THREE-TIER WIRELESS SENSOR NETWORK INFRASTRUCTURE FOR ENVIRONMENTAL MONITORING

by

WEI HAN

B.Eng., Beijing University of Technology, P.R. China, 1993
M.Eng., Asia Institute of Technology, Thailand, 1995

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2011

Abstract

A two-tier wireless data communication system was developed to remotely monitor sediment concentration in streams in real time. The system used wireless motes and other devices to form a wireless sensor network to acquire data from multiple sensors. The system also used a Stargate, a single-board computer, as a gateway to manage and control data flow and wireless data transfer. The sensor signals were transmitted from an AirCard on the Stargate to an Internet server through the General Packet Radio Service (GPRS) provided by a commercial GSM cellular carrier. Various types of antennas were used to boost the signal level in a radio-hostile environment. Both short- and long-distance wireless data communications were achieved. Power supplies for the motes, Stargate, and AirCard were improved for reliable and robust field applications. The application software was developed using Java, C, nesC, LabView, and SQL to ensure seamless data transfer and enable both on-site and remote monitoring. Remote field tests were carried out at different locations with different GPRS signal strengths and a variety of landscapes.

A three-tier wireless sensor network was then developed and deployed at three military installations around the country – Fort Riley in Kansas, Fort Benning in Georgia, and Aberdeen Proving Ground in Maryland - to remotely monitor sediment concentration and movement in real time. Sensor nodes, gateway stations, repeater stations, and central stations were strategically deployed to insure reliable signal transmissions. Radio signal strength was tested to analyze effects of distance, vegetation, and topographical barriers. Omni- and Yagi-directional antennas with different gains were tested to achieve robust, long-range communication in a wireless-hostile environment. Sampling times of sensor nodes within a local sensor network were synchronized at the gateway station. Error detection algorithms were developed to detect errors caused by interference and other impairments of the transmission path. GSM and CDMA cellular modems were used at different locations based on cellular coverage. Data were analyzed to verify the effectiveness and reliability of the three-tier WSN.

THREE-TIER WIRELESS SENSOR NETWORK INFRASTRUCTURE FOR ENVIRONMENTAL MONITORING

by

WEI HAN

B.Eng., Beijing University of Technology, P.R. China, 1993
M.Eng., Asia Institute of Technology, Thailand, 1995

A DISSERTATION

submitted in partial fulfillment of the requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2011

Approved by:

Major Professor
Dr. Naiqian Zhang

Abstract

A two-tier wireless data communication system was developed to remotely monitor sediment concentration in streams in real time. The system used wireless motes and other devices to form a wireless sensor network to acquire data from multiple sensors. The system also used a Stargate, a single-board computer, as a gateway to manage and control data flow and wireless data transfer. The sensor signals were transmitted from an AirCard on the Stargate to an Internet server through the General Packet Radio Service (GPRS) provided by a commercial GSM cellular carrier. Various types of antennas were used to boost the signal level in a radio-hostile environment. Both short- and long-distance wireless data communications were achieved. Power supplies for the motes, Stargate, and AirCard were improved for reliable and robust field applications. The application software was developed using Java, C, nesC, LabView, and SQL to ensure seamless data transfer and enable both on-site and remote monitoring. Remote field tests were carried out at different locations with different GPRS signal strengths and a variety of landscapes.

A three-tier wireless sensor network was then developed and deployed at three military installations around the country – Fort Riley in Kansas, Fort Benning in Georgia, and Aberdeen Proving Ground in Maryland - to remotely monitor sediment concentration and movement in real time. Sensor nodes, gateway stations, repeater stations, and central stations were strategically deployed to insure reliable signal transmissions. Radio signal strength was tested to analyze effects of distance, vegetation, and topographical barriers. Omni- and Yagi-directional antennas with different gains were tested to achieve robust, long-range communication in a wireless-hostile environment. Sampling times of sensor nodes within a local sensor network were synchronized at the gateway station. Error detection algorithms were developed to detect errors caused by interference and other impairments of the transmission path. GSM and CDMA cellular modems were used at different locations based on cellular coverage. Data were analyzed to verify the effectiveness and reliability of the three-tier WSN.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to express my deepest gratitude to my major professor, Dr. Naiqian Zhang, for his academic guidance, supervision, encouragement, and financial support throughout my Ph.D. study. His wisdom, knowledge and commitment to the highest standards inspired and motivated me. His patience and caring for students, his hard working and seriousness in research, and his independent thinking and open-minded attitude towards novelties will serve as a model for my future professional life.

It is a pleasure to extend my gratitude to my supervisory committee: Dr. Mitch Neilsen, Dr. Gurdip Singh, Dr. Donald Lenhert and Dr. Ning Wang. Thanks to Dr. Neilsen for his kind assistance in using Linux system on the Stargate and providing ideas of Internet network programming. Thanks to Dr. Singh for his suggestions in real time system, and providing ideas in wireless sensor network. Thanks to Dr. Lenhert for his encouragement and providing valuable knowledge on using microcontroller. Thanks to Dr. Wang for spending her precious time on guiding and helping the entire research project. I also thank Dr. Donald Fenton for willing to serve as my outside chairperson.

