Proceedings of the International Conference on Smart Infrastructure and Construction
ISBN 978-0-7277-6127-9

doi:10.1680/tfitsi.61279.149

Development of RTOS-based wireless SHM system: benefits in applications

Y.G. Fu*, K.A. Mechitov, V. Hoskere, B.F. Spencer, Jr

University of Illinois at Urbana-Champaign, Urbana, USA
* *Corresponding Author*

ABSTRACT The emergence of wireless smart sensor platforms with powerful computational capabilities has had broad impacts in the field of structural health monitoring (SHM), enabling numerous applications in civil infrastructure. However, as the demands of wireless SHM systems increase, certain properties of event-driven operating systems for these sensor nodes, such as the TinyOS operating system of the iMote2, begin to impose limitations on the development of SHM systems. The problematic characteristics include static resource allocation, single-application focus, lack of real-time scheduling support, and dependence on a non-standard programming language. To address these limitations, we consider the use of a real-time operating system (RTOS), commonly used for industrial control systems and similar applications, as an alternative solution in the development of Xnode, the next-generation smart sensor platform for civil engineering applications. In this paper, features of the RTOS environment are first analyzed systematically and compared with TinyOS to demonstrate how it addresses the major concerns of event-driven operating systems. A distributed data acquisition application from the Illinois SHM Services Toolsuite for the iMote2 is implemented as a demonstration of the RTOS-based framework and its advantages. Most importantly, benefits of the RTOS-based wireless SHM system are catalogued comprehensively, with a particular focus on flexible application framework and a more engineer-friendly environment.

1 INTRODUCTION

Structural health monitoring has gained increasing attention, as it helps to address the major concern about safety of aging infrastructure in our society. Since the 1990s, researchers have made continuous efforts to develop a series of wireless or smart sensor platforms to significantly facilitate damage detection diagnosis of world's structures (Spencer et al., 2004). To date, most of wireless platforms adopt event-driven operating systems to manage computing resources and host applications. Typically, TinyOS is among the most popular event-driven operating systems, which is designed for wireless platforms with extreme limited resources (Levis et al. 2005). In particular, iMote2, one of the most widely-used sensor platforms for data intensive applications such as SHM, is supported by TinyOS, and it has enabled the implementation of world's largest wireless smart sensor network (WSSN) on the Jindo Bridge in South Korea (Rice et al., 2010).

However, lessons learned from experience of iMote2 reveal several limitations of TinyOS (Rice & Spencer, 2009). The problematic characteristics include static resource allocation, single-application focus, lack of real-time scheduling support, and dependence on a non-standard programming language, etc. Of these limitations, concurrency model and programming efforts are considered as the two major concerns:

(i) There are only two level of executions in the concurrency model of TinyOS: tasks and events. Tasks are executed in a First In, First Out (FIFO) manner. Therefore, real-time applications are difficult to be realized, since critical tasks may be delayed by execution of previous tasks. Besides, uncertain delay of task executions is inevitable in some cases. As a result, the applications of TinyOS-based systems are limited.

(ii) Applications are developed in nesC, a dialect of C programming language, for TinyOS. The complex syntax and semantics in nesC makes it a challenge for engineers to develop applications. Besides, modeling nesC program requires involvement of hardware operations in most cases, which makes it complicated to update or debug code (Ammari, 2013). In addition, the single-application focus of TinyOS requires developers to consider every part of a system every time their code is updated. All of these dramatically increase programming efforts.

In order to address these concerns, we consider the use of a real-time operating system, commonly used for industrial control systems and similar applications, as an alternative solution in the development of Xnode, the next-generation smart sensor platform for civil engineering applications. In this paper, the RTOS environment is first analyzed and compared with TinyOS to demonstrate how it addresses the major concerns of event-driven operating systems. A distributed data acquisition application from the Illinois SHM Services Toolsuite for the iMote2 is implemented as a demonstration of the RTOS-based framework and its advantages. Most importantly, benefits of the RTOS-based wireless SHM system are catalogued comprehensively, with a particular focus on flexible application framework and engineer-friendly environment.

2 REAL-TIME OPERATING SYSTEMS

2.1 *Introduction to RTOS basics*

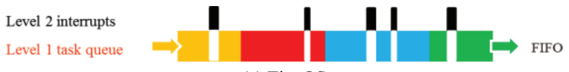
An RTOS is an operating system in which applications are run under specified time constraints with high reliability. Typical properties of an RTOS includes multi-tasking and preemptive scheduling. In general, an RTOS problem is divided into multiple tasks and handled by several processes. A scheduler is used to manage task sequencing at run time, based on specific scheduling algorithms.

