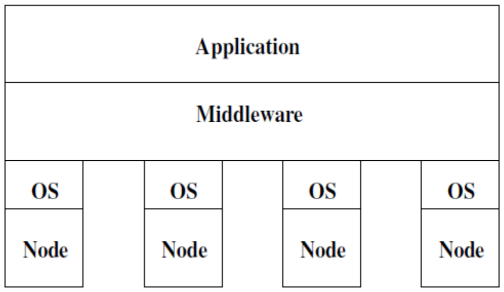


Figure 1: operating system stands in the software layers of the WSN

A common characteristic of many real-time systems is that their requirements specification includes timing information in the form of deadlines. An acute deadline is represented in Figure2.

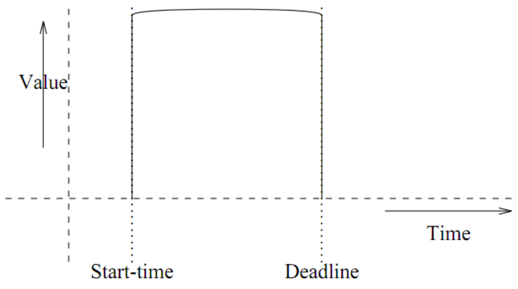


Figure 2: common characteristic of real-time systems

WSN operates at two levels. 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).

According to [1], **Real time scheduling is included**:

- Static table-driven
  - Determines at run time when a task begins execution
- Static priority-driven preemptive
  - Traditional priority-driven scheduler is used
- Dynamic planning-based
  - Feasibility determined at run time
- Dynamic best effort
  - No feasibility analysis is performed

There are some challenges to design an operating system for WSN:

First of all we face restricted resources in wireless sensor nodes to design an operating system. For instance an operating system should provide necessary mechanisms in order to consume the power in optimized way to extend the life of the WSN. Periodic sleeping of sensor nodes is one of the mechanisms to conserve power. The second resource limitation is limited processing power. Operating system should properly schedule the processor according to the priority of jobs. As a second challenge in designing an operating system for WSN, portability should be considered. It means the operating system should be executable on every customized hardware platforms. The operating system should be written in such a way that it is easily portable to different hardware platforms with minimal changes. Multitasking is the 3th challenge. At a given point of time, nodes in the WSN could be doing more than one task.

In wireless sensor networks, sensor nodes are located on remote-site and thus it is very difficult to re-gather them. To update or add a program at run-time of the sensor nodes, sensor operating system must support a dynamic reconfiguration. Many kinds of different mechanisms for reconfiguring sensor nodes have been developed ranging from full image replacement to virtual machines.[4] Cite that, Dynamic reconfiguration in operating system kernels allow modifying a system during its execution, and can be used to implement adaptive systems, dynamic instrumentation and modules. Dynamic reconfiguration is important in embedded systems, where one does not necessarily have the luxury to stop a running system.

This paper proposes a real-time scheduling strategy (RTS) that is includedmessage scheduling and scheduling strategy for sensor network O.S, for this reason the researchers analysed system architecture of TinyOS, FIFO scheduling TinyOS, task mechanism and event-driven mechanism.

TinyOS is anevent-driven operating system designed for sensor-networknodes that have limited memory and computational resources.TinyOS enables developers to access low-level hardware resources at theapplication level, thus resulting in a level of data-acquisitionand communications flexibility that is unavailable to other existing mainstream wireless communications technologies.TinyOS is available open-source and wireless-enabled processor modules that operate with it are commerciallyavailable [6].

On the other hand [3] said TinyOS is designed for Wireless Sensor Network, and it is a lightweight, low-power embedded operating system. The programming language of TinyOS is NesC with modular design method. The use of modular design makes it capable to adapt to the diversity of hardware and makes the applications reuse the general software services and abstract. TinyOS is a typical Wireless Sensor Network Operating System.

Sensor networks are very active research space, with ongoing work on networking, application support, radio management, and security as a partial list. A primary purpose of TinyOS is to enable and accelerate this innovation [7]. Moreover [7] discussed four requirement causes to design of TinyOS that they are include:1- limited resource 2- reactive concurrency 3- flexibility and 4- low power.

