The need for Run-time Reconfigurations and Software Update Mechanisms in Wireless Sensor Networks

Rekha.K.S., Dr.T.H.Sreenivas

Abstract— Wireless Sensor Network is a collection of sensor nodes which are distributed for monitoring the dynamic applications. Earlier the sensor nodes were intended only for specific application. In dynamic application the existing nodes may fail due to many reasons and few new nodes may join the network, leading to the topological changes. Due to the factors like heterogeneous nodes, task reassignment and topology changes the WSN has to adapt reconfigurations at runtime. The Wireless Sensor network reprogramming is a major challenge due to energy consumption, memory and processing power. It is highly impossible for the network administrator to manage the nodes, install new software and hardware each time as the nodes are deployed in an inaccessible region. There is a need to update sensor node software or node configuration. We require a set of code dissemination protocols and an Operating System for the reconfigurations of remote sensor node. In this paper, an attempt is made to present the dynamic adaptation of the Wireless Sensor Networks. The capabilities of Run-time Reconfigurations at Node level Application, Sensor Application and Network level Application are explained. The software code update mechanisms can be carried out with the help of the different programming models. The changing version of software can be managed with the Configuration Management.

Keywords— Adaptation, Context, Configuration Management, Reconfigurations, Topology changes, Task Assignments, WSN.

I. INTRODUCTION

The Sensor nodes are densely deployed in the area of phenomenon to be monitored in the Battle Field, Precision Agriculture, Habitat and Geophysical Monitoring etc. The Sensor nodes can be deployed either manually or dropped from a plane. The nodes identifies the nearby neighbours and establish communication. It is required to collect the context information for the specified time interval. The sensor nodes operate with limited battery power. The deployed nodes may fail due to limited battery and many new nodes may join the network. The change in the location of sensor nodes may lead to topology changes. One of the reconfiguration method is to monitor the WSN’s Quality of Service at the base station and reprogram the sensor nodes. An alternative solution is to locally monitor the environment conditions and perform the reconfiguration on individual sensor nodes.

II. CLASSIFICATION OF WIRELESS SENSOR NETWORK APPLICATIONS

In paper[1], the Authors Pankaj Nagar, Kripa Shanker Chaturvedi have clearly classified the Wireless Sensor Network applications into sensor applications, node applications and network applications. A sensor application may collect the data from the environment and stores the data. Figure 1 shows the architecture of Sensor Networks.
Fig.1. Sensor Network Structure

The Sensor Application can interact with hardware and operating system. The node application is responsible for middleware related functionality and self localization. The sensor network application describes the main tasks and required services of the entire network and acts as an interface to the administrator to evaluate the network results. The Nodes can be contacted through services of the middleware layers. They do not perform any individual tasks. The node middleware handles network formation and maintenance. The Distributed Middleware coordinates the cooperation of services within the network. It is logically located in the network layer but physically exists in the nodes. All layers together in conjunction with their configuration compose the sensor network application.

A. Node Application

The Performance of the Wireless sensor network depends on the node deployment in an area of monitoring. If the area of monitoring is known and nodes are placed manually then data flows in a fixed path. If the Sensor nodes are deployed randomly then cluster should be formed to allow the efficient data transfer. The route will be ad-hoc and data has to pass through of multiple wireless hops.

In paper[2], the Authors, X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill explain about the connectivity and its relationships. The Connectivity refers to the ability of active nodes to stay connected. The relationship between connectivity and coverage is quantified. The authors define k- coverage as any location in the network being monitored by at least k nodes. The network is said to be k-connected if the network remains connected even if any k−1 nodes fail. Two protocols to provide coverage and connectivity guarantees are designed. Coverage Configuration Protocol (CPP) allows the dynamic configuration of the network for various degrees of coverage, assuming that the unnecessary nodes can power down to save energy.