There are many different RTOS that are used in the embedded systems market. FreeRTOS is selected as for development of Xnode due to its open source nature, portable C-language implementation, high degree of configurability, availability of free and commercial support options, and a large user community. Specifically, it has typical advantages as a class of RTOS, compared to event-driven operating systems, such as real-time scheduling and efficient inter-task communication. Also, it has specific advantages of small footprint, low overhead and fast execution, compared to some other classes of RTOS.

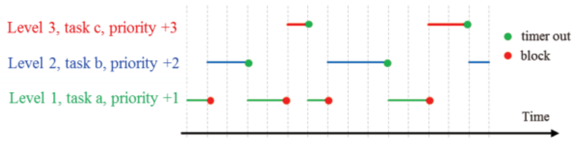
2.2 *Primary solution of FreeRTOS for Xnode*

FreeRTOS provides a priority-based preemptive scheduler, as well as inter-task communication and coordination tools (e.g., queues, mutexes and semaphores). It can efficiently address the limitations of event-driven operating systems on a low-power microprocessors. Primary solutions of FreeRTOS for Xnode include scheduling flexibility and programming efficiency, which are demonstrated through comparison between RTOS-based systems and TinyOS-based systems.

1) As shown in Figure 1, there are only two levels of concurrency in TinyOS, namely the task and interrupt contexts. Different tasks are managed within a single FIFO queue with no reordering or task priorities. Specifically, new tasks are put at the end of the task queue upon scheduling, and they cannot execute until all previous tasks are completed. In this way, some important tasks will be delayed by less critical tasks, for example a high-frequency sensor sampling task can be delayed by a periodic task that checks the battery level, introducing jitter into the sampling process. This is not desirable for real-time applications. By contrast, in FreeRTOS, important tasks can be assigned higher priorities. Tasks with lower priorities will be suspended to give way to important tasks and later be resumed after the important tasks are completed or blocked. This scheduling flexibility will significantly enable complex applications in wireless SHM systems, such as structural control, particularly with multiple applications coexisting on the same wireless network.



(a) TinyOS


Level 3, task c, priority +3
Level 2, task b, priority +2
Level 1, task a, priority +1
timer out
block
Time

(b) FreeRTOS

Figure 1. FIFO scheduling in TinyOS and priority-based scheduling in FreeRTOS.

2) Another major concern that FreeRTOS addresses is programming efficiency. FreeRTOS is predominantly written in standard C programming language, and hence developers can freely use many existing C

libraries, which implement useful functionality such as numerical algorithms and data compression, with much less porting efforts than that in TinyOS-based systems (written in nesC, which is a custom extension on the C language). Also there are several powerful tools written for the C language that can facilitate testing and debugging user-developed applications. In addition, the single-application focus of TinyOS, where application code, operating system code, libraries and device drivers are all compiled into a single image and share a single scheduler with no priority differentiation, which means that developers have to consider every part of a system and possible interactions of all components. By contrast, functionalities of a program in RTOS-based systems can be separated by different tasks and priority levels. Hence, developers just need to modify specific part of the code, rather than consider everything in a program. As a result, the cost and complexity of application development can be significantly reduced.

3 DEMONSTRATION: REMOTESENSING APPLICATION IN FREERTOS

3.1 *RemoteSensing application in brief*

The Illinois Structural Health Monitoring Project (ISHMP) Services Toolsuite, developed through collaboration between researchers in Smart Structures Technology Laboratory and Open Systems Laboratory at University of Illinois at Urbana-Champaign, is the software foundation for the iMote2-based remote sensing platform, which is built on TinyOS. RemoteSensing, a fundamental distributed data acquisition application in ISHMP Services Toolsuite (Rice & Spencer, 2009), is implemented in FreeRTOS as a demonstration of the RTOS-based framework and its advantages (Figure 2). Specifically, this application is used to help the coordination between gateway node and multiple sensor nodes to realize the remote sensing application.

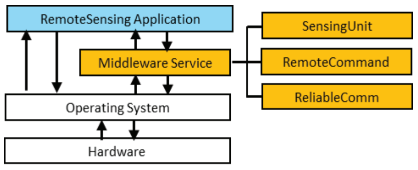


Figure 2. Structure of the RemoteSensing application, which is supported by three middleware services.