I am grateful for assistances and suggestions from many experts. Thanks to Dr. Floyd Dowell for his assistance and providing access to the USDA tower. Thanks to Mr. Carl Johnson for his great effort in field experiments. Thanks to Mr. Darrell Oard for his advice and assistance in wireless sensor network field deployment. Thanks to Mr. Brandon Lee Lantz for helping us in field installation.

Deep gratitude is extended to Dr. Joseph Harner, Head of the Biological and Agricultural Engineering department, for his support and encouragement.

I would like to express special thanks to Dr. Kyeong-Hwan Lee, Dr. Yali Zhang, Dr. Peng Li, Ms. Ning Tang, Mr. Alan Bauerly, Ms. Ling Xue, Mr. Daniel Bigham, Mr. Matthew Worcester, Mr. Peyman Taher, Mr. Joseph Dvorak and Mr. Xu Wang for their assistance.

I am heartily thankful to the department staff, Ms. Barbara Moore, Mr. Randy Erickson, Ms. Cindy Casper, Ms. Arlene Jacobson and Ms. Lou Ann Claassen for helping me on all aspects of my study and research in the department.

This is a great opportunity to express my respect to K-State Badminton Club for making it a memorable journey during my stay at K-State. Also thanks to all of my friends for their support and encouragement.

My profound appreciation and gratitude are given to my beloved family for their encouragement, support, patience and unwavering love.

Lastly, I offer my regards and blessings to all of those who supported me in any respect during the completion of my study at Kansas State University.

CHAPTER 1 - INTRODUCTION

1.1 General background

1.1.1 Environmental monitoring

The demands on environmental monitoring are increasing as we strive to better understand and manage risks to human health and ecosystems (Sharpe, 2003). Environmental monitoring is a technique to efficiently gathering crucial data from environment. These data must be based on reliable, accurate information. The old days of sending people to the remote sites to collect data are obsolete. This method was too prone to errors of inconsistency.

In response to these issues, environmental analysts have sought improvements in laboratory-based analytical methods as well as portable solutions that allow sampling and analysis to be undertaken reliably on-site. The need for up-to-date information implies a need for continuous data collection. Electronic environmental monitors can be preset with sampling frequencies from a second to several hours. They can perform inside a freezer or inside an oven. They can be shipped and placed on many different locations and be relied upon to collect their data 24 hours per day and 7 days per week. The data is stored safely and can be transmitted over a wireless network or downloaded on request to a computer where it can be analyzed.

1.1.2 Monitoring of suspended sediments

Suspended sediments are fine soil particles that remain in suspension in water for a considerable period of time without being in contact with the bottom of the water body. Such material remains in suspension due to the upward components of turbulence and currents (USGS, 2010). Monitoring of suspended sediments is important in water quality assessment and is the way to observe potential contaminations in the water body. Water and waste treatment processes may cause suspended sediment to increase and should be monitored carefully (Zhang, 2009). Army training activities are responsible for soil erosion within and around military installations which causes decline of water quality. Many programs of the Department of Defense (DoD), such as the Strategic

Environmental Research and Development Program (SERDP) and the Environmental Security Technology Certification Program (ESTCP), are designed on assessing the water quality and reducing soil erosion (SERDP, 2010; ESTCP, 2010).

In this study, suspended sediment in streams within or near military installations was monitored using an optical sediment sensor, which was originally designed by Stoll (2004) and further developed by Zhang (2009).

1.2 Wireless sensor network

During the recent years, tremendous attentions have been directed at Wireless Sensor Networks (WSN). The low-power electronic devices integrated with radio functions and sensors were an epoch in computer science and other interdisciplinary fields. The WSN can be developed at a low-cost to operate in unattended environments. The WSN can be used for detecting relevant quantities, monitoring and collecting data, assessing and evaluating information, formulating meaningful user displays, and performing decision-making and alarm functions. The challenges for WSN are enormous (Lewis, 2004).

A typical WSN includes numbers of spatially distributed sensor nodes and a few gateway stations. The sensor nodes in the WSN monitor environmental conditions, such as temperature, pressure, vibration, sound, light, motion, or pollutants. It passes the data to remote users through gateway stations. The activities on the sensor nodes can be controlled by remote user if a bi-directional communication is enabled in the WSN (Wikipedia, 2011). Figure 1.1 shows the architecture of a typical WSN.

