The Design and Implementation of A General WSN Gateway for Data Collection

Newlyn Erratt and Yao Liang
Department of Computer and Information Science
Indiana University Purdue University Indianapolis
Indiana, USA
nerratt@umail.iu.edu, yliang@cs.iupui.edu

Abstract—**In this study, we present our novel development of a general gateway system for data collection in wireless sensor networks (WSNs). The unique characteristic of our developed WSN gateway is that it has been designed easily configurable for nearly any WSN data collection applications, with diverse data managements and WSN protocols. In particular, it supports WSN transport layer at the gateway. Our implementation leverages Java and widely adopted TinyOS platform for various WSNs. We have conducted thorough tests for the validation of our developed gateway system, including the use of three real-world WSN field test-beds. The evaluation and validation results demonstrate the effectiveness and robustness of our gateway system.**

Keywords—wireless sensor networks, gateway architecture; gateway system; data collection; user configurable design and implementation

I. Introduction

Wireless Sensor Networks (WSNs) are networks of wireless embedded systems with attached sensors. WSNs are not only an active area of research, but also are being widely applied to the areas of environmental research and monitoring, commercial/manufacturing processes, healthcare, military, and others. While this research is important, not much work has gone into making deployment based research (e.g. environmental research) easier and more cost-effective for researchers without a lot of technical knowledge.

In most monitoring WSNs development and deployment, one of the most vital goals is to collect and aggregate the sensor data (e.g., environmental data, patient data, battlefield data) for analysis. To this end, WSN gateway development is critical for data gathering in diverse WSNs in real-world tasks.

A WSN gateway connects a WSN to the Internet, as illustrated in Fig.1, where remote users can easily retrieve WSN data through the Internet in a real-time manner, and analyze their collected data either online or offline. As we can see, there are four primary components in Fig. 1. The first component is WSN motes and their application code for data sampling and communication. The second component is the WSN sink and its code to collect all mote data and pass them on to the connected gateway machine. The third component is the gateway system which is in charge of receiving data from the sink, doing any preliminary processing necessary, storing the data to a local database, and forwarding the data to the data management system or application via the Internet. It should be noted here that complex data analysis is not expected to be done in the gateway; only simplistic and necessary work should be done such as transforming the data into an easier to use format. The fourth component is the data management system on the back-end server which receives the data from the gateway, provides functionalities for monitoring/viewing live data and other more complex data analysis, and backs the data up to a database. In this framework, the data management system's database is the primary backup while the gateway's database is a secondary backup in case the management system is down or unreachable, which provides the WSN with two levels of backup for reliable data connection.

However, while many advances in WSNs have been made, there is still a lack of general software platforms for WSN gateways that can be easily deployed in various WSN applications, with a variety of WSN protocols and management systems. Besides, a desirable WSN gateway platform should be able to support WSN transport layer. Due to the resource limitations on WSN motes, WSN transport layer protocols can be quite asymmetric in terms of processing, where the protocol processing at individual motes can be easily performed (e.g., some data compression mechanisms) but the final processing may overwhelm if performed at sink mote. This motivates our work. In this paper, we present our design and implementation of a general WSN gateway software system in a novel way for various WSNs. The unique characteristic of our work is that our developed WSN gateway is aimed to be easily configurable for nearly any WSN data collection applications, with diverse data managements and WSN protocols. In particular, it supports WSN transport layer at the gateway which is assumed not to be power limited. In this way, the developed gateway should directly facilitate heterogeneous WSNs management [1] and support various WSN routing protocols and transport-layer data collection protocols such as compressed data-stream protocol CDP [2].

The rest of the paper is organized as follows. Section II briefly discusses previous works. Section III presents our design criteria for the gateway. Section IV describes important design features. Section V discusses some specific implementation decisions. Section VI gives our initial testing of the gateway system. Section VII presents our conclusions and outlines future work.

