Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 5 (2011) 264–271

Procedia Computer Science

The 2nd International Conference on Ambient Systems, Networks and Technologies (ANT)

Dynamically Reconfigurable Energy Aware Modular Software (DREAMS) Architecture for WSNs in Industrial Environments

Ahmad El Kouche, Louai Al-Awami, Hossam Hassanein

School of Computing, Queen's University, Kingston, ON, Canada

Abstract

This paper describes a dynamically reconfigurable energy aware modular software (DREAMS) architecture to facilitate software development for energy harvesting Wireless Sensor Networks (WSNs) in harsh industrial equipment monitoring applications. The DREAMS architecture simplifies WSN software development for applications with strict requirements for energy harvesting as an alternative power source to conventional batteries. Each software module has a set of features including an energy stamp (ES) and associated priority level (PL) required for the module execution management unit (MEMU) to carry forward a dynamic execution model in compliance with the current harvested energy level. In order to accommodate various design requirements addressing inherent difficulties in industrial equipment monitoring applications, software modules can be selected and configured at compile-time or remotely upgraded at run-time based on the low energy stamps or stringent application requirements. Loosely coupled software modules add increasing level of software architecture flexibility and dynamic configurability while allowing for optional optimizations for specific aspects such as reliability, security, power savings, or time critical event reporting. DREAMS architecture was successfully prototyped to monitor real-time eminent faults present in the ligaments of large vibration screens used in the filtration process of the Oil Sands located in North Alberta, Canada.

WSN; Modular; Software Architecture; Energy Harvesting; Industrial; Harsh Environment; Equipment Monitoring; Oil Sands;

1. Introduction

Industrial production facilities constantly look for new cutting edge technologies capable of improving production quality while reducing production cost. Production efficiency is directly impacted by downtime due to periodic maintenance, manual checkups, or failure recovery of heavy loaded machinery or equipments. In addition, the common nature of industrial environments is the harsh factor that is imposed on both the monitoring system and the human operator, such that post deployment is physically impractical for human intervention, which is a situation that necessitates energy harvesting for continuous monitoring operation. Therefore, Wireless Sensor Networks (WSN) are a highly anticipated and attractive technology for industrial facilities to close the gap between monitoring difficulties in harsh environments, associated costs, and low energy footprints. In this paper, we introduce the DREAMS architecture to address software modeling difficulties for applications that are battery-less or passive

* Corresponding author. Tel.: "+1-613-533-6000 ext. 78232" ; fax: +1-613-533-6513 .
E-mail address: elkouche@cs.queensu.ca (A. El Koushe) louai@cs.queensu.cs (L. Al-Awami) hossam@cs.queensu.ca (H. Hassanein)

doi:10.1016/j.procs.2011.07.035

powered, such that energy harvesting is the main power source for the WSN. Furthermore, DREAMS architecture is mainly attractive for applications requiring easy to use dynamic re-configurability of software modules and associated priorities with optional over-the-air infrequent software module upgrades for hard to reach deployments.

2. Related Work

Energy harvesting based WSNs have seen a wide range of applications in different monitoring environments **[1][2][3]**. In addition, many studies address the software aspects of energy harvesting based WSN. In **[4]**, a programming language called EON and a runtime system are presented for perpetual system. The concept is to allow the programmer to annotate different flows for different energy levels. During execution of the program, the runtime system switches to different flows based on the energy level. Besides, the runtime system adapts the frequency of execution by providing a higher or reduced level of service based on the energy level. The goal of the solution is to sustain the working state of the system at all times. This work differs than ours, such that our system exhibits intermittent operation and tries to execute the more urgent tasks first rather than maximizing system life time. In addition, instead of annotating different command flows, we allow for specifying the sequence of modules which simplifies the program job. The work in **[5]**, discuss Levels, which is a software abstraction strategy aiming at increasing network lifetime for applications with predictable lifetime. However, the scheme does not consider dynamic energy harvesting.