2. Related works:

Farshchi S, Nuyujukian P, PesterevA , have developed awireless platform for small, low-power, and low-cost embeddedSensors using COTS microcontrollers and transceivers. They prove thiseffort led to the development of nesC, an extension to theC programming language designed to embody the structuringconcepts and execution model of TinyOS. They defined TinyOS is anevent-driven operating system designed for sensor-networknodes that have limited memory and computational resources(e.g., 8 kB of program memory, 512 B of RAM). They also said,TinyOS enables developers to access low-level hardware resources at theapplication level, thus resulting in a level of data-acquisitionand communications flexibility that is unavailable to otherexisting mainstream wireless communications technologies.

ZHAO Zhi-bin and GAO Fuxiangproposed a real-time scheduling strategy for wireless sensor networks to enhance the communication throughput and reduce the overload. They showed RTS adopts two-layer priority scheduling strategy according to the demand for real-time analysis, and solves the real-time task scheduling problems in WSN commendably.They divided all tasks into two layers and endued diverse priorities. RTS utilizes a preemptive way to ensure hard real-time scheduling. Their experimental results indicate that RTS has a good communication throughput.

3. Design Characteristics

A. Flexible Architecture

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 on the architecture of the operating system. If the architecture allows packing all the required services together into a single system image, then size of the core kernel will increases.

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.

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.

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.

E. Resource Management

Resources available in a typical sensor node are processor, program memory, battery, and etc. Efficient use of processor involves using a scheduler with optimal scheduling policy. Usage of memory involves memory protection, dynamic memory allocation, etc. Figure 3 shows the brief design characteristics of operating system in wireless sensor network.

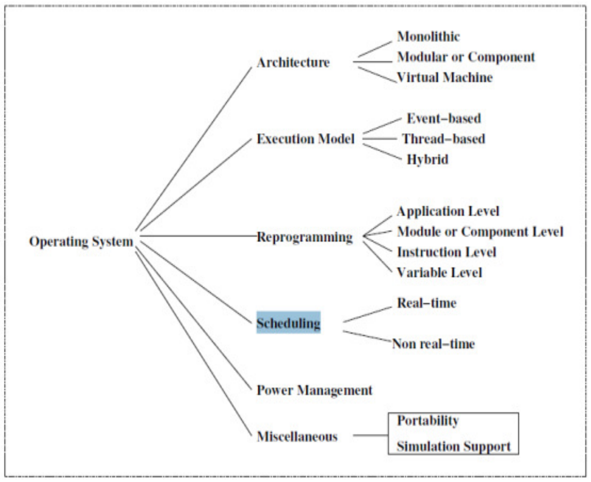


Figure 3: Operating system in WSN

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 non 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. The critical and non critical classification is based on whether the execution of the tasks is in specific time or not.

4. Execution model

Based on the execution model of operating system in wireless sensor network we can classify these operating systems to three base kinds:

- Event-based O.S
- Thread based O.S
- Hybrid.

A. Event-based

TinyOS is an example of event driven operating system which provides a programming framework for embedded systems. It has component-based execution model implemented in nesC. TinyOS concurrency model is based on asynchronous events, tasks and split phase interfaces. 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. TinyOS event driven model has obvious disadvantages like low programming flexibility, non-preemption that are associated with event model.

SOS and EYES are other examples of event driven operating system, that SOS was developed in C and follows event-driven programming model, and EYES that started with the motivation of meeting the goals like small size, power awareness, distribution and ability of reconfigure. It adapted event-driven execution model in order to achieve small size of code and limited available energy.

B. Thread-based

MantisOS is thread-driven operating system model for sensor networks. Thread is a simple computational entity which has its own state. Network stack and scheduler are implemented as threads just like an application. Apart from these threads there is idle thread which runs when all other threads are blocked. 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.

C. Hybrid

CONTIKI is example of hybrid model; 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. Polling mechanism is used to avoid race conditions.