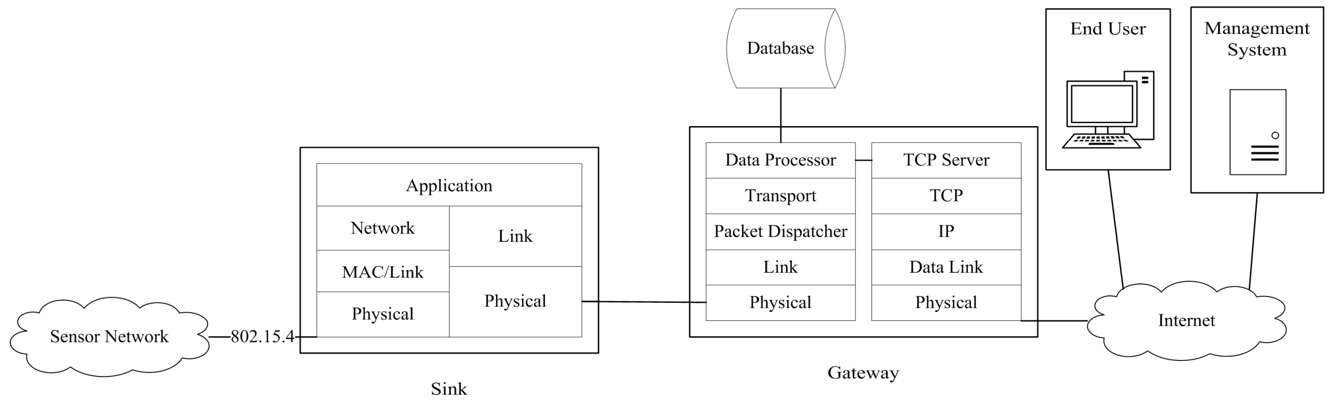

Figure 1. A network view of gateway.

II. RELATED WORK

Several studies exist about WSN gateway systems, ranging from ordinary aspects of WSN gateways (e.g., [3, 4, 5, 6, 7]) to specific concerns such as security [8]. In [4] static packets are required since all customization is done through XML. Reference [7] presents a system using the Stargate hardware and takes into account that the gateway may be resource limited whereas our assumption is that the gateway machine has access to wall power. While many of the systems reported in the previous work are research projects and not yet available for broader use, Xserve [9] is an industrial gateway available to use, but it is proprietary and therefore does not support the level of customization that users may require. Besides, Xserve is also completely tied to the proprietary WSN management system and the Xmesh network protocol, which significantly limits its potential applications to many real-world tasks. In contrast, our work aims to develop a general user-configurable WSN gateway system to work with any WSN routing protocols and management systems in principle. While the work in [6, 7] shares some goals with ours, our work was independent from theirs, and addresses some challenging issues not addressed in the previous work.

III. CRITERIA OF THE GATEWAY

We begin with defining some important design criteria that we follow in this work.

- The focus is a general user-configurable WSN gateway architecture and system for effective and efficient data collection.
- The gateway must easily support both static and dynamic packet formats. Much of the current research on gateway design for WSNs relies on the fact that packet formats are static by using methods such as XML [4] to define packets. However, in many situations, variable packet size and dynamic packet formats are necessary. For example, the packet in CDP [2] is dynamic.

  The gateway should be separate from WSN management system, but at the same time, must provide a simple interface for the management system. The separation of gateways from WSN management system is to facilitate network management for heterogeneous WSN systems [1], where different gateways could be adopted for different WSNs, with a unified management system.
- The gateway must reliably log all received packets to a database. To eliminate possible packet loss, it is desirable to conduct logging at the gateway instead of leaving that solely to the management system.

IV. GATEWAY DESIGN

In addition to our criteria, we describe some specific design goals first in section IV.A that we attempt to achieve. Section IV.B then details the actual design of the system in a top-down manner.

A. Gateway Design Goals

In designing our gateway we have four primary goals.