The authors in **[6]** discuss a power management scheme for energy harvesting based WSN. The paper presents a power management scheme that generates a profile by learning from energy harvesting history, and adapts operation to the predictable rate of energy as to sustain a continuous operation. The scheme is also extended to the case of a network where different nodes have different potential for energy from the environment. The later extension includes a routing protocol with load balancing mechanism that shifts the load according to the energy availability of different nodes in the network. In **[7]**, the authors provide a decision engine for maximizing the number of tasks performed at each execution round. The engine takes as input a set of energy prediction for each task and an energy budget and generates the list of tasks to be executed. The object is to maximize the number of tasks for each execution cycle. Our work differs in the sense that DREAMS architecture tries to satisfy tasks according to their priorities rather than maximizing the number of tasks.

3. DREAMS Architecture

The core of the system architecture is designed for ease of use, flexibility, dynamic over-the-air reconfiguration, low cost, rugged, and self sustainable power management utilizing energy harvesting as an alternative energy source to conventional battery dependent architectures. The motivation of the software architecture is to easily allow the reusability and upgradability of energy efficient software modules into a broad range of applications in harsh operating conditions where post deployment physical accessibility is impractical or impossible, thus, energy harvesting is the only source of available energy. Therefore, we introduce DREAMS architecture which takes into consideration the rate at which energy is being harvested and splits the entire firmware into manageable loosely coupled software modules that formulate the building blocks of the entire application.

The DREAMS architecture is made up of loosely-coupled reusable software modules. A software module is a set of functionalities that serve a particular purpose from as little as reading a port and saving the value to a memory location to as sophisticated as encrypting and reporting a packet. The meaningful combination of software modules executed in a particular logical sequence gives rise to a meaningful application. In other words, the software modules are the building blocks of the application which can be dynamically reconfigured based on the energy required per software module at runtime. Energy requirement per module plays an important role in deciding which modules should be included, executed, or requires further optimization for energy efficiency. Therefore, it is recommended to design software modules that are relatively small in memory size, fast to execute, with low energy profiles, and functionally orthogonal in operation from other modules as possible.

Each software module has a set of associated attributes such as energy stamp (ES), priority level (PL), memory space requirement (MS), hash code module identifier (HC), and execution order (EO). The energy stamp of a module determines the average amount of energy required to execute the software module on the sensor node from start to finish. The energy stamp may be specific per hardware platform per operating frequency, or more general such as the number of instructions in a software module scaled per MIPS and associated system operating voltage. We recommend calculating an average energy stamp for every module per used platform prior to deployment, which is a one time job per module. We will later discuss a method on how to calculate energy stamps for a given software module. The priority level of the software module indicates the relative importance of this module to other software models, which is a value that is dynamically promoted or demoted due to various reasons such as user feedback or the occurrence of a crucial event such as a sensor reading reaching a critical threshold level. The execution order of a software module determines the relative execution position of the software module in the execution list. The memory space requirement is the amount of ROM and RAM required by each module. Specifically, the ROM memory space requirement plays an important role in over-the-air module upgrade, such that the amount of harvested energy required and associated time delay is proportional to the ROM requirement of the module. Therefore, it is crucial to maintain a relatively small ROM footprint to avoid long wait times for module upgrades due to the slow process of energy harvesting. The hash code module identifier of the software module serves as the module identifier. Since most modules will be potentially orthogonal in functionality, the hash codes generated are unique and simple methods of identifying and distinguishing different modules.