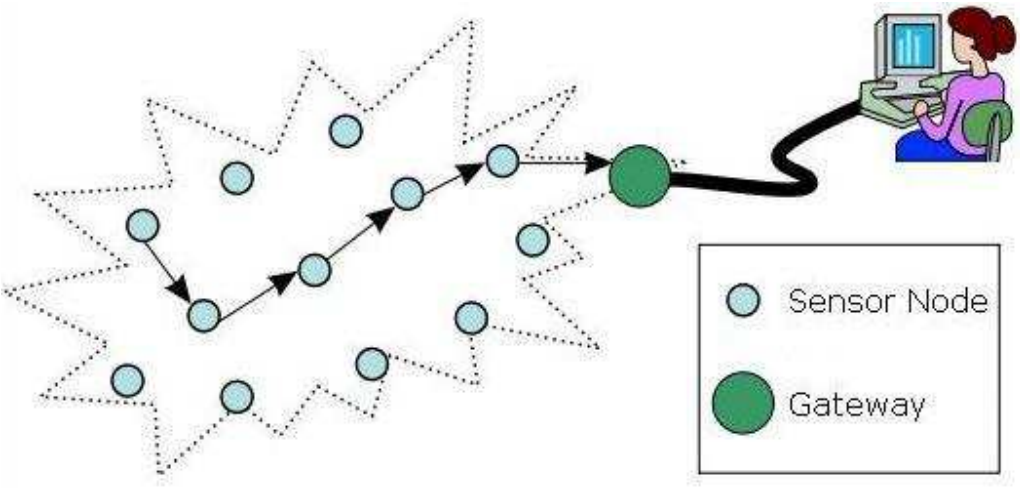


Figure 1.1 Typical wireless sensor network architecture (Wikipedia, 2011)

A sensor node is a node in the WSN that has the ability to process information, gather sensory data, and communicate with other connected nodes or gateway stations. A sensor node usually has a radio transceiver with internal antenna or connection to an external antenna, a microcontroller, an interface to the sensors and a power source using battery or other power harvesting resources. Figure 1.2 shows a typical architecture of a sensor node (Wikipedia, 2011).

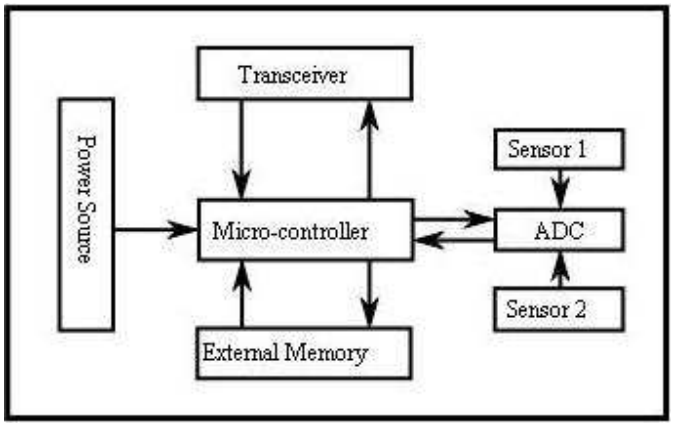


Figure 1.2 Typical architecture of the sensor node (Wikipedia, 2011)

A gateway station is a node in the WSN that interfaces a sensor network to the external world, including remote users. It gathers data from all connected sensor nodes. Compared with a sensor node, a gateway usually has additional abilities to process more information, access larger internal or external resources, such as memory, Internet connection, USB, RS-232 and other communication ports.

A WSN is usually developed for a particular application at low power and low cost. It usually does not require a complex, general-purpose operating system such as Microsoft Windows or Linux. The operating system for WSN is similar to an embedded system. TinyOS is an operating system specifically designed for WSN based on an event-driven programming model. TinyOS supports nesC programming language which is built as an extension to the C programming language. LiteOS is another operating system developed for WSN and it supports C programming as well. Contiki is an operating system for WSN, which uses a simpler programming style in C (Wikipedia, 2011).

Many researchers and companies have been studying various real-time monitoring systems using WSN. However, a full-fledged technology for suspended sediment monitoring is not available. The wireless-hostile environment for typical suspended sediment monitoring is an obstacle to most WSNs. In this study, to develop a reliable and practically useful, real-time wireless system, we introduced two-tier and three-tier WSNs. The main hardware components of this WSN included motes, Stargates, spread-spectrum radio dataloggers, and cellular data service devices. Software was developed on a TinyOS platform using the nesC programming language.

CHAPTER 2 - OBJECTIVES

The general objectives of this study were to develop a remote, real-time environmental monitoring system using two-tier and three-tier wireless sensor networks.

The specific objectives were:

1) Based on the characteristics of a Suspended Sediment Concentration (SSC) sensor, develop wireless sensor networks to monitor sediment in streams in a real-time manner and to:
    - investigate the feasibility of using wireless sensor network for SSC monitoring;
    - select wireless devices based on SSC sensor data structure and data rate;
    - develop electrical circuits to interface the SSC sensors with the wireless sensor network;
    - design a gateway station for data collection and handling; and
    - develop data transmission protocols and algorithms.