The Authors Sergio Khalife, Kiran Dasari, Joseph St. Pierre, Jeffrey DeArruda [3], have proposed a way to calculate the distance between the nodes and gateway node. The node rotates around the gateway on a fixed radius. The X and Y position values of each node for a given time interval are considered to calculate the distance between nodes and the gateway node. Equations 1 and 2 show how to calculate the next X and Y positions of a given node.

Equation 1: \( Y = R \sin(\text{AngSpeed} \times t \ + \ \phi) \)
Equation 2: \( X = R \cos(\text{AngSpeed} \times t \ + \ \phi) \)

Where: \( R \) = the radius relative to the gateway
\( \text{AngSpeed} \) = angular speed of the node
\( \phi \) = angle of initial position of node

There should be an application at each node, which would add new functionality or replace an existing functionality without changing the sensor behaviour. This is the major reconfiguration required at the node level. The flexible architecture of the node software to support the reconfiguration are as shown below in Figure.2.

![Fig.2. Structure of node Application](image)

The Structure of node is made flexible with the functionality shown in terms of different blocks to support the scalability feature. The Operating System layer is being divided into Node Specific Operating System and a Driver Layer. The Driver layer consists of Hardware Driver and Sensor Driver. The Node specific Operating System manages the device specific operations such as memory management and process management. Host Middleware is to organize the distributed nodes in the network. The functionality is divided in terms of modules. The Algorithms are designed according to the nature of modules. The Virtual Machines enable the platform independent execution of programs.

At run-time, nodes locally observe the design-time defined events. An event either triggers the change of a local mode or is communicated to a centralized location, for example as part of a distributed detection of a global event. A centralized location, such as a sink, initiates the change of global mode by communicating to all nodes in the network. Due to the design-time definition of the modes, a simple, low overhead flooding approach suffices. The reconfiguration approach is made robust for practical deployments by a recovery approach that allows nodes to determine the currently used global mode after, re-joining the network or missing a request to change a global parameter[4].

B. Network level Application

At network-level reconfiguration, the entire network design and performance has to be analysed based on the factors like power consumption and reliability of network. The factors...
affecting the network performance may be due to change in user requirements, new parameters received from the application. The reconfiguration techniques should support the dynamically changing behaviours and inject the suitable configurable parameters into the application network.

Many sensor-network applications require some form of self configuration, where sensor nodes take on specific functions in the network. The pre-deployment configuration is often infeasible because some configuration parameters such as node location and network neighbourhood are typically unknown prior to deployment. Also, node parameters may change over time, necessitating dynamic re-configuration. When sensor nodes join the network, they are in an initial, homogeneous software state. However, nodes may differ in their hardware capabilities and parameters such as their location or their network neighbourhood. The goal of configuration is to break the initial symmetry and assign specific roles to individual sensor nodes based on their properties[5]. The parameters and node properties keeps changing according to the context and immediately role assignments should be updated to reflect the changes.

C. Sensor Application

Configuration data is stored on non-volatile storage space i.e., EEPROM of the mote. The sensor node is supposed to read it and take appropriate action. The process of accessing and modifying the configuration data of a sensor residing on EEPROM of the mote, without touching the mote software which resides on the program memory of the mote is known as Sensor Reconfiguration.

The Authors Koustubh Kulkarni, Sudip Sanyal in [6] have explained the Sensor Reconfiguration process by intruding into the network and propagating the new configuration parameters. This reconfigurations of sensor network is dynamic because the network is already in place and running. It is doing its desired job and we are changing its configuration parameters[6]. If there are there be multiple code images stored on the EEPROM of the mote, then it is the responsibility of the network administrator to select the code configurations and inject into the network.

III. DISSEMINATION PROTOCOLS

To obtain the estimated performance of a network, the new code update should be sent to the maximum nodes in a network. The algorithm used to send the code updates to remote node should be reliable and dynamic. The new code would be sent to all the nodes once the context is analysed. The system should receive the new code update, compare with its old version and load it into the memory.