5. MESSAGESCHEDULING

According to message scheduling in operating system in wireless sensor network, SOS uses cooperative scheduling to share the processor between multiple lines of execution by queuing messages for later execution. On the other hand, TinyOS uses a streamlined scheduling FIFO message queue that will be discussed later. This creates a system with a very lean scheduling loop. SOS instead implements priority queues, which can provide responsive servicing of interrupts without operating in an interrupt context and more general support for passing parameters to components. To avoid tightly integrated modules that carefully manage shared buffers, a result of the inability to pass parameters through the messaging mechanism, messaging in SOS is designed to handle the passing of parameters. To mitigate memory leaks and simplify accounting, SOS provides a mechanism for requesting changes in data ownership when dynamically allocated memory is passed between modules.

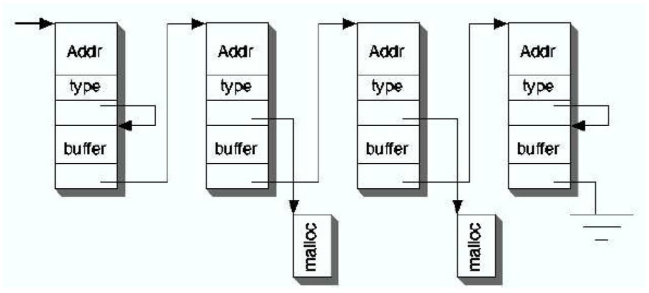


Figure 4: Memory Layout of an SOS Message Queue

Figure 4 provides an overview of how message headers are structured and queued. Message headers within a queue of a given priority form a simple linked list. The information included in message headers includes complete source and destination information, allowing SOS to directly insert incoming network messages into the messaging queue. Messages carry a pointer to a data payload used to transfer simple parameters and more complex data between modules. The SOS header provides an optimized solution to this common case of passing a few bytes of data between modules by including a small buffer in the message header that the data payload can be redirected to, without having to allocate a separate piece of memory.

SOS uses the high priority queue for time critical messages from ADC interrupts and a limited subset of timers needed by delay intolerant tasks. Priority queues have allowed SOS to minimize processing in interrupt contexts by writing interrupt handlers that quickly construct and schedule a high priority message and then drop out of the interrupt context. This reduces potential concurrency errors that can result from running in an interrupt context.

6. SCHEDULINGSTRATEGY

According to application of wireless sensor network, most of the tasks are requested to run in a real-time way. Neither Earliest Deadline First (EDF) nor First in First out(FIFO) can ensure real-time scheduling in WSN. In [3] a real-time scheduling strategy (RTS) is proposed. In this strategy, all tasks are divided into two layers and endued diverse priorities. RTS utilizes a preemptive way to ensure hard real-time scheduling.

A. System Architecture of TinyOS

As I mentioned before TinyOS is developed for the embedded System with high concurrency and is made in NesC language supporting the Wireless Sensor Network architecture. TinyOS adopts event-based concurrency model. Event is corresponding to the emergency case such as external interrupts, and it can preempt tasks or other events and take preference to execute.

In TinyOS there is a widespread use of phased operation, which is dividing the longer operation into some relatively short ones to avoid busy-waiting. The basis of this division is the beginning and end of the operation can be separated in time domain.

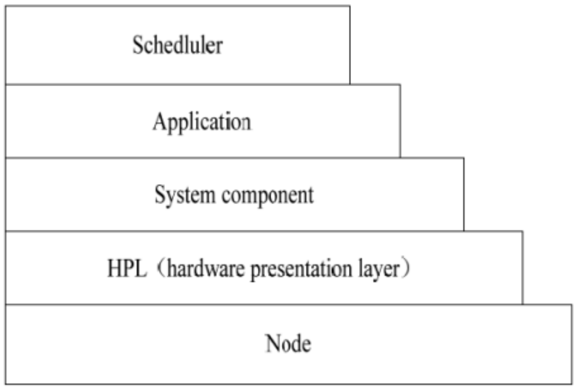


Figure 5: The framework of TinyOS

Figure 5 shows the framework of TinyOS. To providing favourable modular structure that supports the diversity in wireless sensor designing and application, the system is composed by component-based pattern, and primarily consists of master components (including the scheduler), application components, system service components and hardware abstraction components. Hardware abstraction components implement the abstraction of wireless sensor hardware platform, including the sensor subsystem, the wireless communication subsystem, the input/output devices and the power control system on the bottom layer. System Services components are composed of three parts including communication services, sensor services and power management.

