Message Transformation Services for Wireless Sensor Networks (MTS-WSN)

Brian Carter and Rammohan Ragade
Computer Engineering and Computer Science
University of Louisville
Louisville, KY, 40292

Abstract - *This paper describes the design and implementation of message transformation services for wireless sensor networks. We examine a scheme to integrate heterogeneous sensors at the message level as part of our middleware research. Our framework is implemented on a base station to send and receive message packets from a variety of sensor network platforms and protocols. An advantage of our approach is in the ability to interact with the sensor at the lowest level, allowing for tighter control of integration. Experimentation in this paper provides the details of our framework for integration of the TinyOS platform. Our experimentation shows that our framework can indeed be beneficial to the construction of middleware for wireless sensor networks.*

Keywords: wireless sensor networks, middleware, transformation, messages.

1 Introduction

One of the current research challenges in wireless sensor networks is how to access and interact with multiple sensors and sensing devices. This is motivated by the need to provide more powerful applications that aim to discover, aggregate, and control multiple types of networks in parallel [1]. This is becoming a larger issue, with new Wireless Sensor Network (WSN) platforms being introduced at a rapid rate. Common platforms today include MICA2 Motes from Crossbow [2], ZigBee Tmote Sky from Moteiv [3], and many other platforms. Each platform has different protocols and applications for control and communication between the gateway and the sensors as shown in table 1.

Table 1: Components of WSN platforms.

	Crossbow MICA2	Moteiv Tmote sky
SN OS	TinyOS	TinyOS
SN Protocol	Surge Reliable	Sensornet
Radio	916 MHz	2.4 GHz, 802.15.4
Interface	RS232	USB
Gateway	MIB510	Tmote
Applications	Serial Forward Surge-View	Trawler

Past projects have limited their selection to one platform or vendor, deploying a homogeneous sensor network and storing the information on a fixed devise [4, 5, 6]. While this solution works for limited study, it is not practical for long term deployment where requirements change or new platforms are made available. To go beyond the niche applications, most researchers and companies will demand easy integration of the wireless sensor networks [7].

To address this issue, we present Message Transformation Services for Wireless Sensor Networks (MTS-WSN). MTS-WSN is a software framework that integrates diverse sensor networks platforms at the message level. The goal of our research is to provide a lightweight approach that directly interacts with the sensor gateway. This allows our framework to eliminate using multiple applications to collect the sensor data, each written for specific platforms and protocols on a variety of operating systems. The aim of MTS-WSN is to process the sensor readings into a consistent format and reporting units. Our goal is not to replace sensor protocols, but to provide a transformation scheme to integrate information provided from these protocols at the most effective level.

2 Sensor Network Overview

A sensor network is composed of wireless sensors, a gateway, and a base station as shown in figure 1. Typically, several homogenous wireless sensors, all installed with the same protocol application, communicate directly or through a multi-hop scheme to a gateway. The gateway is connected directly to the base station. More advanced gateways can connect to the base station wirelessly. Sensor messages flow from the wireless sensors to the gateway where it is processed on the base station. Sensor control messages flow from the base station to the gateway and then to the intended sensor.

As indicated in table 1, each platform has its own applications and protocols to transport the messages in the WSN. From our own experience, the biggest obstacles in incorporating sensor modules to applications are learning the specific software packages and understanding the sensor

readings that are extracted from the protocols. After researching several platforms, we found that messages could be processed directly from the gateway allowing MTS-WSN to be a viable option. The raw message bits are streamed in different formats, forcing initial research on the protocols that produced them. In the next sections, we present our MTS-WSN Architecture followed by our initial process for integrating sensor network platforms and concluding with our prototype implementation.

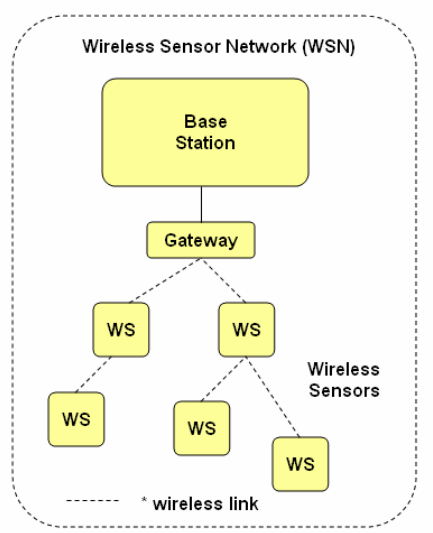

Figure 1: Typical WSN Architecture.