The protocols developed for disseminating a software update in a wireless sensor network are based on data dissemination protocols, with key examples being Directed Diffusion [9], Deluge [8]. The key differences between protocols for data collection and for updates are:

(a) the data flow for software updating is from the gateway to the nodes
(b) typically, intermediate nodes both store and forward the data,
(c) the data transfer must be reliable and achieve network-wide coverage[7].

Jonathan W. Hui, David Culler in[8] explains the Data dissemination protocol Deluge. The Deluge is an epidemic protocol and operates as a state machine where each node follows a set of strictly local rules to achieve a desired global behaviour: the quick, reliable dissemination of large data objects to many nodes. In its most basic form, each node occasionally advertises the most recent version of the data object it has available to whatever nodes that can hear its local broadcast. If S receives an advertisement from an older node, R, S responds with its object profile. From the object profile, R determines which portions of the data need updating and requests them from any neighbour that advertises the availability of the needed data, including S. Nodes receiving requests then broadcast any requested data. Nodes then advertise newly received.

Chalermek Intanagonwiwat, Ramesh Govindan, Deborah Estrin, John Heidemann, and Fabio Silva[9] explains Directed diffusion as code dissemination protocol consisting of several elements: interests, data messages, gradients, and reinforcements. An interest message is a query or an interrogation which specifies what a user wants. Each interest contains a description of a sensing task that is supported by a sensor network for acquiring data. Typically, data in sensor networks is the collected or processed information of a physical phenomenon. Such data can be an event which is a short description of the sensed phenomenon. In directed diffusion, data is named using attribute-value pairs. A sensing task (or a subtask thereof) is disseminated throughout the sensor network as an interest for named data. This dissemination sets up gradients within the network designed to “draw” events (i.e., data matching the interest). Specifically, a gradient is direction state created in each node that receives an interest. The gradient direction is set toward the neighbouring node from which the interest is received. Events start flowing towards the originators of interests along multiple gradient paths. The sensor network reinforces one, or a small number of these paths[9].

IV. THE EXECUTION ENVIRONMENT

The execution environment is where the software update is disseminated and executed. Whenever there is a change in the context, the corresponding Software update functionality
need to be transferred to the appropriate nodes. The Figure.3 shows the Execution Environment. The code has to be injected by overwriting the program memory, implemented in the operating system(e.g.SOS[10]), in a middleware layer(e.g.Agilla[11]), or implemented as an application(e.g.MOS[12]).

(a) Application  (b) Middleware  (c) O/S

Fig 3: Execution Environment

V. SOFTWARE UPDATES IN WSNs

WSN’s software update research is described under three categories: the sensor node as execution environment, the protocols for disseminating the update, and size reduction of transmitted update at the Host[13]. The flow of software update in the components of WSN are as shown in Figure.4.

Fig 4: Software Update in components of WSN

VI. PROGRAMMING MODELS IN WIRELESS SENSOR NETWORKS

Reconfigurations can be performed either by Centralized Reconfiguration method or Distributed Reconfiguration method. The Centralized Reconfiguration monitors the Quality of Service at the base station and reprogram the sensor nodes when necessary. The Distributed Reconfiguration method locally monitor the context and perform reconfiguration on each node [14]. The Figure 5. Shows the few major types of Reprogramming models.

WSN Programming Models

Fig 5: WSN Programming Models

A. Full Software Image Upgrades

The most popular way to update software in sensor networks is to compile a complete new binary image of the application software together with the system code and overwrite the existing system image of the sensor node. Since the image is compiled and linked afresh in every iteration, these solutions offer a very fine-grained control over the possible reconfigurations. However, these approaches result in bandwidth overhead as unchanged parts of an application need to be re-disseminated in the network[15]. TinyOS [15] is a popular operating system for sensor nodes that generates a monolithic binary image of the entire application. Deluge [8] is a networked boot loader and dissemination protocol that performs full image upgrades of TinyOS applications. The high update cost is primarily due to the large size of the monolithic binary image (30-40 KB) that needs to be transferred reliably to the entire network. Installing the full image also disrupts the ongoing applications on the nodes, resulting in a loss of work and resources that has already been spent in processing the previous data.