3.1. Source Node DREAMS Architecture

The core of the DREAMS architecture resides within the software operation of the source nodes. The architecture consists of a pool of modules which contains all the possible modules needed by the source node for a particular application, see Fig. 1. The *module pool* should maintain a fair amount of independence and loose coupling between the functionality of various modules while including optional modules that could be used if extra harvested energy is made available such as the debug set of modules. There are no restrictions on how comprehensive or specific a single module can be, however, there are tradeoffs for each case. Having a single module largely comprehensive such as acquiring a sensor reading and reporting may increase compiled memory efficiency and hence energy, however, future upgrades of that module require the complete recompilation and transmission of the module which will in turn require future additional energy to upgrade. Smaller memory sized modules tend to be less memory efficient, thus, consume more energy accumulatively than a single module. However, greater amount of flexibility is possible by connecting and inter mixing smaller modules together to attain a variety of comprehensive tasks with little time spent over-the-air for future module upgrades. During compile-time, software modules are compiled and loaded into well known and organized memory sectors of the ROM, such that every module is given a set of default features and identifiers such as energy stamp (ES), priority level (PL), execution order (EO), hash code (HC), and memory space (MS), which are utilized by the CMMU. The *module update* takes care of updating modules or installing new modules, such that future application updates are possible by either sending new modules or patched modules over-the-air to replace older modules or to be stored on available free module sectors in ROM while preserving older modules in case a roll-back is necessary. The *EH monitor and control* provides the CMMU with the current energy level harvested, and it is also responsible for any control related to EH such as switching on additional capacitor banks. Energy harvesting hardware is not restricted to any particular type as long as the generated power P_{out} is greater than the minimum input power required to sustain the normal sleep state of the source node, such that a positive accumulated energy is produced over time.

The CMMU is responsible for management of the execution modules according to their priorities and energy stamps as well as the energy available. The modules are assumed to be ordered according to a logical order. For example, sense → encrypt → report → receive. This sequence implies the logical order in which modules should be executed. In addition, every module has an execution level (L) and an energy stamp (ES), see Fig. 2a. The execution level reflects the level of priority of the module, while the energy stamp reflects the energy needed to execute the module. Modules with the same level L are strictly executed together according to the execution order.

In every execution cycle τ, the CMMU evaluates the available energy E. Then, it finds the highest value of $L = \alpha$, such that the sum of all ES of all modules belonging to $\lneqq \alpha$ is smaller than E. The list of modules chosen is then executed according to the execution order. Fig. 2b shows the decision making process at the CMMU to generate the list of tasks to be executed. For example, let's assume that a basic sensor can do sensing, encryption, and reporting, in that order. Also, assume that sensing and reporting corresponds to L=1, while encryption has L=2. If the energy available E can satisfy L=1 only, the node senses and then reports without encryption. On the other hand, if the energy available can satisfy both the modules in L=1 and 2, then the node would sense, encrypt, and report. If the energy available does not allow for any level of execution to be met, the sensor would go back to sleep.