3.2 *Sensing application framework*

During RemoteSensing application, sensor nodes are required to start sensing when they receive commands from a gateway node, and to send sensor data back to the gateway node once the sensing is completed. To illustrate the flowchart of RemoteSensing in a single sensor node, a basic sensing application framework is developed in FreeRTOS (Figure 3). In the proposed framework, three tasks are defined to realize the sensing process, namely the Application Task, Sensing Task, and Radio Task. Commands are first received by the Radio Task and delivered to the Application Task. Then, the Application Task executes the commands and activates the Sensing Task to control sensors to generate sensor data periodically. Once sensing is completed, the Application Task gets sensor data from the Sensing Task and transfers it to the Radio Task for transmission back to the gateway node. During this process, inter-task communication and coordination is enabled by means of queues and semaphores. Meanwhile, different priorities are assigned for deciding which task should be executing at a particular time. Particularly, the Radio Task should be assigned a high priority, because any delay of the Radio Task may result in missed data packets that are transmitted from the gateway node.

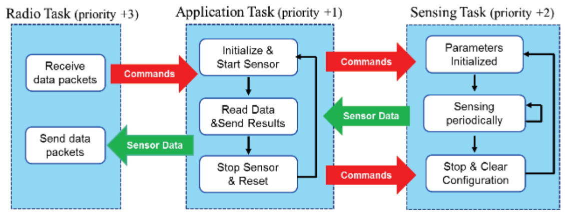


Figure 3. Flowchart of the sensing process in a sensor node in the FreeRTOS version.

3.3 *SensingUnit Implementation*

SensingUnit (SU) is a service component which governs the coordination between a gateway node and multiple sensor nodes during a sensing process (Sim & Spencer, 2009). It is used to transfer sensing information, initiate synchronized or unsynchronized sensing and transmit measured data. A series of states are defined using a state machine to facilitate the control of performance of different nodes in a sensor network. Basically, SU consists of two parts: SensingUnit GatewayNode (SUGW) and SensingUnit SensorNode

(SUSN), which are designated to serve a gateway node and a sensor node respectively.

Figure 4 shows the flowchart of SensingUnit in FreeRTOS, as well as its connection with the Application Task in a gateway node (GatewayNode_app) and with the Application Task in a sensor node (SensorNode_app). The main logic of SU in FreeRTOS remains the same as that in the TinyOS version, but interfaces of SU are modified significantly. Upon receipt of a command from GatewayNode_app and SensorNode_app, SUGW and SUSN will change its state to INIT in the initialization. Then GatewayNode_app will check each state of SUGW and trigger the transition between different states based on results of each iteration in a state machine. Sensing command will be transmitted from SUGW to SUSN by means of radio communication. Once the sensing process is completed in the sensor node, GatewayNode_app will directly send the data request command to SensorNode_app to retrieve the sensor data.

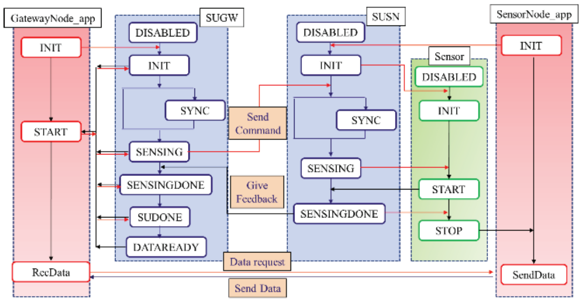


Figure 4. Flowchart and implementation of SensingUnit in the FreeRTOS version.

3.4 RemoteCommand Implementation

RemoteCommand is an efficient messenger for delivering commands, response or measured data between the gateway node and sensor nodes. It consists of several commands and event handlers, which are used in the designated order (Mechitov, 2012).

In the RemoteSensing application, RemoteCommand is applied to deliver sensing commands from SUGW to SUSN before sensing process, and it is also used to send data requests from the Application Task in the gateway node (Main_app_GW) to the Application Task in sensor nodes (Main_app_SN) after sensing is completed (Figure 5).

Figure 6 shows the flowchart of RemoteCommand in FreeRTOS. Main commands maintain the same interfaces as in the TinyOS version, but event handlers are replaced with an additional task with similar functionalities. At the beginning, sensor nodes will call registerCommand to specify a start function. Then executeCommand is called in a Gateway node to deliver commands. Once the commands are received by the RemoteCommand Task in sensor nodes (RecTask_SN), the corresponding start function is called in a designated way. After that, executionDone is called to send working results back. Upon RemoteCommand Task in the gateway node (RecTask_GW) receiving the data message, a callback function is executed and the next command is ready.

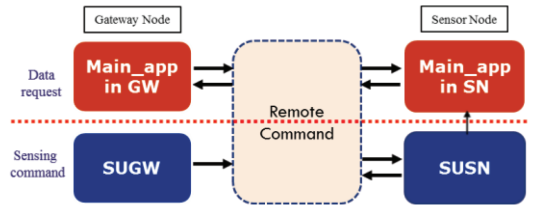


Figure 5. Implementation of RemoteCommand in RemoteSensing.