2) To develop a mid-to-long range wireless data transmission system and a data storage mechanism and to:
    - select equipment for mid-range wireless data transmission;
    - select sites that best serve data transmission based on radio propagation models;
    - design protocols for data communication;
    - develop long-range distance data transmission methods using commercial cellular data services;
    - develop a reliable database for data storage; and
    - design and develop a web publication system for data display.

3) To test the wireless environmental monitoring system in the laboratory and field and to:

- test soil sediment measurement and flow velocity measurement using optical sensors;
- test the precipitation and water temperature measurements;
- test an air blast cleaning system for the optical sensor;
- develop a deployment mechanism for the network system as a turnkey solution;
- study and test energy harvesting and power saving plans for a long system life span;
- test auto-recovery methods for a reliable and self-healing system;
- test the network system in the field;
- study methods to achieve low data loss rate and error rate; and
- keep the system cost at a low level.

4) To study the potential of and limitations of the wireless sensor network for environmental monitoring.

CHAPTER 3 - LITERATURE REVIEW

3.1 Wireless communication system

Wireless communication is one of the most active areas of technology development. This development is driven primarily by the transformation from what has been largely a medium for supporting voice telephony into a medium for supporting other services, such as transmission of video, images, text, and data (Wang and Poor, 2003). Current research describes many approaches for achieving wireless communication. The wireless network technologies can be classified by communication ranges (Wang et al., 2006).

Short-range approaches for wireless personal area networks include Bluetooth, ZigBee, and Ultra-wideband (UWB). Bluetooth is a protocol designed for low power consumption in a short range. A system for automated irrigation using Bluetooth was developed with successful results (Kim et al., 2006). ZigBee protocols are intended for use in embedded applications requiring low data rates and low power consumption. As an example, a WSN monitoring and control system using ZigBee was developed to monitor the environment within a greenhouse (Zhou et al., 2007). Ultra-wideband (UWB) is a technology for transmitting large amounts of data over a wide spectrum of frequency bands with low complexity, low cost, and low power consumption. This technology has an enhanced capability to penetrate through obstacles (Yang and Giannakis, 2004). USB Dongle WUWBD-101 is designed based on the UWB technology and is available for the market (Gemtek, 2008). Other short-range methods, including Crossbow's (Crossbow, 2008) MICA2, provide high-level functional integration designed to extend the wireless mesh networking technology into a wide variety of sensing applications. An example is a multi-hop wireless sensor network using MICA2 to monitor wildfire behavior changes due to temperature, relative humidity and wind (Hartung et al., 2006).

Wireless local area network (WLAN) is an example of a mid-range technology implemented to extend or substitute for a wired LAN (Chan, 2007). WLAN also is

referred to as the IEEE 802.11 family of standards. It consists of access points that provide access to different WLAN users.

Broadband wireless access (BWA) is a long-range communication technology. BWA is an interface specification aimed at providing high-speed wireless access for wireless metropolitan area networks (Eklund et al., 2002). Nortel (Nortel, 2007) offers an end-to-end WiMAX solution, which is an example product for this broadband technique.

Cellular phone systems also are categorized as long-range technologies. The evolution of the mobile network for major wireless systems can be clarified as follows. The second generation ("2G") systems include GSM, launched in 1991, and IS-95 CDMA, first deployed in 1995. These two technologies evolved to General Packet Radio Service (GPRS) and CDMA2000 1xRTT, respectively, to provide data service that exceeds the second generation ("2.5G"). Enhanced Data-rate GSM Evolution (EDGE), a third generation ("3G") standard, reached a higher data rate with minimal modifications in GPRS. GPRS and CDMA2000 1xRTT then continuously evolve to Universal Mobile Telecommunication System and CDMA2000 3x (or CDMA2000 EV-DO Rev. B), respectively (Chan, 2007).

Microwave-link radio frequency technology also is included in the long-range transmission category. Although not well known by end-users in a network, it is an important component of wireless communication. One example is Campbell Scientifics' CR200 series datalogger, which has a communication range of up to 16 km. MaxStream also produced many kinds of radio frequency modems for indoor or outdoor, with ranges up to 22 km (MaxStream, 2008).

Methods for global range, such as satellite and meteor burst communications, also exist. SatWest offers a satellite network system ideal for small teams or individuals with high data requirements (SatWest, 2008).

Meteor burst communication (MBC) is an alternative method for a sink node. The basic principle is simple: billions of meteors are caught by earth's atmosphere everyday. Most of the meteors are burned up and create meteor trails. Although these meteor trails last only a few seconds, it is long enough to reflect radio waves (Healy et al., 1989). By utilizing the ionized trail of gases left from the entry and disintegration of the meteors, people are able to create communication networks between different points by reflecting

signals on the disintegrating meteor's trail of ionized gases. These signals can be received in a receiver station which is located up to 1600 km away (Brown, 1985).