3 MTS-WSN

3.1 Architecture

MTS-WSN is designed to provide the bridge between WSNs and middleware applications. As shown in figure 2, it is responsible for receiving low level sensor messages from the sensor networks and converting them to a standardized message format.

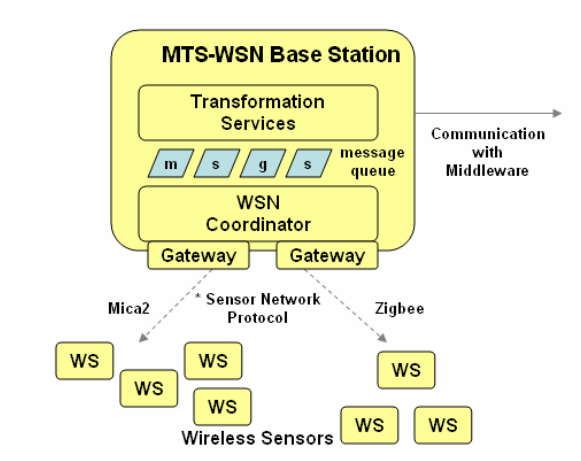

Figure 2: MTS-WSN Architecture.

It then forwards the message to the middleware for processing. Currently MTS-WSN supports TinyOS based sensor networks. Future prototypes will include support for other WSN hardware manufacturers. MTS-WSN is installed on the base station and is composed of the WSN Coordinator and the Transformation Services.

3.2 WSN Coordinator

The WSN Coordinator reads raw sensor data that is broadcasted from the sensor networks. Its role is to buffer and parse the messages. For each WSN platform that is connected, the WSN Coordinator assigns a port and configuration for the gateway. Figure 3 outlines the configuration that is required. The gateway receives and buffers the incoming data and then transfers it directly or via a serial or network connection to base station for processing. The base station receives the messages byte by byte, and then reassembles the message. A complete message is indicated by receiving a starting frame synchronization byte and receiving an ending synchronization byte. The complete message is added to the message queue for processing along with the platform information and a timestamp.

```
<platform id="xbow_mica2" protocol="surgereliable">
  <port name="com8">
    <baudrate>57600</baudrate>
    <parity>none</parity>
    <databits>8</databits>
    <stopbits>0</stopbits>
    <mode>hex</mode>
    <packet famesync="0x7E" />
  </port>
</platform>
```

Figure 3: Platform Configuration File.

3.3 Transformation Services

The Transformation Services process the messages that are in the queue and generates an XML message containing the computed values. The format for the sensor message is defined by the operating system and application protocol installed on the sensor. In the case of Tiny OS based sensor networks, the Transformation Services knows how to parse the sensor data by looking at the platform and protocol values. As new application protocols become available, the services can be configured with new interface implementation definitions that can read the new application specific data. This approach is similar to defining a circuit [8].

There are several steps involved with creating a new interface implementation definition based on the message format. This consists of two parts, a protocol definition for the operating system and one for the application. As shown in our MTS-WSN testbed section, we examine TinyOS using the Surge_Reliable application protocol. The same

process can be used for other platforms. As shown, creating the message definition is not a trivial task. When adding a platform for the MTS-WSN framework, the device message format must be well understood.

Once the sensor message has been deciphered, it is converted to a standard XML message format. The engineering values received are converted to standard scientific values. For example, the temperature readings from the MICA2 platform require a different conversion formula than the readings taken from the Moteiv platform. Each of the readings are converted and placed in a consistent environmental sensor XML format. This allows the middleware platforms to receive all sensory readings in the same structure and units. By converting the message and the sensory readings in the MTS-WSN framework, this eliminates duplication of this logic further downstream in the middleware platforms.

4 MTS-WSN Testbed

4.1 Overview

We have developed a prototype MTS-WSN sensor testbed that enables us to study the design issues of sensors messages using actual hardware as shown in figure 4. This has helped us to improve the MTS-WSN framework and to gain a better understanding of the impacts of the hardware. We found many issues that need to be addressed that were not found using a simulated environment. This includes receiving partial messages, interference, and network collisions.