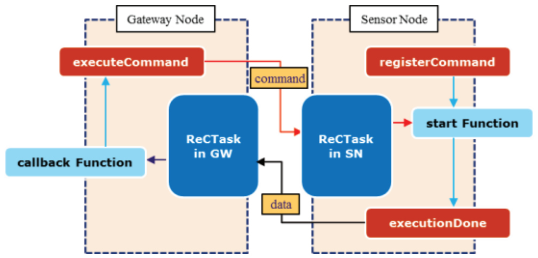


Figure 6. Flowchart of RemoteCommand in FreeRTOS.

3.5 ReliableComm Implementation

RelibleComm is a foundation service to address the concern of data loss in wireless systems, by means of acknowledgement-based reliable communication protocols (Nagayama & Spencer, 2007). The sender sends a message to the receiver through a series of data packets. At the end of data transmission, the receiver checks the number of packets and send acknowledgment back to the sender, requesting the sender to resend missing packets. After that, the sender will keep sending missing packets until it receives the acknowledgement that all packets are received.

Figure 7 shows the flowchart of ReliableComm in FreeRTOS. The Radio Task is implemented to receive data packets continuously in both the sender and the receiver. Two states, COMM and CHECK, are defined in a state machine to control the behavior of ReliableComm. Specifically, in the COMM state, data packets are transmitted between the sender and the receiver, while in the CHECK state, an acknowledgement is sent from the receiver to the sender, requesting missing packets. As shown in Figure 8, ReliableComm consists of several interfaces which are used to connect with RemoteCommand and the Radio driver.

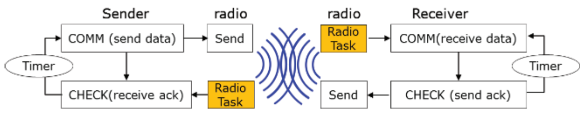


Figure 7. Flowchart of ReliableComm in FreeRTOS.

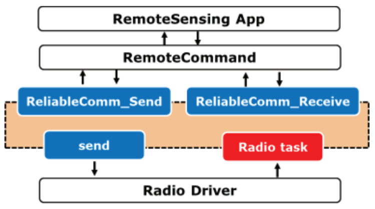


Figure 8. Implementation of ReliableComm in RemoteSensing

3.6 *ReliableComm Implementation*

A total of 4 types of tasks are defined for implementation of RemoteSensing in FreeRTOS, assigned with different priorities (Figure 9).

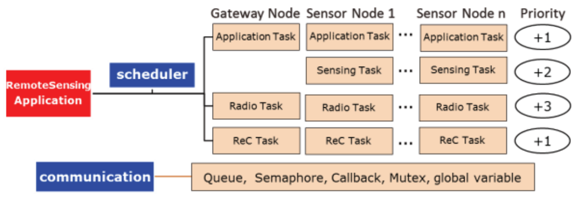


Figure 9. Structure of RemoteSensing in summary.

Specifically, the Radio Tasks have highest priorities to ensure the efficient communication between a gateway node and sensor nodes. Meanwhile, several tools are applied to enable inter-task communication, including queues, semaphores, callback functions, mutexs, etc. The diagram in Figure 10 demonstrates how the tasks in RemoteSensing are scheduled by FreeRTOS. Using FreeRTOS, different functionalities in RemoteSensing are isolated within separate tasks, and these tasks are executed efficiently, by means of a pre-emptive, priority-based scheduler.

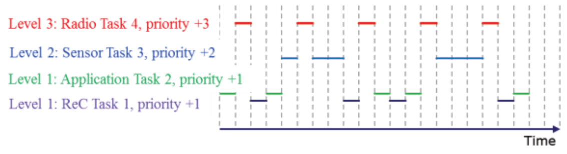


Figure 10. Tasks scheduled in the FreeRTOS version of RemoteSensing.

4 BENEFITS IN APPLICATIONS

4.1 *Flexible application framework*

One primary benefit of Xnode, a RTOS-based wireless SHM system, is the more flexible application framework. In moving to this new platform, the RTOS enables several benefits for application developers compared to the iMote2 (ISHMP/TinyOS-based) system, in part due to its real-time scheduler.

1) Over-the-air programming (OTA) can be realized by uploading a single module or the entire application image, which is a significant benefit for SHM applications. In general, wireless sensor nodes are densely deployed in a SHM project. If pre-installed damage detection algorithm is obsolete, it would be of great convenience to reprogram wirelessly for multiple sensor nodes, rather than to collect sensor nodes manually and update algorithms one by one.