The presence of a MBC link is dependent on the arrival rate of meteors from space. The maximum number of trails per hour occurs at around 6AM local time, with a minimum trail count occurring at about 6PM. The ratio between these values is approximately 4 to l. This cyclical variation is due to the rotation of the Earth and its passage through space. Over an hour, the MBC channel is generally available for less than 2% of the time (Handley et al., 1992).

Low probability of intercept (LPI) and anti-jam (AJ) is the primary attractions of MBC. It is affected less by various natural and man-made ionospheric disturbances, including nuclear explosion. Statistically, the average burst duration is 0.58s while the average time interval between bursts is 10s. An average throughput of 75 bps could be achieved with a few hundred watts of transmitted power. The frequency range for radio propagation is 30-100 MHz (Oetting et al., 1980).

A typical MBC network usually has the following components and structures: A large master base station has approximately 5,000 watts of radio power and a large antenna array. Any numbers of remote sites have a 100 watts radio and a directional antenna on each of them. The master station can be connected with all those remote stations. At a predetermined power level, a continuously, coded probe signal in a certain direction and angle was transmitted by the master station. When a meteor comes with proper speed, size, and trajectory, the remote station receives the signal reflected back to the earth by the meteor from the master station. When the remote station received the probe signal, it becomes "awakened" from its idle status, and it starts to decode the probe signal, turn on its own transmitter, and send a response signal back along the trajectory path to the master station. On condition that the link is reliable enough for correct reflection, data can be exchanged. Data transfers between master and remote stations are serial bursts of high-data-rate transmissions due to the extremely short duration of a meteor trail may appear. For moving larger sets of data that cannot be transmitted within one burst, multiple successive bursts can be used (Cumberland et al., 2004).

MBC is a reliable form of communication; however, it has certainly not become a great success. It is not recommended for highly interactive applications. Nevertheless, for

users who require more reliability, low system cost, two-way messaging over medium to long distance, MBC is one of the interesting choices (Brodsky, 1990).

In summery, the strong points of MBC are simplicity of implementation, easy unattended operation, low initial and running cost, reliability and survivability. The weak points are long delay, low throughput and high transmission power.

Although the MBC technology has had a long history, the development in both theory and practical is not full-fledged. Two major drawbacks were the main reason hindering the development of MBC: low throughput and long message waiting time. However, as stated by Healy et al. (Healy et al., 1989), appropriate network topology, proper transmission protocol, and good routing algorithms can alleviate the impacts of these drawbacks and improve the efficiency. A successful implementation of MBC may provide a secure and reliable link for environmental monitoring systems, and a novel technique for the WSN.

3.2 Wireless sensor network for remote monitoring

There are numerous applications in observing material characteristics of the world, such as temperature, light, humidity, acceleration and sound. Examples include monitoring habitat of seabirds (Mainwaring et al., 2002), behavior of rats (Osechas et al., 2008), cold-chain for temperature-sensitive products (Carullo et al., 2009), structural health (Xu et al., 2004), water transmission pipeline networks (Stoianov et al., 2007; Lin et al., 2008) and volcanic eruptions (Werner-Allen et al., 2005). These applications are a few examples from a seemingly boundless realm of WSN. While versatile WSNs bring us an exciting new approach, various challenging technical issues are still waiting for investigation. The challenges include power-aware design (Marín et al., 2005; Johann et al., 2008), data aggregation (Krishnamachari et al., 2002; Commuri et al., 2008; He et al., 2004), path loss model for signal transmission (Turkka and Renfors, 2008; Darr and Zhao, 2008), energy harvesting (Niyato et al., 2007; Raghunathan et al., 2006; Corke et al., 2007), deployment strategies (Younis and Akkaya, 2008; Xu et al., 2005), error correction (Sanchez et al., 2007; Mukhopadhyay et al., 2009), Medium Access Control (MAC) protocol (Demirkol et al., 2006), routing protocol (Mao et al., 2008), and synchronization (Ren et al., 2008). The corner stone for the WSN design techniques is the

energy efficiency algorithms to prolong the lifetime of the system. This is especially true for wireless system deployments in hard-to-retrieve locations.

The existing WSN monitoring systems are great successes in observation of environment. They opened a new entrance for exploring the physical characteristics of the world. However, for remote areas lacking terrestrial telecommunication networks and basic infrastructure such as power supply, the Internet access, or telephone service, none of them give a substantial solution. To realize a continuously sediment-runoff monitoring, we need to evaluate the following technical challenges.

Data aggregation has been selected as a fundamental prototype for wireless routing in sensor networks. The idea is to combine the data coming from different sources to eliminate redundancy, minimize the number of transmissions and thus save energy. This prototype shifts the focus from the traditional address-centric approaches for networking to a more data-centric approach. The method presented some suboptimal data aggregation tree generation heuristics which achieved significant energy gains. These gains are greatest when the number of sources is large, and when the sources are located relatively close to each other and far from the sink (Krishnamachari et al., 2002). There are advanced schemes for data aggregation. One method is to implement a dynamic data aggregation using a Field Programmable Gate Arrays (FPGAs), which gives the necessary flexibility in data aggregation techniques demanded by real-time applications (Commuri et al., 2008). Other people proposed an adaptive application-independent data aggregation in a time sensitive manner, so that the degrees of data aggregation can be varied based on network traffic using a feedback control (He et al., 2004). However, for the purpose of transmitting massive data such as the stream flow velocity, we do not have an efficient yet simple solution.