- The gateway should not make any assumptions regarding the protocol stack running on the motes. The reconfiguration of the sensor network to use a new network stack should not require a major change in the gateway. This enables the gateway to easily integrate with different WSNs that implement different network stacks. Any changes that must be made should be handled in one specific and well-defined configuration mechanism.
- The gateway should be easily configurable for different applications. It should provide all of the base functionalities including receiving packet data from the network for decoding and processing. Additionally, there should be a system for writing data to the database as well as forwarding it on to the network. These functionalities should not require any changes by the user in the general case. The user should provide code to support the packet structure, data processing, and converting the processed data into formats suitable for writing to the database and the network. To achieve this, we need to separate the code that must be customized, based on the application, from the core of the gateway.

- The gateway functionality should require little to no specific control code on the motes and base station. When any control code is required on the motes it is desirable to keep it very modular in the form of small libraries with well-defined interfaces that can be easily used by any applications. Consequently, any application related functionality will be clearly separated from the control code of motes in our modules.
- New WSN applications should require little to no modification of the core gateway. The gateway will provide well-defined generic interfaces to support any new application to be added. These well-defined gateway interfaces should be well documented for the deployment of new applications.

B. *Top-Down Design of Gateway*

We take a top-down design approach. This top-down methodology facilitates in illustrating how our criteria and goals are considered and how they have shaped our design decisions. The discussion of our design is primarily at the architecture level of the gateway. We then present the gateway system design at module level, illustrating interactions between user modules and core modules. Finally, we describe each of the individual modules and interfaces in depth.

1) *Gateway Architecture*

We focus on the software architecture of our WSN gateway system. Basically, our WSN gateway has dual network stacks. As shown in Fig. 1, one network stack (referred to as WSN stack) connects to WSN sink, while the other network stack (i.e., a standard Internet network stack) connects to the Internet. As WSN network layer is terminated at the sink, the sink usually connects to the gateway directly through data link layer. The gateway aims to support a wide range of physical layer and data link layer of the WSN stack via gateway configurations. One unique feature of our gateway architecture is that we have considered a generic transport layer residing in the gateway's WSN stack, which can effectively and efficiently support WSN transport layer protocols that may be deployed in WSN motes, such as CDP [2]. Incoming packet streams flow through the gateway dual network stacks as follows. A packet is first received at the physical layer and the data link layer of the Gateway machine. Then, Packet Dispatcher switches the packet into the relevant transport protocol. The transport layer performs any necessary processing and passes the results to the Data Processor system. The Data Processor will execute any processing that must be done, convert the packet into appropriate formats, and then write it to buffers for the database as well as the TCP server which is a generic component in our gateway system.

Upon receiving data in its buffer, if a connection is available, the TCP Server will send the data out to the back-end server over TCP, through its Internet network stack. Similarly, the database buffer will be written to a database if the database is currently available.

2) *User and Core gateway spaces*

We will now discuss the Gateway from the perspective of separation of user space from the core space. Fig. 2 shows the Gateway Core and the Customization Core and their relationship. The explicit separation of these two spaces makes it easier to determine how to design the interface between the core gateway systems and the systems that users may want to customize. Choosing an appropriate boundary for the separation of these spaces will dramatically improve the ease with which a user can customize the gateway for their application.

In the Gateway Core, four primary functions take place. First, the transport layer does processing based on the transport layer used in the WSN stack. Second, packet dispatching occurs. Upon receiving a packet, the dispatcher must determine where to send the packet. To properly achieve this, the gateway provides an abstract type that defines the required functionality of a Data Processor. This type is detailed in the next paragraph. Third, the gateway core contains a Database Handler. Listening to a thread-safe buffer, whenever incoming data become available, the Database Handler connects to the database, writes the information received from the Data Processor, and logs any errors that might occur. At the same time, the TCP Server will maintain an open server socket and listen to a thread-safe buffer. As data become available on the buffer, the server will check to see if it has a current connection. If there is a connection it reads the entry from the buffer and sends it to connected destination port. It will then log any errors that might occur. It is only required that the gateway supports a single TCP connection, since it is expected that a Management System will be connected for multiple and concurrent users, with which sophisticated data retrievals and management services will be provided for end users. The use of thread-safe buffers is important to the gateway design since processing of data in the gateway requires parallelization of the system in case that data are received faster than they can be processed.