B. FIFO Scheduling in TinyOS

In comparison of other category, TinyOS adopts the two-level concurrent models based on the combination of tasks and event-driven.

- **Task Mechanism:**

In TinyOS's task mechanism, we have three basic rules:

First, tasks are equal and there is no concept of priority and no preemption between tasks. All tasks share one executing space, which saves the memory overhead in run time.

Second, Tasks are managed by a circular task queue in system, and the task scheduling follows FIFO mode. Tasks are scheduled by the simple FIFO queue. Resources are distributed before, and currently there can only be seven waiting tasks in the queue. The task-processing model is shown in Fig 6.

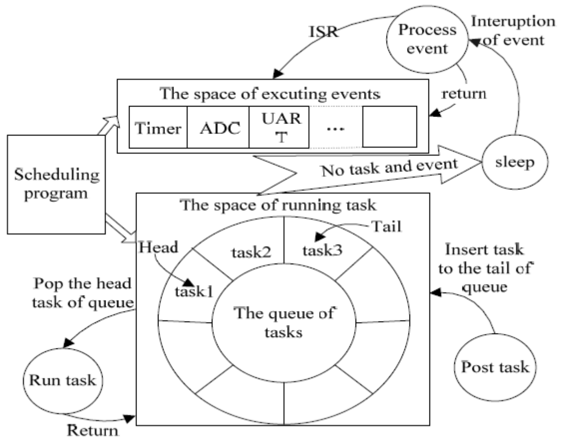


Figure 6: The scheduling strategy of TinyOS

3th, if the task queue is null and there is no events occurring, then the processor will enter into SLEEP mode automatically, and will be woken up by hardware interruption event subsequently. This is conducive to saving energy of system.

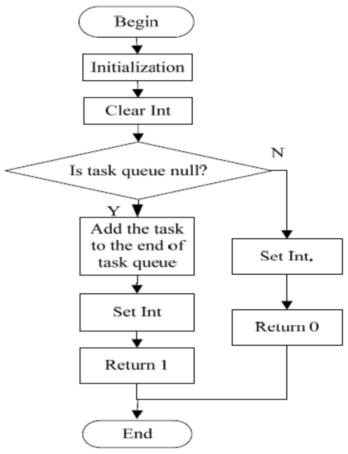


Figure 7: The function of TinyOS_post

Task is defined by user application, and can be created by applications or event handlers. After creating the task, it will be posted to the queue. The procedure is shown in Figure 7. The task scheduler in the core scheduling algorithm returns as soon as it puts the task into the task queue and the task will be carried out. When the task queue is empty, the task can be submitted.

- ***Event-driven Mechanism:***

Events are generated by hardware interruption such as external interruptions and timer interruptions directly or indirectly. When receiving event, TinyOS will execute the event handler corresponding to the event immediately.

Event can preempt the running task. It is an asynchronous, time response fast executive mode. In the TinyOS scheduling mechanism, the task mechanism is not a real-time one. It makes some more important or more real-time tasks not be completed before the deadline and leads to packet-loss, overload, decline of the throughput etc. So it is applicable to non-preemptive, non-time-critical application. Event handler can preempt the current running task and this can be applicable to time-critical application.

- ***Disadvantage of FIFO Scheduling:***

In some situations of this scheduling, TinyOS does not work very well and may present overload, causes the conditions of task-loss, communication throughput declining and cannot guarantee real-time.