Path loss model is another topic of attraction in WSN. Some researchers provide practical values for path loss models which are useful for mobile wireless network planning (Turkka and Renfors, 2008). Although the study of empirical propagation models for mobile phone channels has been exhaustive, similar models for WSN are yet to be investigated. A two-dimensional path loss prediction model was developed in poultry layer facilities. The model was able to predict 86% of the system variability and was able to produce an average error of -0.7 dB for all combined points. It can be used to

a similarly designed poultry environment (Darr and Zhao, 2008). However, there have not been reports on similar empirical models for WSN in areas near water bodies such as small rivers or creeks.

Different energy harvesting technologies, including solar panel, vibration-based energy harvesting technology and thermoelectric devices (Niyato et al., 2007), have been studied. Solar-energy harvesting through photovoltaic conversion provides the highest power density, which makes it the paradigm of choice to power a sensor node. In order to take into account the temporal variations in the energy harvester's output, Raghunathan et al. (Raghunathan et al., 2006) showed a system that carefully scheduled the node's computation, communication, and sensing tasks. An experiment used a wireless system with a solar panel charging two AA size NiMH batteries for each sensor node. In an area where the daily average sunshine hours range from 6.5 to 8.5 hours, the system has run for more than two years (Corke et al., 2007). However, there is a lack of literature to explore the unknowns for larger systems that utilize solar energy continuously and require more power for gateways and base stations for long-term operations.

A Forward Error Correction (FEC) technique was evaluated by researchers. By adding information bits into the data packet, the Bit Error Rate (BER) for received packets was obviously reduced (Sanchez et al., 2007). However, their method added the burden of overhead on the data packet and was not efficient for a WSN with energy-constrained resources. A novel approach to handle multiple sources of errors with low-cost and small footprint devices was presented. It utilized the spatial and temporal correlations residing in sensing results to correct errors from various sources. This achieved correction of all types of transient errors in a simple and efficient way, and imposed no overhead at the sensor node (Mukhopadhyay et al., 2009). The method greatly enhanced data reliability but with limitations. It cannot be used for a system designed to monitor data with faster variations than the sampling rate of the system. Also, the computation requirements for the gateway are increased tremendously. Another drawback is that the collection of correlation information depends on the redundancy of the sensing data, which contradicts the purpose of data aggregation.

Besides putting a lot of enthusiasm and research efforts on all aspects of the sensor nodes, the communication methods for gateway and sink node are also crucial for

a reliable and robust wireless system. If the gateway fails, all sensor nodes connected to that gateway would die. If the sink node faults, all WSN through that sink node would be lost. Considering with throughput, cost, and energy consumption, people choose different devices as their sink node. Direct connection to the Internet is preferred; however, most locations of interest are remote and have no such infrastructure. General Packet Radio Service (GPRS) is one of the popular choices (Stoianov et al., 2007; Barrenetxea et al., 2008). A more expensive solution for sink node could be a satellite link (Mainwaring et al., 2002; Mathioudakis et al., 2007).

3.3 Wireless sensor network for precision agriculture

Adopting WSN technology in precision agriculture is intriguing and practical. There are researchers investigating the WSN for precision agriculture using Bluetooth (Zhang, 2004). Although challenges such as battery life and transmission latency exist in his application, his work gives hopes for the future of WSN in agriculture applications. A method was presented for measuring environmental parameters in a greenhouse using WSN (Liu et al., 2007). After sinking data to a terminal, this terminal transfers data by short message service from a GSM system. The small size of the WSN device is a double-edged sword. The battery-powered sensors and wireless devices enable flexibility in deployment but shorten the useful lifetime of the system. The existing WSN systems did not give a long-term, low cost solution for areas far away from Internet access or power supply.

With these ideas and problems in mind, the purpose of our research was to build an unattended, robust, and remote real-time monitoring system; and provide the design details and configuration of this remote WSN, with specific emphases on system reliability, power source life span, transmission range, and cost effectiveness.

CHAPTER 4 - TWO-TIER WIRELESS SENSOR NETWORK

The real-time sediment-runoff monitoring system was deployed at three locations. Before these deployments, individual modules were tested separately in the laboratory.

4.1 Network architecture

The conceptual configuration of the system, including sensor, communication components, and monitoring backbone servers, is illustrated in Figure 4.1.