In the Customization Core three processing functions are handled. First, the Customization Core performs any application layer processing that is needed (e.g. converting raw sensor data). Second, it converts the data into a query and buffers it for the Database Handler. Third, it reformats the data into a format suitable for sending over TCP and buffers it on the TCP buffer. This Customization Core system is built on top of three abstract data types provided by the Gateway Core:

- Data Processor: Functionality defined includes initialization of the object by the Packet Dispatcher using the packet and starting the processing task.
- Query: Functionalities defined include creating the query, retrieving the database schema information (used during gateway startup), and retrieving the query.
- TCP Packet: Functionalities defined include the creation of the packet object and retrieval of the packet in a format suitable for transmission.

3) *User and Core modules*

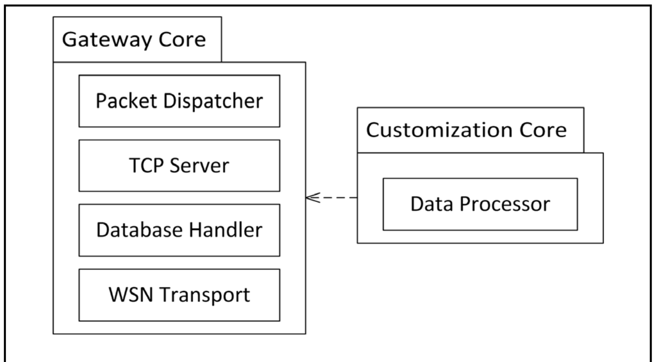

Figure 2. Core view of the gateway.

This section presents a more in-depth view of the modules discussed above to detail both the elements of the Customization Core and the Gateway Core. Our discussion of the Customization Core describes how the module can be used to support all of the gateway criteria, while the discussion of the Gateway Core details how the primary functionality of the gateway works and how each module should operate.

In the Customization Core the only defined module is the Data Processor. A user's implemented version of this module is responsible for receiving the result of transport layer processing, performing processing of the data in that packet, and formatting it for both the database and the TCP Server. That is, this module is defined as an abstract type for user's implementation. This approach supports our design criteria as elaborated as follows.

To meet the gateway criteria of being able to support both static and dynamic packet formats this approach allows a deployment of the gateway to have multiple implementations of the Data Processor with each implementation representing one specific type of packet. The user may use his/her implementation to process packets in whatever way is most useful and necessary.

To meet the criteria of providing a simple interface for connections with a management system and database, this approach enables the gateway to connect to any management system that the user wishes to use, with a well-defined interface.

The Data Processor type helps achieve the goals of both providing an easily configurable system and making no assumptions regarding packet types. To prepare for a deployment, the user implements an object that conforms to the Data Processor's defined interface. This object receives, as input, the appropriate packet object from the transport layer and is able to process that packet in any manner deemed appropriate by the user. The user need not worry about filtering packets because the Gateway Core only sends packets associated with that Data Processor. By providing both a Query abstraction, which provides a method for setting and retrieving a SQL query, and a TCP Packet abstraction, which provides a method for setting and retrieving a TCP packet, the Data Processor enables the user to write data to the database and TCP sockets using any format they require. The Gateway Core removes the necessity of writing the low-level TCP and database code and allows the user to simply design in terms of SQL statements and packets.

The Gateway Core consists of four different modules. The WSN Transport Layer module receives packets and offsets transport layer processing which was not performed at the sink. The Packet Dispatcher module receives packets from the Transport Layer module and forwards them to the appropriate Data Processor. The Database Handler module writes data into the database. Similarly, the TCP Server module is responsible for transferring data over a TCP socket through a standard Internet stack.

The Packet Dispatcher is registered as a listener for all packets in which the Gateway is interested, and is the main control thread of the Gateway. As a packet is received its type is determined. If the packet has a corresponding Data Processor, the dispatcher will create an instance of the Data Processor and pass that packet, the database buffer, and the TCP buffer to it, and then post this Data Processor task for execution.