4.2 Hardware Setup

An overview of the hardware setup is shown in figure 4. The wireless sensor network in our testbed consists of Crossbow MICA2 motes for sensors. The MICA2 motes use an Atmel ATmega128L microcontroller, 128KB of flash memory for code, 4KB of EEPROM for data, and a radio operating at 915MHz. Each of the motes has an MTS300 sensor board attached to collect light, temperature, and acoustics. The motes are running the TinyOS, a small open-source operating system designed for embedded wireless sensor networks. The Surge Reliable application protocol is installed on each mote, which is written in nesC, an extension of the C programming language.

We are using a MIB510 gateway for receiving messages from the sensor network. The gateway connects to a standard laptop running Windows XP and Visual Studio.Net 2005. An RS232 connection is used between the gateway and the laptop.

For comparison, a battery operated standalone temperature unit is used. This allows us to compare the indoor and outdoor sensor readings from the standalone unit to the readings received from the motes.

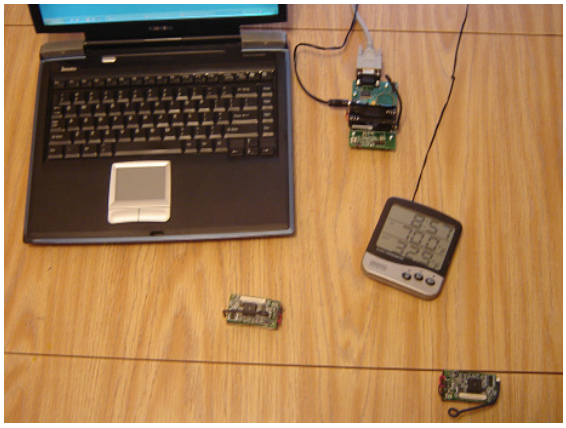

Figure 4: MTS-WSN Testbed.

4.3 Deciphering the TinyOS Message Packet

Our first task was to decipher the TinyOS message packet. We need a full understanding of this layout to develop the interface implementation definition that MTS-WSN uses. In this section, we walk through the processes for researching a new platform. For our testbed, there are four layers involved. This can be broken down into:

- TinyOS framework packet
- TinyOS message packet
- TinyOS multi-hop packet
- Surge Reliable message packet

We begin this process by showing a sample of a raw message received from the gateway:

```
7E 42 7D 5E 00 11 7D 5D 16 00 00 01 00 8B 00 00
00 00 00 00 00 62 00 80 C7 F5 7A FF FF 02 CC 3B
0F 7E
```

From our research, we found that the TinyOS framework packet has a beginning and ending frame synchronization byte of 0x7E. The packet type is the second byte followed by the message packet. The message packet is defined by the operating system, which for TinyOS is defined in the tos\types\AM.h file and shown in table 2. There are a few other conversions that must be handled in the message stream. TinyOS reserves certain values for control. To distinguish data from reserve codes, a prefix value (0x7D) is placed in the message stream to indicate the next byte has been OR'ed with 0x20. In our example, the address (7D 5E 00) is actually (7E 00), representing the destination address. The other conversion is byte swapping. The destination address is a 16 byte field which is 0x007E. The bytes must be swapped to represent the correct value. This is due to

the byte conversion that is required between TinyOS and Windows.

Table 2: TinyOS Message Format.

typedef struct TOS_Msg {	Message Bytes
Uint16_t addr;	7D 5E 00
Uint8_t type;	11
Uint8_t group;	7D 5D
Uint8_t length;	16
int8_t data[TOSH_DATA_LENGTH];	0x16 = 22 bytes
Uint16_t crc;	3B 0F
} TOS_Msg;	

Since this is a multi-hop protocol, the data packet in the TinyOS message must be deciphered. This is defined by the TinyOS multi-hop packet provided in the contrib\xbow\tos\lib\ ReliableRoute\MultiHop.h file. Table 3 shows the definition and the deciphering of the example message.

Table 3: Surge_Reliable Multi-hop Message Format.

Typedef struct MultihopMsg {	Message Bytes
uint16_t sourceaddr;	00 00
uint16_t originaddr;	00 01
int16_t seqno;	00 8B
uint8_t hopcount;	00
uint8_t data;	22 – 7 = 15 bytes
}attribute__((packed))TOS_MHopMsg	

The Surge Reliable message packet defines the remaining 15 bytes. The byte definition is provided from Crossbow in the xbow/apps/surge_reliable/surge.h file as shown in table 4.