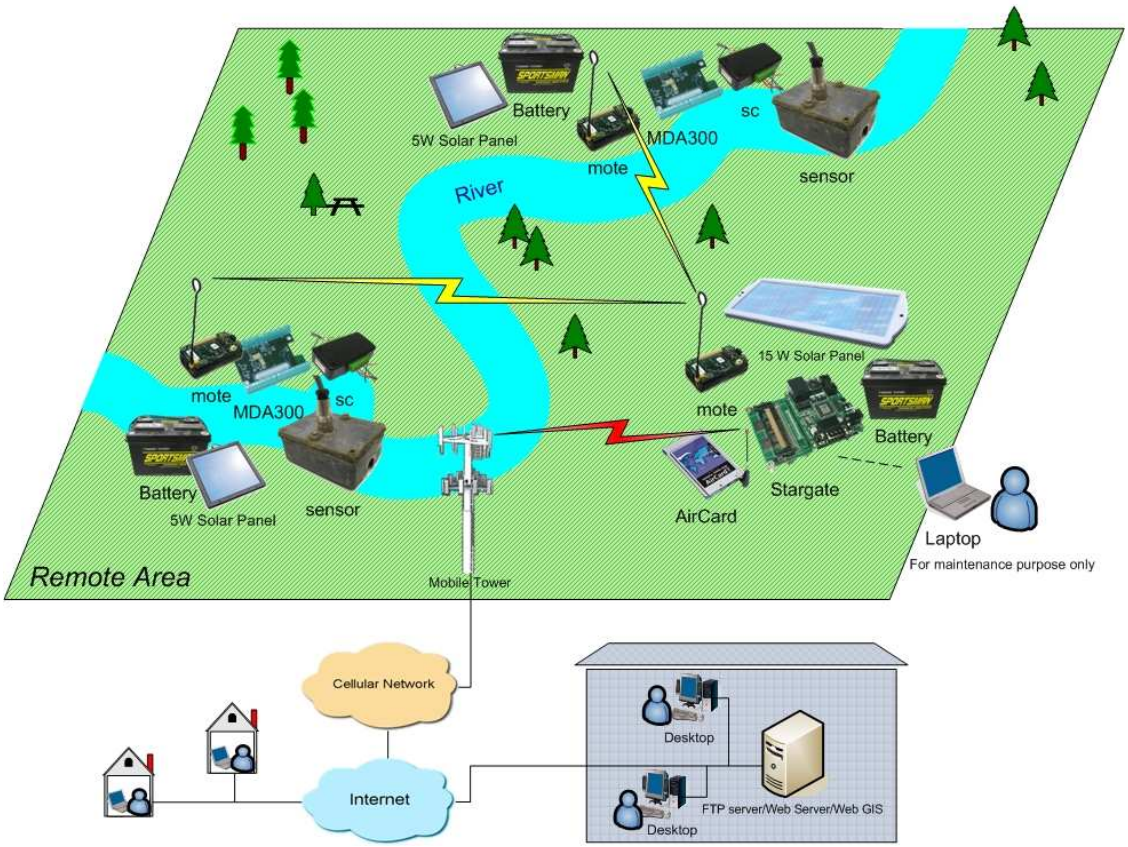


Figure 4.1 Conceptual configuration of a real-time sediment runoff monitoring system

The system was divided into two parts: a remote system and an indoor server computer system. In Figure 4.1, the components displayed within the green zone labeled “remote area” belonged to the remote system. They were optical sensors, signal

conditioners, motes, data acquisition boards, car batteries and a solar panel that charged the battery. A Stargate had a mote mounted on it and was equipped with AirCard 750 to wirelessly communicate with the server computer. The Stargate was powered by the car battery and the solar panel. For the outdoor environment, all equipment was protected in weather-proof enclosures and was wired properly.

An FTP server was set up on the indoor server computer to receive remote files transferred through the GPRS data service. The received data files were stored in a database. Both Java and C languages were used to ensure data processed in a timely manner. Raw data were processed and converted to engineering unit automatically using the SQL programs in the database.

In our preliminary wireless communication tests, we designed a network with a few motes transferring data to a mote sitting on the Stargate wirelessly, and then receiving data in a computer via an RS-232 cable. This architecture was used for testing mote transmission range, battery life and sensor control. It was also used for checking data processing operation on the Stargate.

To make the system more practically useful, we decided to add an AirCard 750 to the Stargate so that we could use the cellular data service to further transfer data to a remote server computer. With the AirCard, the mobility of the system was greatly enhanced. Using this architecture, we carried out a few field tests in several locations.

4.2 Methodology

4.2.1 System components

4.2.1.1 Sediment sensor

Concentration of soil sediment, a proximate prediction index of the natural water condition, must be monitored continuously to ensure that changes can be observed. An optical sensor was used to measure suspended sediment concentration (SSC) (Zhang et al., 2007). The sensor was also modified to measure flow velocity in a recent study. To get accurate in-situ data in a river or creek, the sensor must be fully submerged into water.