In wireless sensor network, when the processing speed of system tasks is lower than the frequency of tasks occurring, the task queue will be jammed up soon. It will lead to task losses. As for the local sensor acquisition rate, we can artificially control, for instance, decreasing the sampling frequency. Occurrence of this phenomenon is mainly due to that packet sending and receiving is restricted to the local tasks. When the occurring frequency of local task is too high, the task queue will be stuffed up soon, and then tasks of transmitting or receiving could be lost, resulting in packet loss.

Generally, the following situations, TinyOS's scheduling strategy may also lead to problems. First, certain tasks (such as encryption and decryption mission in security applications) have very long implementing time. If some real-time missions enter the task queue after the task at this time, the real-time will be affected. For the receiving and transmitting of packets, the baud rate will be affected.

Secondly, when the occurring frequency of local task is high, the task queue will be stuffed up at a short time; other tasks could be lost; besides, if there are many local tasks, this will also affect the normal communications.

finaly, when a certain task in the queue is blocked or performs abnormally because of suddenness, it will affect the subsequent task's running; even will cause the system go down.

C.Analysis on Improved Scheduling Policies in Wireless Sensor Network operating System

According to results of analysis on the simple queue scheduling, it has overload, task loss, low packet throughput, etc. Also, there remains the need to design a multitasking system due to poor throughput and CPU utilization caused by the adoption of single task kernel.

So, to achieve real-time schedule, priority based preemptive scheduling policy is often used. According to application requirements many priority-based multitasking scheduling algorithms are put forward, one of which may be that, for example, each composing phase of the communication route should work timely to ensure other tasks finish properly. Tasks scheduling strategy in WSN will decide whether the nodes finish the tasks in time or not.

Priority-based task scheduling strategy divides tasks into three types: sending data packet, transmitting data packet, and sensing local data according to the functions of different tasks in network. Therefore, it guarantees the more important task to be run in a priority way. Thus, throughput of the system is improved.

This scheduling strategy does not behave well to meet the requirement of real-time. Firstly, it may drop the task before running because the task has exceeded the deadline; secondly, because of non-preemption, the short-time tasks may be blocked to wait for the long-time ones and it leads to the overload for short-time tasks.

Earliest Deadline First (EDF) is widely used in real-time system. Preemptive EDF strategy is the most optimal scheduling for single processor scheduling strategy. That is, if preemptive EDF can't schedule a set of tasks in single processor, other scheduling strategy can't either. Substantively, it is a dynamic process. The algorithm allows a relatively short task to be a preferential one, which makes the system flexible and real-time performance improved.

Rate Monotonic scheduling strategy (RM) is a kind of fixed-priority scheduling strategy. Once the priority of one task is identified according to its periodicity, it will not change with time. A task in smaller periodicity has higher priority. RM can schedule tasks set while other fixed priority strategies can.

Fixed-priority strategy is suitable for wireless sensor network operating system, because it needs to be scheduled one time before running. This fixed-priority will be able to ensure the cyclical behaviour, and the tasks are scheduled only in one queue.

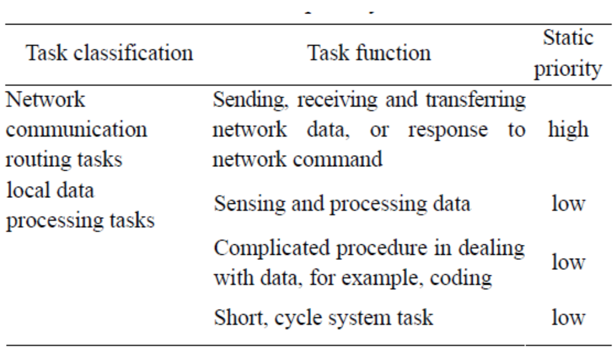
D. Real-time task scheduling

In RTS, tasks are divided into two priorities, static priority and dynamic priority. To ensure real-time and major network packets in the wireless network transmitted reliably. First of all, in accordance with the function of task, it adopts the two relatively static level of priority, which will not change as the time passes, belonging to the fixed priority. Secondly, tasks deadline and run time are the two constraints of dynamic priority, which ensures the reliability of real-time task.