B. Modular Upgrades

Systems that support modular upgrades consist of a run-time loader and linker. The loader is responsible for tracking the storage of the binary modules in the code memory and allocating appropriate resources for them to execute. The linker is in charge of resolving any references made by the modules to the kernel, common libraries, or other modules in the system. SOS [10] is the operating system allowing modular binary upgrades at run-time.

- SOS

The architecture of SOS consists of a thin kernel that is statically installed on all the nodes in the network. The rest of the system and application components are implemented as modules. Modules are binary software components that can be dynamically installed on a node at runtime. The SOS kernel provides support for loading and unloading modules at runtime, besides a rich set of services such as dynamic memory allocation, software timers, sensor manager and high-level I/O interface.
C. Virtual Machines

Virtual machines provide a run-time environment that isolates the execution of applications from the underlying platform. The execution engine proposes a set of high level instruction set, enabling a compact representation of the application code. Therefore, size of applications is reduced in this way and software updates can be distributed more easily. However, since execution takes place within a virtual machine, execution cost is higher compared to native code.

• Mat’e

Mat’e provides a method for creating virtual machines for sensor nodes [17]. It runs on the top of TinyOS operating system, and works as a byte code interpreter. Mate has high level user interface, and it allow using large program by breaking the code into multiple small pieces called Capsules. Each capsule contains 24 instructions, and length of each instruction is a byte. The program is injected into sensor network faster and easily by using Mate.

D. Component-based Reconfiguration

Componentization offers the capability of black-box integration of system modules in order to simplify modification and reconfiguration of dynamic systems. This abstraction simplifies reconfiguration realization by formulating the way through which software modules can interact. In particular, software components interact with their environment (i.e., other components) exclusively through interfaces and receptacles[18].

• FIGARO

In FIGARO, a component represents a single unit of functionality and deployment. The services provided by a component are described by its interface. For instance, Components must provide the code for all the operations in the interface declaration. The programming model of Figar has two core constituents:

- the component model defines constructs for structuring the code on the single nodes. It is designed with reconfiguration in mind, thus providing dedicated constructs to deal with component dependencies and versions, and to simplify the reconfiguration process.

- the distribution model defines constructs to restrict component dissemination only to a given subset of nodes—the reconfiguration target—based on programmer specified characteristics of the nodes or their current software configuration.

In FIGARO, programmers do not need to manage the reconfiguration manually. Instead, the underlying run-time automatically and transparently manages the reconfiguration process, based on dependencies and component versions. When components are instantiated at startup, the run-time keeps track of their version, the interface they implement, and their dependencies. In FIGARO, dependencies are explicitly declared by the programmer using the DECLARE DEPENDENCY macro. The first parameter of this macro is a receptacle, the dual of an interface. An interface specifies a set of operations provided by a component to others, while a receptacle specifies the set of interfaces a component requires from others. Using FIGARO, the programmer can deal explicitly with component dependencies and version constraints, as well as select precisely the subset of nodes targeted by reconfiguration, leaving the others unaltered. The run-time support imposes a very limited processing and memory overhead. The communication overhead lies within 9% of the theoretical optimum[16].

• Remora

REMORA, a lightweight component model designed for resource constraint embedded systems, including WSNs. The strong abstraction promoted by this model allows a wide range of embedded systems to exploit it at different software levels from Operating System (OS) to application. To achieve this goal, REMORA provides a very efficient mechanism for event management, as embedded applications are inherently event-driven. REMORA components are described in XML as an extension of the Service Component Architecture (SCA) model in order to make WSN applications compliant with the state-of-the-art componentization standards. Additionally, the C-like language for component implementation in REMORA attracts both embedded system programmers and PC-based developers to programming for WSNs. Finally, REMORA features a coherent mechanism for component instantiation and property-based component configuration in order to facilitate lightweight event-driven programming in WSNs[18].