2) Resource optimization is another attractive aspect. Developers can configure the system with dynamic resource allocation mechanism, hence resources can be allocated and deallocated at run-time. In this way, highly efficient operation is enabled for long-term structural health monitoring, especially for a large-scale wireless sensor network.

3) Real-time sensing and control applications are possible to be handled on the basis of the real-time scheduler. Specifically, different functionalities are defined by multiple tasks, and the task with the strictest deadline is assigned the highest priority, specified with maximum time spans. This is highly beneficial for real-time application development, such as structural control using wireless sensor nodes, in which time delay may result in serious detrimental consequences.

4) In addition, ability of SHM systems will be enhanced to capture extreme events, which is known as rare event detection. Because of strict timing control supported by RTOS-based system, uncertain delay in the start of sensing can be possibly avoided. As a result, real-time triggering of the wireless sensor network is feasible to capture earthquakes and impact events, as duration of those events can be rather short.

4.2 *Engineer-friendly environment*

Another attractive benefits of Xnode is the more engineer-friendly environment. In the field of SHM, most users are civil engineers who do not have a solid foundation of computer science knowledge or programming skills. This platform is attractive for those users, because of its more common programming language and stronger separation of concerns.

1) The C and RTOS-based system is easier for engineers to develop applications than TinyOS-based systems. Beginners will face a steep learning curve for programming on RTOS, using a standard programming language. However, in TinyOS, users have to understand the custom nesC extensions as well, which require deep understanding of the nesC/TinyOS concurrency model.

2) Some useful tools, such as an interactive debugger can help users to build their applications efficiently. In TinyOS-based systems, if users want to develop the RemoteSensing, they probably have to compile and install a test application in wireless sensor platforms every time when the code is updated or debugged. In particular, approaches to check if the code works well are limited (e.g., LED indicators are overly relied on in iMote2 development). However, development cost and complexity will be significantly reduced when developing the RemoteSensing application in RTOS-based systems.

3) RTOS-based systems provides an isolation of individual tasks and hence a separation of concerns. Application designers can focus on their specific parts without affecting the timing of other code. This is really beneficial for engineers. For example, algorithm development for damage detection is a main topic in the field of SHM, and implementation of algorithms in wireless SHM systems is always a major obstacle for non-programmers. In this case, RTOS-based systems will lower the barriers for people to enter application development in the field of SHM.

5 CONCLUSIONS

Primary limitations of event-driven operating systems for development of SHM systems are demonstrated, i.e., the concurrency model and inordinate programming effort. They are addressed by using FreeRTOS as alternative solution in the development of Xnode, due to its open source, portability, flexible scheduling and strong support for real time application. RemoteSensing application is implemented as a demonstration of RTOS-based framework and its advantages. Specifically, a basic sensing application framework is developed, and three middleware services (SensingUnit, RemoteCommand and Rliable-Comm) are implemented in FreeRTOS. Besides, great benefits from RTOS-based wireless systems are analyzed, with a particular focus on flexible application framework and engineer-friendly environment.

REFERENCES

Ammari, H.M. 2014. *The art of wireless wensor networks*. Springer, Berlin, Heidelberg.
Levis, P., Madden, S., Polastre, J., Szewczyk, R., et al. 2005. TinyOS: an operating system for sensor networks. *Ambient Intelligence*, Weber, W., Rabaey, J.M., Aarts, E., Eds. 115-148, Springer, Berlin, Heidelberg.
Mechitov, K.A. 2012. *A service-oriented architecture for dynamic macroprogramming of sensor networks*, Ph.D dissertation, University of Illinois at Urbana-Champaign.
Nagayama, T. & Spencer Jr, B.F. 2007. *Structural health monitoring using smart sensors*. Newmark Structural Engineering Laboratory. University of Illinois at Urbana-Champaign.
Rice, J.A., Mechitov, K., Sim, S.H., et al. 2010. Flexible smart sensor framework for autonomous structural health monitoring, *Smart Structures and Systems* **6**, 423-438.
Rice, J.A. & Spencer Jr, B.F. 2009. *Flexible smart sensor framework for autonomous full-scale structural health monitoring*. Newmark Structural Engineering Laboratory. University of Illinois at Urbana-Champaign.
Sim, S.H. & Spencer Jr, B.F. 2009. *Decentralized strategies for monitoring structures using wireless smart sensor networks*. Newmark Structural Engineering Laboratory. University of Illinois at Urbana-Champaign.
Spencer Jr., B.F., Ruiz-Sandoval, M & Kurata, N. 2004. Smart sensing technology: opportunities and challenges, *Structural Control and Health Monitoring* **11**, 349-368.