- ***The static priority:***

They divide the tasks into network communication routing tasks and local data processing tasks, and give the tasks two relatively static priorities, high and low. Network communication routing tasks is prior to local data processing tasks. Table 1 shows these priorities

Table 1: Static priority of tasks



Task classification	Task function	Static priority
Network communication routing tasks	Sending, receiving and transferring network data, or response to network command	high
local data processing tasks	Sensing and processing data	low
	Complicated procedure in dealing with data, for example, coding	low
	Short, cycle system task	low

The static priority of tasks is in top-layer priority. Tasks in low or high static priority are set by different priorities in bottom-layer, as shown in Table 2. The tasks with high static priority in their top-layer priority have dynamic priority in their bottom-layer priority, but the ones with low static priority in their top level priority have fixed priority in their bottom-layer priority.

Table 2: Two- layer priority of tasks

Task	Top-layer priority	Bottom-layer priority
Task1	High static priority	Dynamic priority
Task 2	Low static priority	Fixed priority

- **The dynamic priority:**

When new task comes, its attributes such as arriving time, running time and relative deadline will be submitted. They decide task's dynamic priority by the attributes mentioned above. The determination of dynamic priority is shown in Table 3.

Table 3: the dynamic priority of tasks

Task	Running time	Relative deadline	Dynamic priority
Task1	Short	Small	Highest
Task2	Long	Small	Higher
Task3	Short	Large	Lower
Task4	Long	Large	Lowest

- **The fixed priority:**

Different fixed priority is divided according to arriving time, running time and relative deadline. What different from dynamic priority is the fixed priority is determined by execution periodicity and running time for periodic tasks. For non-periodic tasks, the fixed priority is determined by relative deadline and running time. Fixed priority is determined only when tasks are initialized.

Analysing data shows that the utilization ratio of processor is about 100%. EDF is the optimized dynamic, preemptive priority scheduling algorithm for single-processor real-time system, that is, for any real time task set, once there is an algorithm for scheduling, EDF will be there.

E. Taxonomy of real time scheduling:

The blow figure (fig 8) show the taxonomy of real time scheduling

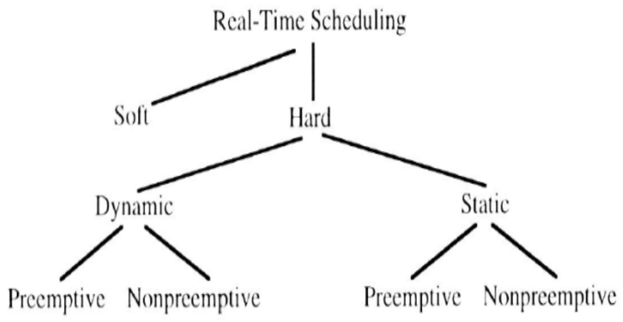


Figure 8: taxonomy of real time scheduling

F. Scheduling strategy for real-time tasks

Scheduling while task queue is empty:

According to figure 8 if there is more space, new task (&task5) will be inserted to queue rear. We don't think about the static priority and use fixed priority to guarantee the task with lower static priority. This schedule will ease the starving case in local data processing. The fixed priority for every task is shown in Table 4 and task queue with fixed priority is shown in figure 9. In the queue, fixed priority is sorted but static priority is not. &task2 arrives thirdly, and &task4 is the second.

Table 4: Two-layer priority of tasks in the queue

task	Static priority	Fixed priority
&task2	1	2
&task1	1	2
&task4	0	3
&task3	1	3
&task5	0	4

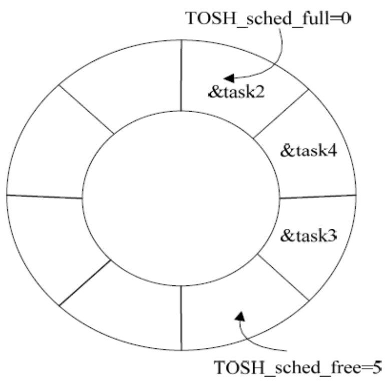


Figure 9: Tasks sorted by fixed priority in the queue

Scheduling when task queue is full:

At first, we need sort the task by static priority. In task queue, the two-layer priority for each task is shown in Table 5. The full task queue with sorted two-layer priority is shown in figure 10. If the two-layer priority of current task is higher than the rear, exchange them. Then, insert the rear task into appropriate location of the queue. At last, abandon current task.