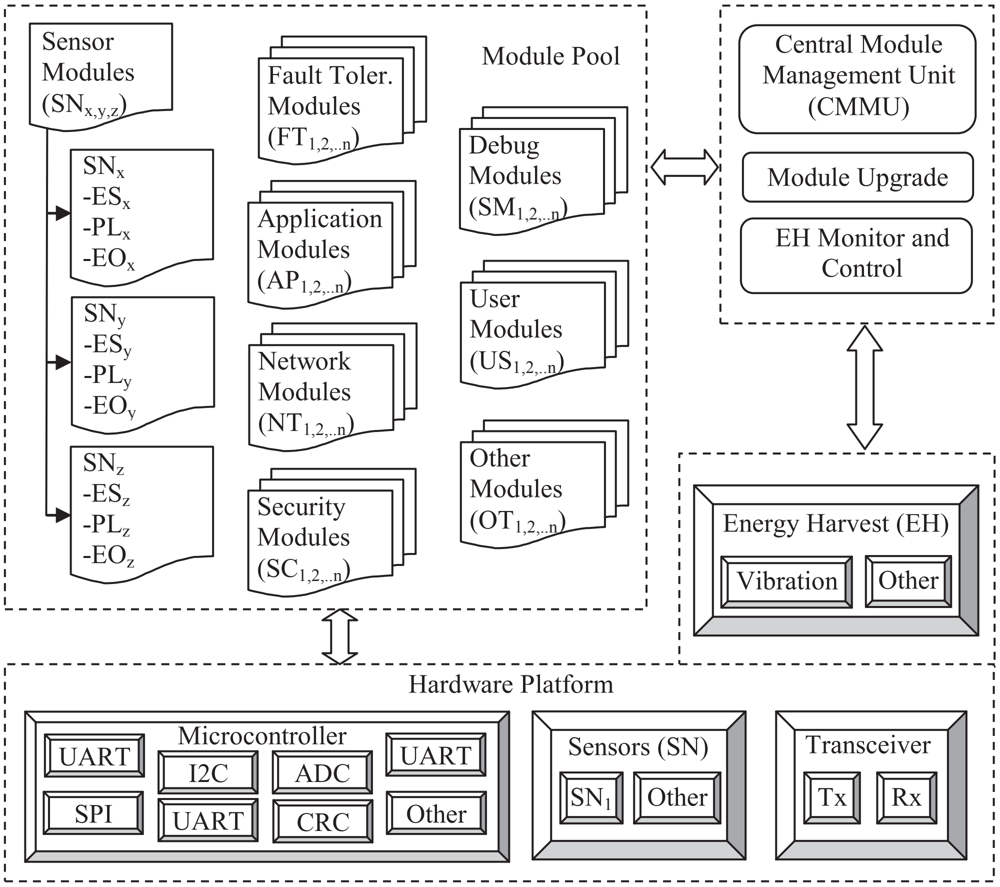

Fig. 1. DREAMS Architecture.

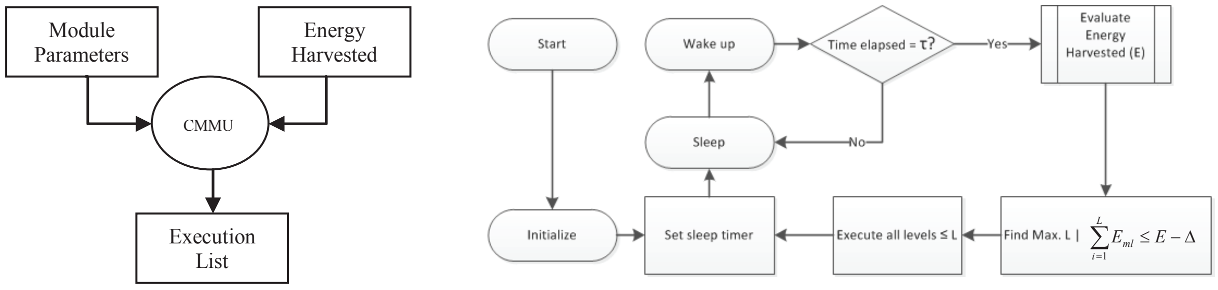

Fig. 2.(a) CMMU Input and Output; (b) CMMU Module Execution List Processing.

3.2. Sink Node software Architecture

The sink node is responsible for decrypting the packets received from the sensors and relaying the packets to the Data Management Centre (DMC). It also analyzes the performance of the source nodes by processing some performance indicators which are sent by the source node. In addition, the sink node coordinates the Medium Access Control (MAC) between the source nodes in the network. Without loss of generality, the discussion of the architecture assumes a single-hop between sink and source nodes due to the restrictive amount of harvested energy in our targeted application. However, the proposed software architecture can be applied to multi-hop networks if enough energy can be harvested to support relaying or routing. To facilitate fault tolerance, redundant sink nodes can be utilized in addition to multiple receivers per sink. The sink nodes exchange heartbeat messages to indicate live activity. The sink nodes work in an active or passive mode meaning that only one sink is active at a time. When a failure occurs, the passive sink node detects the failure and assumes the role of the active sink.

Since the sink node executes multiple tasks with no energy constraints, a sink node can run a full OS to aid in reusing the standard services, such as memory management, hardware port interfacing, Ethernet or WiFi communication, fault-tolerance, etc. The additional proprietary functions such as MAC coordination, data processing and forwarding, and network updates and configuration can be easily run on top of the OS. Figure 3 shows a general architecture for the sink node. The choice of an OS based sink node aids in simplifying the implementation as well as avoiding the need to reinvent the wheel, especially for common tasks. There exists many embedded OS that can be used for that purpose, such as TinyOS **[8]**, or other general purpose OS such as Linux.