The Database Handler is a thread that will monitor the database buffer and write the data into a database. It waits for new data on its buffer. As data is received, it connects to the database and writes the query to the database. On the other hand, once a gateway query has been executed, any possible errors would be logged. If there are more gateway queries available it will continue writing them to the database. If there are no more queries to be written it may disconnect and sleep until a new packet is added to its buffer.

The TCP server is a thread very similar in operation to the Database Handler. Started by the main thread, it establishes a TCP server socket and waits for data to arrive in its buffer. When a packet is available for sending, one of two cases will occur: (1) if no connection has been established to the server socket the packet will be discarded; (2) if a connection either from the user or management system has been established to the server socket, the packet will be written out to the connected client.

V. Implementation

For our implementation of the gateway, we decided to use TinyOS [10] as our WSN platform for the initial implementation. TinyOS is widely used and familiar due to our previous work on CDP [2]. TinyOS libraries are primarily offered in C++ and Java. We chose to use Java for our implementation because it is used extensively in our group's WSN network and data management project [1]. Additionally, we are using PostgreSQL for the gateway database. To meet design goal 1 we use the Message Listener interface to implement our Packet Dispatcher. This allows the user to leverage the Message Interface Generator (MIG) provided by TinyOS to obtain a Java representation of the packets being received. To avoid requiring a very specific system on the motes and sink we have used a version of the Base Station application provided in TinyOS 2.1.1 that was modified to use the Collection Tree Protocol (CTP) [11] for upstream communication.

During our implementation of the gateway, some specific aspects have to be addressed in an attempt to effectively achieve all of the design goals that we set. For example, we consider how to handle the asynchronous nature of our gateway design. We also have to determine the best way to handle a user's interaction with the system.

The Java Runnable interface and is used to achieve the asynchronous nature of our gateway. The TCP Server, Database Handler, and Data Processors are all implemented using the Runnable interface. Data Processors execute in a Thread Pool to prevent the creation of too many threads if data are incoming too quickly to process. The Thread Pool limits the number of running threads which eliminates the overhead incurred when too many threads exist. Also, each packet is processed independently at the gateway. To forward information from the Data Processors to the Database Handler and TCP Server we use BlockingQueue.

During our gateway implementation, we made the decision to make a small change to how data flows through the gateway. This is an implementation specific decision that is made to utilize the MIG generated packets more effectively. To make swapping of network stacks easier, we simply merge the WSN transport layer processing into the Data Processor. The Data Processor may be implemented by separate objects for transport layer processing and higher level processing so that a transport layer processor may be provided by the protocol developer and utilized by end users. The flow of the gateway can be seen in the flowchart of the main loop in Fig. 3. When the gateway starts the TCP Server and Database Handler threads are started first. Then the gateway starts its main loop. This loop consists of checking for a received packet and passing that packet on to the Data Processor. The Data Processor performs any transport layer processing, sensor data processing, query formatting, and TCP packet formatting and then exits.

Property files are used to provide a configuration interface to users of the gateway. We have chosen this configuration file based interface because the gateway is designed as server software and does not handle any data visualization. This system can also allow Customization Core preferences to be stored in the same central properties file.

VI. TESTING

We have conducted two separate tests of our gateway implementation. Firstly, we have performed a load test. Secondly, we have tested the gateway in several small real-world test-beds.

A. *Load tests*

For our load test we aim to test the maximum throughput that our gateway can achieve and maintain. To this end, we designed a special version of the sink application, in which the sink sends packets to the gateway computer repeatedly at the maximum sustainable rate. The testing hardware consists of the sink, a single MICAz [12] mote connected to a MIB520-USB [13], and the gateway computer with a single-core Intel Atom processor (1.6Ghz) and 2 gigabytes of RAM, running a desktop distribution of linux. The MIB520-USB's USB interface is limited to 56.7 kBd due to USB being implemented by using a FTDI FT2232D [14] chip to handle conversion from the MICAz microcontroller's serial output to USB. The test application was setup to send 100,000 packets and was conducted 5 times. During each run of the test every packet was received and processed. This test revealed that, even on modestly powered hardware, our gateway is able to maintain a packet reception rate greater than a MICAz/MIB520-USB based sink could possibly forward data to the gateway machine.