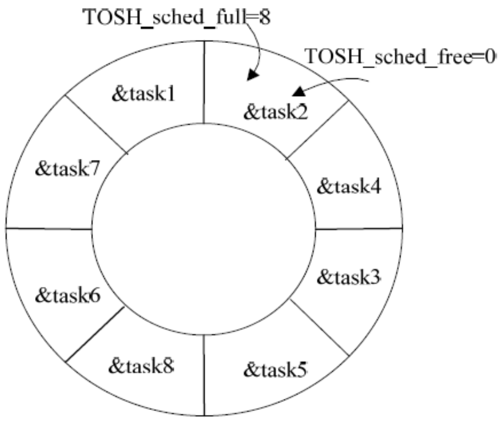


Figure10. Tasks sorted by two-layer priority while queue is full

Table5: two-layer priority of tasks in the queue

tasks	Static priority	Fixed priority
&task2	1	2
&task4	1	2
&task3	1	2
&task5	1	5
&task8	1	4
&task6	0	3
&task7	0	6
&task1	0	7

G. Scheduling of a Real-Time Process

The below figure (figure 11), show an example of scheduling of a real time process

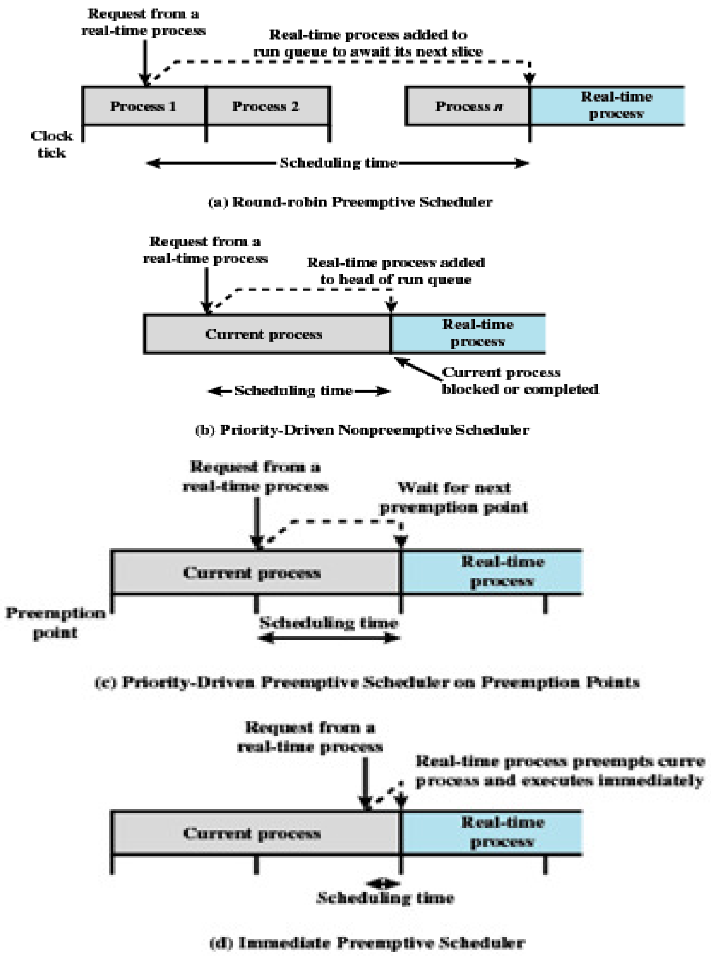

Request from a real-time process
Real-time process added to run queue to await its next slice
Process 1
Process 2
Process n
Real-time process
Clock tick
Scheduling time
(a) Round-robin Preemptive Scheduler
Request from a real-time process
Real-time process added to head of run queue
Current process
Real-time process
Current process blocked or completed
Scheduling time
(b) Priority-Driven Nonpreemptive Scheduler
Request from a real-time process
Wait for next preemption point
Current process
Real-time process
Preemption point
Scheduling time
(c) Priority-Driven Preemptive Scheduler on Preemption Points
Request from a real-time process
Real-time process preempts curre process and executes immediately
Current process
Real-time process
Scheduling time
(d) Immediate Preemptive Scheduler

Figure 11: scheduling of a real time process

H. Evaluation

In order to evaluate RTS (real time scheduling) strategy, the algorithm is simulated, compared with TinyOS's own task scheduling strategy FIFO and task scheduling algorithm EDF is proposed.