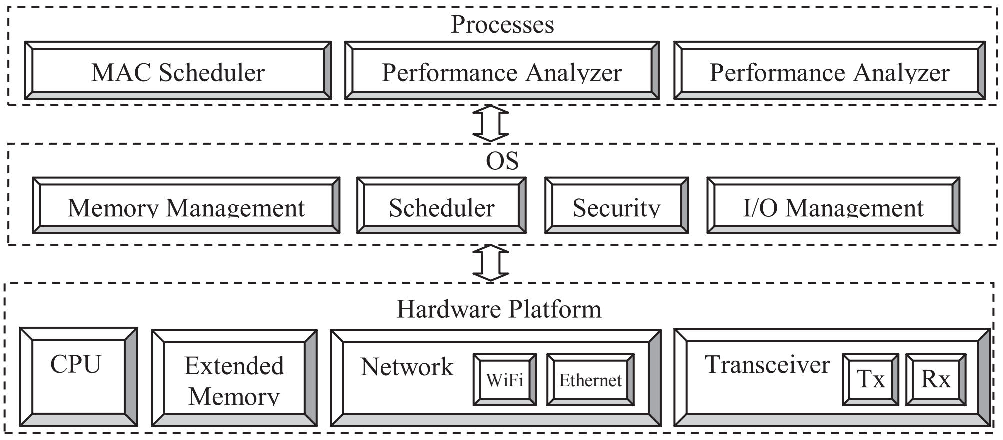

Fig. 3. Sink Node Software Architecture.

3.3. Server Software Architecture

The backend of the WSN system encompasses a set of components including a database server, web server, processing engine, and user alert mechanism. We refer to this collection of components as the Data Management Centre (DMC). The sink node communicates the data collected from the network to the DMC through WiFi, Ethernet, Cellular, or even Satellite. All data is stored in a database, and processed to generate certain information and reports. The web server allows system users a convenient tool to monitor and interact with the system. Furthermore, the system administrator can send configuration updates to the sensor network through the web interface even from his/her web-enabled mobile device. In case of emergency alerts, users can be notified via email or instant messages in realtime. A block diagram showing the DMC is shown in Fig. 4.

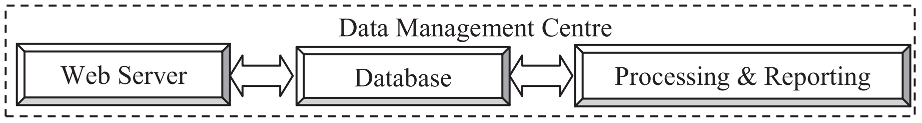

Fig. 4. Data Management Centre (DMC).

3.4. Software Module Energy Analysis Tool

In order to achieve a completely passive powered solution using energy harvesting, each software module must be analyzed over time to reduce or eliminate power hungry system calls within the module. When the software module is optimized for minimal power operation, an average energy stamp is calculated for each module. Therefore, we developed a software module energy analysis tool using Labview, seen in Fig. 5, which measures the average energy consumption of each module on the specified standard platform. Using National Instruments data acquisition card PCIe-6259, the differential microcontroller core voltage drop ΔV_{core} is measured across a current sensing resistor Rs to calculate instantaneous current consumption ΔI_{core}, such that $\Delta I_{core} = \Delta V_{core} * R_s$. Integrating the product of ΔI_{core} and the operating core voltage V_{core} over the period of time a software module, M_x, is running produces the energy stamp of the software module, such that the measurement is repeated k times to produce an average, as described in eq.1.

$$\text{Average Energy Stamp } (M_x, k) = \frac{(n * Rs)}{k * fs} \sum_{m=1}^{k} \sum_{i=1}^{n} (\Delta \text{Vcore}_i * \text{Vcore}_i)_m \qquad (1)$$

Such that the sampling frequency of the data acquisition card is $f_s=(1/\Delta t)$, and n is the total number of samples taken during the operation of the software module M_x from start to end.