E. Reconfiguration Middleware

In this approach, dedicated middleware platforms are designed for networked sensor systems, with abstractions that can offer consistent and general mechanisms to configure, deploy, and dynamically reconfigure both system and application level software.

• μDDS:

μDDS is a publish/subscribe middleware for real-time wireless embedded systems based on Data Distribution Service specification and implements a subset of standard interfaces for event subscriptions and publication to be used by applications. Applications implemented on top of μDDS can disseminate and collect data through a publish/subscribe interface provided by the middleware. Different routing protocols can be used to implement the overlay network; the
middleware is currently implemented on 802.15.4 standard devices which can support the star, tree and mesh topologies [19].

- **MoMi**

MoMi uses a rule-based system to detect faulty nodes in a wireless sensor network. MoMi uses a Model-Based Diagnosis (MBD) framework to present the likely causes of system abnormalities to an administrator. Observations about local and surrounding nodes are compared with each other based on predefined rules, after which conflicting observations are sent to a gateway. The gateway in turn generates a prediction of possibly faulty nodes[20].

 VII. CONFIGURATION MANAGEMENT

An Administrator will send a request to the gateway to change the parameters, to be sent to the nodes. If there is a change in the software, then a new configuration of software update need to be sent through the Dissemination Protocol to the intended sensor nodes. The sensor nodes checks the existing configuration version number and updates itself, if the new updates is greater than the existing configuration number. Version number is compulsory to load the current software configurations into the network. Nodes update themselves if they know the maximum version number.

Namespace management is required to provide full control in a heterogeneous environment: for different hardware platforms, or for different required functionality on identical hardware platforms. Probably a more flexible scheme than just individual, group-based, and network-wide updates should be supported[7].

 VIII. CONCLUSIONS AND FUTURE WORKS

The Wireless Sensor Networks may be deployed in dynamic applications. This paper is an attempt to provide an extensive survey of reconfigurations and software updating in Wireless Sensor Networks. The different level of Run-time Reconfigurations occurring at the Sensor level, Node level and in Network level are identified. In order to design an efficient application, various Programming Models such as Full Software Image upgrade, Modular Image Upgrade, Virtual Machines, Component Based and Middleware based model can be used. There is a trade-off between the efficiency of the execution environments and update overhead. Depending on the type of application, any programming models and Execution platforms may be chosen.

Regarding future work, we would like to explore an optimized algorithm to minimize the memory utilization and energy consumption for the distribution of code updates An adaptive framework has to be designed for Run-time Reconfigurations and Code update mechanism.

REFERENCES


[4] J Marcel Steine1, Cuong Viet Ngo2, Ramon Serna Oliver2, Marc Geilien1, Twan Basten1,4, Gerhard Fohler2, Jean-Dominique Decotignie3, “Proactive Reconfiguration of Wireless Sensor Networks”


5th European Conference on Wireless Sensor Networks (EWSN),
Bologna, Italy, LNCS vol. 4913, 286-304


BIOGRAPHIES

Rekha.K.S. holds Master’s degree (2007) in Software Engineering, from Visvesvaraya Technological University, Belgaum. Currently she is working as an Assistant Professor in the Department of CS&E, The National Institute of Engineering, Mysore. She is pursuing her Ph.D. Her research interests include Wireless Sensor Networks, Middleware design, Software Engineering, Software Architecture, Embedded Systems and Distributed Systems.

Dr. T.H. Sreenivas holds a Master’s Degree(1986) from IIT, Khanpur and Ph.D (1999) from IIT, Madras. Currently he is working as Professor in the Department of Computer Science & Engineering, The National Institute of Engineering, India. His research interest includes Operating systems, kernel development and programming, real-time kernels, Real-time operating systems and Wireless Sensor Networks.