The performance is improved significantly for EDF scheduling algorithm. It can ease instantaneous overload phenomenon effectively. Thus, in RTS, task set's dropping rate doesn't go up obviously though the utilization rate of processor rising. As shown in Figure 12, four classes of tasks, which use EDF scheduling strategy, have a little shorter average response time than those who use FIFO. To sum up, RTS guarantees the system good real-time performance.

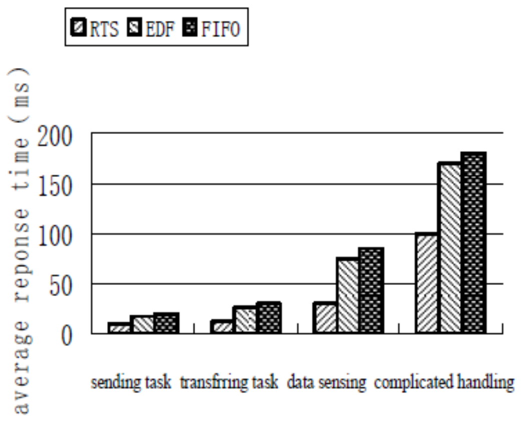


Figure 12: The average response time of different tasks

7.CONCLUSION

Some characteristics and limitation of operating design in wireless sensor network were discussed in this study. In this paper the researchers discussed about message scheduling and scheduling strategy for sensor network O.S, for this reason the researchers analysed system architecture of TinyOS, FIFO scheduling TinyOS, task mechanism and event-driven mechanism.

RTS scheduling strategy proposed in these papers reserves the advantage of high utilization ratio of processor in EDF scheduling strategy. According to RTS, the task with high static priority will be responded and executed prior to other task. At the same time, RTS absorbs the high efficiency of RM in dealing with instantaneous overload problem. Therefore instantaneous overload, which is often happened to low-priority tasks, is solved in RTS.

Through analysis of the experimental results, RTS scheduling strategy has good performance in communication throughout, dropping ratio of overtime task, and response to real-time task. It is valuable in the field of task scheduling in wireless sensor network OS. It is an exploration for WSN real-time research.

Finally, RTS has a good performance both in communication throughput and over-load

8. REFERENCE

[1] N. Audsley and A. Burns," Real-time system scheduling", Department of Computer Science,University of York, UK.,2005

[2] A. Reddy, P. Kumar, D. Janakiram, and G. Ashok Kumar, *Operating Systems for Wireless Sensor Networks: A Survey Technical Report,* May 3, 2007.

[3] Z. Zhi-bin and G. Fuxiang, *Study on Preemptive Real-Time Scheduling Strategy for Wireless Sensor Networks,* September 2009.

[4] J. Polakovic and J. Stefani, *Architecting reconfigurable component-based operating systems,* 15 January 2008

[5] S. Yi , H. Min, Y. Cho, and J. Hong, *An adaptive dynamic reconfiguration scheme for sensor operating systems,* 26 October 2007

[5] Farshchi S, Nuyujukian P, Pesterev A, et al. *"A tinyOS-based wireless neural sensing, archiving, and hosting system", IEEE transactions on neural systems and rehabilitation engineering, vol. 18, no. 2, april 2010*

[6] Werner W, M. Rabaeyj , Emile H. L, " *Ambient intelligence".Nov 9, 2010*

[7] Coleri s , Cheung s.y and Varaiya p ,*"Sensor Networks for Monitoring Traffic"August 5, 2004*