The acquired power consumption data can be saved, reloaded, and statistically analyzed using two time-domain vertical markers. The user can slide the two vertical cursers to enclose the region of interest to automatically obtain a statistical representation of that region, such as averages, maximums, and minimums for power, voltage and current collectively as well as time duration, and average energy consumption. Using the energy analysis tool, we were able to detect power intensive software modules, and effectively reduce average energy stamps and active runtime by a considerable factor for each software module. The graphical representation of the software module power consumption can add an extra dimension in evaluating and considerably improving WSNs software modeling in terms of energy and execution time.

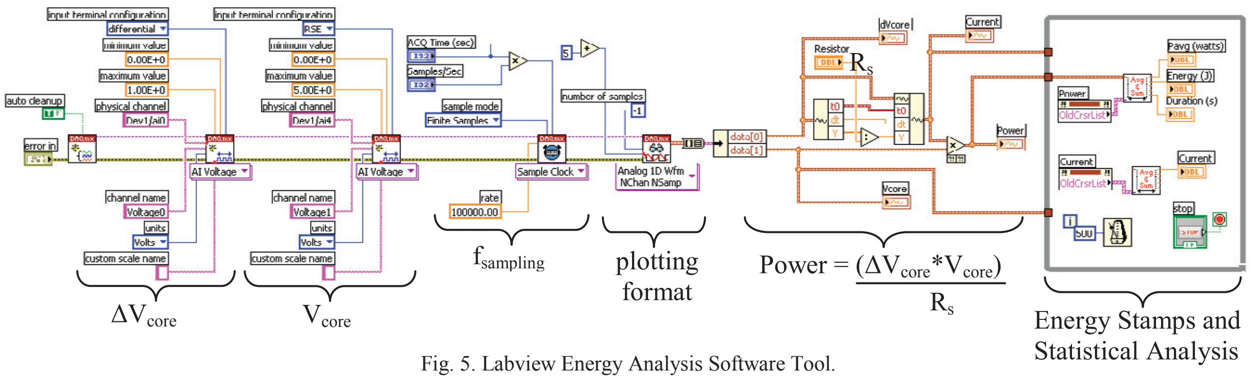

Fig. 5. Labview Energy Analysis Software Tool.

4. Experimental Implementation in the Oil Sands

The Oil Sand operator who sponsored this project is a crude oil production company located in the Athabasca Oil Sand mines of north Alberta, Canada. The company uses large vibration screens to separate large ores from the slurry mixture by means of mixing hot water with oil sand and with the help of the vibrating action. Fig. 6 illustrates some snapshots of different parts of the operation. The problem lies with the high volume and velocity of the slurry falling on the screen and the abrasive nature of sand. After around 2000 hours, the tungsten coated screen mesh wears down completely and eventually some links break off forming larger apertures in the screen mesh allowing large ores to sift through causing upset to the downstream process. Manually checking the vibrating screen cloth forces the production line to shutdown resulting in hundreds of thousands of dollars loss due to production downtime and premature screen replacements. The harsh environment surrounding the operation of the vibration screen introduces several challenging issues for WSNs such that the sensor nodes are forced to be located underneath screen to shield them from direct contact with the abrasive flow of the slurry mixture, and the constant vibration eliminates the possibility for any wired solutions extending outside the screen. Therefore, the health monitoring of the vibration screen lends itself perfectly to a low cost, wireless monitoring, energy harvesting, physically rugged, and remotely reconfigurable energy aware software architecture. Thus, the DREAMS architecture was realized as a software modeling architecture for ultra low power energy harvesting applications operating in extremely harsh environments, and implemented on a our unique hardware platform as a self powered vibration-to-energy harvesting, physically rugged, modular, volumetrically small, and low cost WSN platform to monitor the health conditions and fault detection of the vibration screen mesh.