Table 4: Surge_Reliable Message Format.

typedef struct SurgeMsg {	Message Bytes
uint8_t type;	00
uint16_t reading;	00 00
uint16_t parentaddr;	00 00
uint32_t seq_no;	62 00 80 C7
uint8_t light;	F5
uint8_t temp;	7A
uint8_t magx;	FF
uint8_t magy;	FF
uint8_t accelx;	02
uint8_t accely;	CC
}__attribute__((packed))SurgeMsg;	

Once the message has been deciphered, the engineering values gathered by the sensors must be converted to scientific readings. The temperature reading as indicated by 0x7A is first convert to decimal (122). From the MTS-MDA series user manual [2], the motes ADC output can be converted to Kelvin using the calculation shown in figure 5.

Conversion Formula:

$$1/T(K) = a + b \times \ln(R_{thr}) + c \times [\ln(R_{thr})]^3$$

where:
$R_{thr} = R1(ADC\_FS-ADC)/ADC$
a = 0.00130705
b = 0.000214381
c = 0.000000093
R1 = 10 k Ω
ADC_FS = 1023

ADC = Value from Mote's ADC measurement.

Figure 5: Temperature Conversion.

Once in Kelvin, the temperature units are converted to Celsius by subtracting 273.15. Our reading of 122 is converted to 23.046 Celsius or 73.48 Fahrenheit. The messages are deciphered and transformed to a standardized XML format defined for that type of sensor. For our testbed, we have defined an environmental sensor XML format.

4.4 MTS-WSN Framework Implementation

Our target platform for MTS-WSN is Microsoft C#.Net as shown in figure 6. We implemented the prototype using the version 2 of the .Net framework, since it now supports direct communication through ports in the System Object.

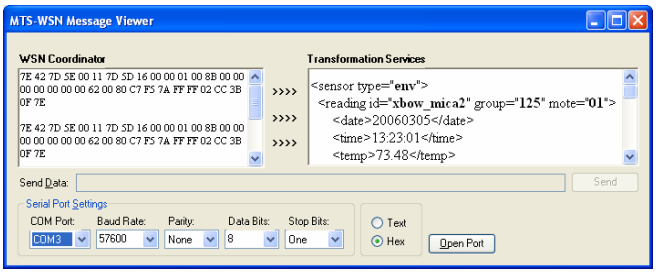

Figure 6: MTS-WSN Prototype.

The received message bytes are cached until a full message has been received. The WSN Coordinator hands the message off to the Transformation Services which creates the XML format and perform the appropriate sensory conversions. The XML format provides a standardized

layout to be integration into middleware frameworks and applications.

Our current implementation supports the MICA2 platform and the Surge Reliable protocol. Through experimental validation, ZigBee and other protocols are proposed to enhance our testbed.

5 Conclusion

New wireless sensor hardware is being developed at a rapid rate along with new application protocols. In this paper, we presented a scheme for integrating WSN platforms at the message level. Our testbed indicates that this is a viable option, eliminating many of the platform specific applications. From our experience, MTS-WSN is a very useful research tool to study heterogeneous sensor networks at the lowest level.

6 Acknowledgements

The authors acknowledge the fellowship support from Southern Region Education Board (SREB) and equipment grants from the University of Louisville.

7 References

[1] Jonathan Ledlie, Jeff Shneidman, Matt Welsh, Mema Roussopoulos, and Margo Seltzer, “Open Problems in Data Collection Networks”, 11th ACM SIGOPS European Workshop, Leuven, Belgium, September 2004.

[2] Crossbow Corporation: http://www.xbow.com.

[3] Moteiv Corporation: http://www.moteiv.com.

[4] Mainwaring, Alan. Polastre, Joseph. Szewczyk, Robert. Culler, David. Anderson, John, “Wireless Sensor Networks for Habitat Monitoring”, First ACM Workshop on Wireless Sensor Networks and Applications, September 28, 2002. Atlanta, GA, USA.

[5] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, “Monitoring volcanic eruptions with a wireless sensor network”, in Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN’05), Jan. 2005.

[6] J. Burel, T. Brooke, and R. Beckwith, “Vineyard computing: Sensor networks in agricultural production”, IEEE Pervasive Computing, vol. 3. no.1, pp. 38-45, 2004.

[7] Kristi Hobbs, “Wireless Sensor Networking Software, The Next Generation”, Sensors, February 2006, pp. 23-25.

[8] Jeff Shneidman, Peter Pietzuch, Jonathan Ledlie, Mema Roussopoulos, Margo Seltzer, Matt Welsh, “Hourglass: An Infrastructure for Connecting Sensor Networks and Applications”, Harvard Technical Report TR-21-04.