B. *Real-World Test-bed tests*

In addition to our load test, we performed the field testing of our developed gateway by running an example application in three independent small real-world test-beds, including the long-running backyard test-bed operated by the University of Pittsburgh [15]. The primary goal of these tests is twofold. Firstly, we want to find and eliminate any potential bugs that could appear only during real-world usage. Secondly, we want to verify that no significant issues arise when our gateway is being used in realistic and uncontrollable conditions. In all test-bed tests the sink node hardware is identical to our load testing and each individual mote in a WSN test-bed is a MICAz mote connected to an MDA300 [16] data acquisition board. The test software consists of the sink, which receives a CTP message on its radio and passes that message on to its USB interface, and the mote application, which samples temperature, humidity, and several analog-to-digital conversion (ADC) channels. Sampling is varied between sampling every 1 minute to sampling every 10 minutes.

Initial tests, primarily attempting to achieve our first goal, were run on a test-bed in our lab, consisting of five motes and a sink. During testing, each time a bug is found and eliminated a

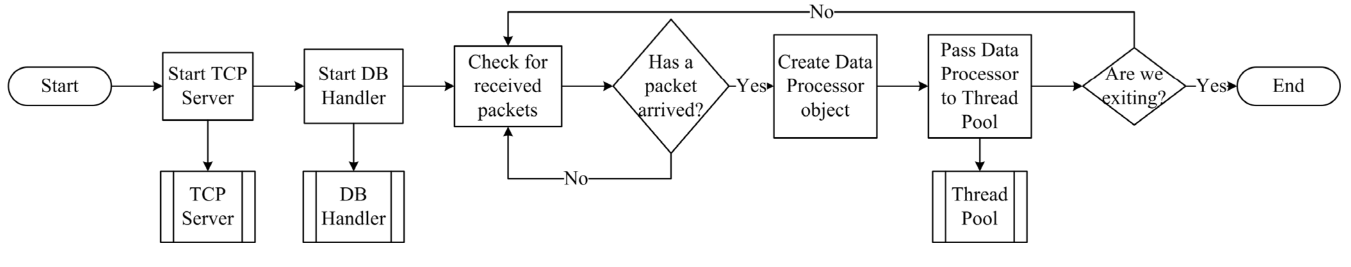

Figure 3. Flowchart of the main control thread of the gateway.

new test is started. Prior to testing in the Pittsburgh test-bed two tests were successfully performed for about two weeks.

We have performed testing at the University of Pittsburgh's backyard test-bed [15]. Fig. 4 shows the layout of the test-bed consisting of one sink and nine motes. The red node is the indoor sink and gateway machine. This test ran until the batteries were depleted, sampling once every 10 minutes. No gateway-related errors were experienced during this test.

In addition, we set up a seven mote indoors test-bed in Indianapolis for some further testing of our gateway system. This test resulted in the discovery and elimination of a minor bug in the gateway.

VII. Conclusion and Future Work

In this paper, we have explored the design and implementation of a general and configurable WSN gateway software system for data collection. Our presented and developed WSN gateway system can be easily configurable for nearly any WSN data collection applications. This system is configurable for both WSN research and practice, including the support of any newly developed WSN transport layer by researchers/users, static or dynamic packet formats, and various conversions of sensor data needed for diverse data gathering WSNs. Our developed gateway system has been thoroughly tested and validated via load test and real-world field tests including two indoors WSN test-bed and one outdoors WSN test-beds.