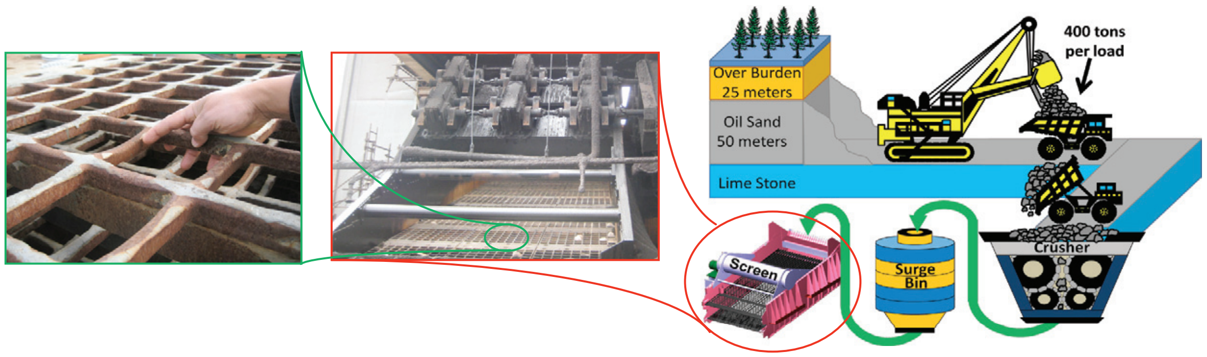

Fig. 6. (a) Etched mesh screen; (b) Vibration filter Screen; (c) Production process.

4.1. Self-Powered Source Node System Architecture

The source node system architecture was designed to be ultra low power, battery-less, low cost, rugged, and modular in both hardware and software aspects. The source node utilizes energy harvesting using a custom designed piezoelectric transducer, which is matched to the resonant frequency of the vibration screen. Most common energy harvesting methods such as piezoelectric, solar, thermoelectric, biomechanical, electrostatic, etc generate various types of power signals which are rectified and stored as energy on a local capacitor. The capacitor value plays an important role in determining the maximum storage energy available for the DREAMS architecture, such that a very large capacitor value will essentially delay the startup wait period to establish a minimum working voltage level. Conversely, a very low capacitor value will limit the number of executable software modules during run time. Consequently, the actual recommended value of the capacitor is application dependent and heavily influenced by the amount of continuous generated power of the energy harvesting unit. We recommend a minimum capacitor value of 470μF with low internal resistance for applications with low generated input power in the low μW range.

As a proof of concept, we chose the PIC24F16KA102 new XLP family with the lowest deep sleep currents of 25nA without RAM retention at 1.8V, and approximately 140nA normal sleep current with RAM retention at 3V. We also chose the enhanced Nordic nRF24L01p, currently the lowest power consumption transceiver with the highest data rate output of 2Mbps. Higher communication data rates lowers the total energy consumption per transmitted bit and reduces possible packet collisions due to the short transmission time period at the cost of higher bit error rates. Source nodes are placed at every intersection underneath the mesh, see Fig. 7, as described in **[9]**, and a sensor probe is extended in each direction from the intersection to the center of the ligament to acquire sensor readings about the health conditions of the vibration screen.

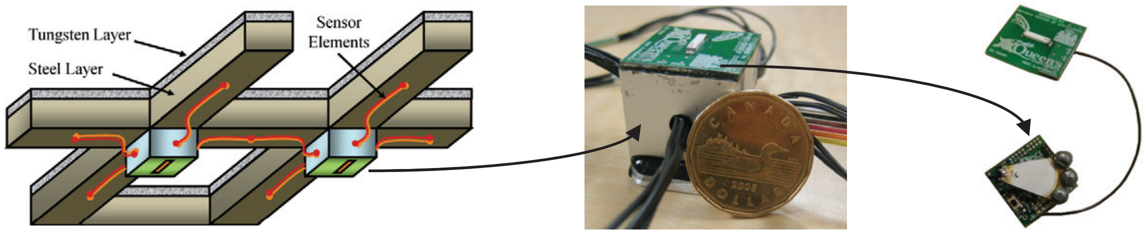

Fig. 7. (a) Sensor nodes underneath the mesh screen; (b) Packaged sensor node; (c) Unpackaged energy harvesting sensor node.

The sink node was designed as a low power, rugged, general purpose sensor platform easily adaptable to other applications in harsh environments. The sink node collects all the data from the source nodes using two transceivers, and transfers it over to a PC or laptop through a USB connection. The laptop acts as the server base station where the received packets from source nodes are analyzed, stored on a remote database, and displayed in an easy to understand GUI. In addition, the sink node is responsible for synchronizing all source nodes registration and MAC protocol. In order to reduce packet collisions, the sink node controls two transceivers which operate on two different radio channels. The first transceiver receives synchronization packets to register nodes. The second transceiver receives report packets from source nodes that are synchronized and reporting periodically.

4.3. Graphical User Interface

There are two graphical user interfaces designed to be simple and intuitive. The first interface is a local GUI, which resides in the application server. The GUI allows the user to configure system settings such as the number of sensors in the network, average charging time, periodic reporting time, USB communication speed, etc. In addition, the GUI displays the current reading of each sensor and shows how many sensor nodes currently online, active, or reporting, which is very important for the reliability of the system and for debugging purposes.

The second interface is a web-based-interface, which allows users to view the current health status of the mesh screen remotely. Using a secured login, the user can view the health of each screen with more information on the sensor level if needed. The web interface was designed to be viewable on a wide range of devices including standard size laptops or desktops, to tiny handheld mobile phones and PDAs.

5. Conclusion

We have presented DREAMS architecture, a software architecture model for ultra low power WSNs operating solely on energy harvesting. The DREAMS architecture focuses on modeling WSN applications using software modules interconnected and executed dynamically at runtime by evaluating the energy stamp requirement of each module and corresponding priority level. The advantages of the DREAMS architecture is the added level of flexibility, module reconfiguration, dynamic runtime execution of software modules, and the future ease of over-the-air low energy requirement of upgrading, replacing, or adding further software modules. We have also presented the energy analysis tool using Labview to evaluate energy consumption of software modules. The DREAMS architecture was successfully implemented on a low cost, ultra low power, rugged, energy harvesting WSN platform, which allowed oil sand operators, for the first time, the ability to remotely retrieve real-time thickness sensory reports about individual ligaments in the vibration screen mesh.

References

[1] M. Barnes, C. Conway, J. Mathews, and D.K. Arvind, “ENS: An Energy Harvesting Wireless Sensor Network Platform,” *Systems and Networks Communications (ICSNC), 2010 Fifth International Conference on*, 2010, pp. 83 -87.

[2] J. Gakkestad and L. Hanssen, “Powering Wireless Sensor Networks Nodes in Northern Europe Using Solar Cell Panel for Energy Harvesting,” *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, 2011, pp. 1 -5.

[3] C. Alippi, R. Camplani, C. Galperti, and M. Roveri, “A Robust, Adaptive, Solar-Powered WSN Framework for Aquatic Environmental Monitoring,” *Sensors Journal, IEEE*, vol. 11, Jan. 2011, pp. 45 -55.

[4] J. Sorber, A. Kostadinov, M. Garber, M. Brennan, M.D. Corner, and E.D. Berger, “Eon: a language and runtime system for perpetual systems,” *Proceedings of the 5th international conference on Embedded networked sensor systems*, New York, NY, USA: ACM, 2007, pp. 161–174.

[5] A. Lachenmann, K. Herrmann, K. Rothermel, and P.J. Marrón, “On meeting lifetime goals and providing constant application quality,” *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, Nov. 2009, pp. 36:1–36:36.

[6] A. Kansal, J. Hsu, M. Srivastava, and V. Raqhunathan, “Harvesting aware power management for sensor networks,” *Design Automation Conference, 2006 43rd ACM/IEEE*, 2006, pp. 651 -656.

[7] T.V. Prabhakar, S. Devasenapathy, H.S. Jamadagni, and R.V. Prasad, “Smart applications for energy harvested WSNs,” *Communication Systems and Networks (COMSNETS), 2010 Second International Conference on*, 2010, pp. 1 -7.

[8] “TinyOS Home Page: http://www.tinyos.net/,” Jun. 2011.

[9] A. El Kouche, Al-Awami, L., H. Hassanein, and Obaia, K., “WSN Application in the Harsh Industrial Environment of the Oil Sands,” *Accepted for presentation at the 7th International Wireless Communications and Mobile Computing Conference*, Istambul, Turkey: 2011.