While our WSN gateway has performed well for data collections, we have plans to make our gateway more robust and full-featured in our future work. Firstly, we are working on developing a downstream command system of the gateway. By defining a command format and using callback functions we can develop this system in such a way that the users can provide their own command functionality. Secondly, we plan to extend our current gateway design to enable the user to integrate his/her new Data Processor by requiring only the addition of some information to the configuration file. Finally, we plan to introduce a user-friendly and robust error logging system using a pre-existing logging framework.

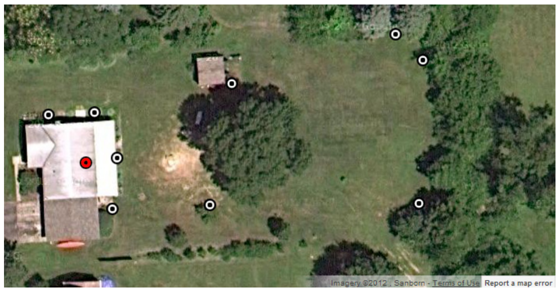

Figure 4. An illustration of the backyard WSN test-bed used in our gateway system testing.

Acknowledgment

This work is supported in part by National Science Foundation under grant CNS-0758372.

References

[1] M. Navarro, D. Bhatnagar, and Y. Liang, "An Integrated Network and Data Management System for Heterogeneous WSNs," in Mobile Adhoc and Sensor Systems (MASS), 2011 IEEE 8th International Conference on, 2011, pp. 819-824.

[2] N. Erratt and Y. Liang, "Compressed data-stream protocol: an energy-efficient compressed data-stream protocol for wireless sensor networks," Communications, IET, vol. 5, pp. 2673-2683, 2011.

[3] L. Steenkamp, S. Kaplan, and R. H. Wilkinson, "Wireless sensor network gateway," in AFRICON, 2009. AFRICON '09., 2009, pp. 1-6.

[4] L. Wu, J. Riihijarvi, and P. Mahonen, "A Modular Wireless Sensor Network Gateway Design," in Communications and Networking in China, 2007. CHINACOM '07. Second International Conference on, 2007, pp. 882-886.

[5] M. E. Raluca, M. E. Razvan, and A. Terzis, "Gateway Design for Data Gathering Sensor Networks," in Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON '08. 5th Annual IEEE Communications Society Conference on, 2008, pp. 296-304.

[6] H-C. Lin, Y-C. Kan, and Y-M. Hong, "The Comprehensive Gateway Model for Diverse Environmental Monitoring Upon Wireless Sensor Network," Sensors Journal, IEEE, vol. 11, pp. 1293-1303, 2011.

[7] H. Chen, D. Han, and H. Feng, "Research on standard architecture of sensor networks gateway," in Advanced Intelligence and Awarenss Internet (AIAI 2010), 2010 International Conference on, 2010, pp. 287-290.

[8] J. Kim and D. Choi, "esGate: Secure embedded gateway system for a wireless sensor network," in Consumer Electronics, 2008. ISCE 2008. IEEE International Symposium on, 2008, pp. 1-4.

[9] J. Suh and S. Jacob, "Distributed sensory systems and developer platforms from Crossbow technology," presented at the Proceedings of the 3rd international conference on Embedded networked sensor systems, San Diego, California, USA, 2005.

[10] TinyOS 2.x Working Group, "TinyOS 2.0," Proceedings of the 3rd international conference on Embedded networked sensor systems, 2005.

[11] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis, "Collection tree protocol," presented at the Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems, Berkeley, California, 2009.

[12] Crossbow Technology, "MICAz Wireless Measurement System," datasheet.

[13] Crossbow Technology, "Mib520 USB Interface Board," datasheet.

[14] Future Technology Devices International ltd, "FT2232D Dual ISN to Serial UART/FIFO IC," datasheet, Oct. 2006 [Revised April 2011].

[15] X. Liang, T. W. Davis, M. Navarro, D. Bhatnagar, and Y. Liang, "An Experimental Study of WSN Power Efficiency: MICAz Networks with XMesh," International Journal of Distributed Sensor Networks, vol. 2012, p. 14, 2012 2012.

[16] Crossbow Technology, "MDA300 Data Acquisition Board," datasheet.