The prototype optical sensor was designed by Stoll (2004). Several wavelengths were selected to measure the spectral responses for various water types. Further

developed by Zhang (2009), a few more practical models of the sensor were developed and deployed in field test for measuring SSC and flow velocity in real time.

The principles of the optical sensor were illustrated in Figure 4.2. A light emitting diode (LED) with designated wavelength emits the lights for a period of time, three phototransistors (PT) located at different angels from the incident light of the LED receive and measure transmitted (180°), scatted (90°) and backscattered (45°) lights, respectively. Based on an analysis of the spectral responses measured in different water samples, a prediction model was developed to estimate the SSC.

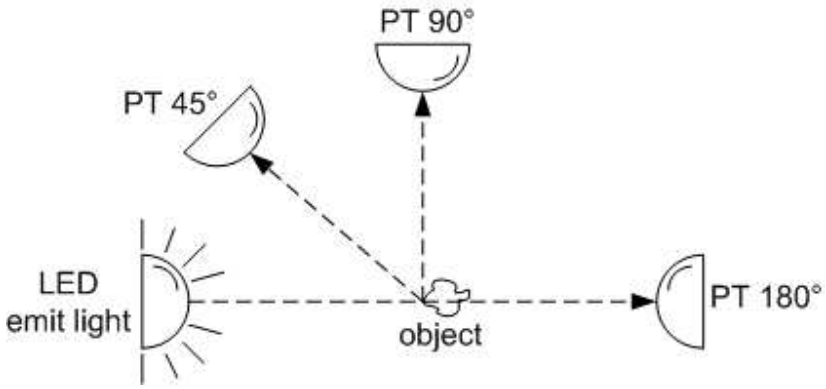


Figure 4.2 Illustration of the principle of an optical sensor

LED light sources of three colors were used in each optical sensor. Those colors were blue-green, orange and infrared. Based on a statistical analysis on colors of the LED and receiving angles of the phototransistors, design of the optical sensor was simplified (Zhang, 2009). Out of nine combinations of these colors and receiving angles, only three were actually selected for the design. They were infrared light received at 45° (IR45), orange light received at 45° (ORA45) and orange light received at 180° (ORA180).

Optical sensors are sensitive to ambient lights (Zhang, 2009). In order to reduce the influence of ambient lights, we measured both the signals received by the phototransistors when the LED light was turned on and off. Then we subtracted the signal measured when the LED was off from that when the LED was on, hence minimizing the ambient lights effect.

Four generations have been designed for the sensor over the past years. Figure 4.3 shows a prototype of the latest (4th generation) design. The sensor and signal cable were sealed in a black polyvinyl chloride (PVC) case and had a well shaped channel for water flow. It also had embedded air passages for cleaning the optical lenses of the LEDs and phototransistors. This design, compared with other anti-fouling mechanisms, was found effective in reducing biofouling.

Figure 4.3 Prototype of the 4th generation optical sensor

Additional orange LED and two phototransistors (ORA45-2 and ORA180-2) were placed on this sensor. These additional LED-phototransistor pairs worked with the original orange LED-phototransistor pairs (ORA45 and ORA180) to measure the flow velocity. A dye injection mechanism was required to create sufficient marker for this sensor while performing the flow velocity measurement (Zhang, 2009).

4.2.1.2 Signal conditioner

A printed circuit board (PCB) was designed for signal conditioning. It was used to amplify the sensor data and adjust the signal range. The board also provided a 5 VDC regulated power and control signals for the optical sensors. It was an interface between a

data acquisition board and the optical sensor. More complicated design related to mote and sensor control will be discussed in details in Chapter 5.

4.2.1.3 The motes

The motes, also called smart dusts, enable a low-power wireless sensor network. The third generation motes (MICA2, Crossbow Technology Inc., San Jose, CA) are used to achieve an ad-hoc mesh network. In the network, the main role of the mote is controlling sensor, collecting data, and then performing short distance wireless communication.

For this study, we selected MICA2 as our mote device. MICA2 utilizes Atmel ATMega 128L as its core processor which is a low-power, CMOS, 8-bit microcontroller based on AVR enhanced reduced instruction set computing (RISC) architecture (Atmel Corporation, 2006). The operating voltage is 2.7 - 3.3 VDC, which can be easy satisfied by two AA size batteries or an external power supply. A 900 MHz FSK modulated radio frequency in an unlicensed ISM band is used for MICA2 wireless communication. The radio transmission power is adjustable. The maximum data transfer rate is 38.4 kbps, which is greater than the required 9.6 kbps for the application in this study. The outdoor line of sight radio transmission range is 150 m. A 51-pin expansion connector supports analog inputs, digital I/O, I2C, SPI and UART interfaces (Crossbow, 2003). With a proper driver program installed, adequate power supply, and the MICA2 mounted on a data acquisition board (MDA300) through a 51-pin expansion connector, the measured sensor data are ready for broadcast. Figure 4.4 shows a MICA2 mote.

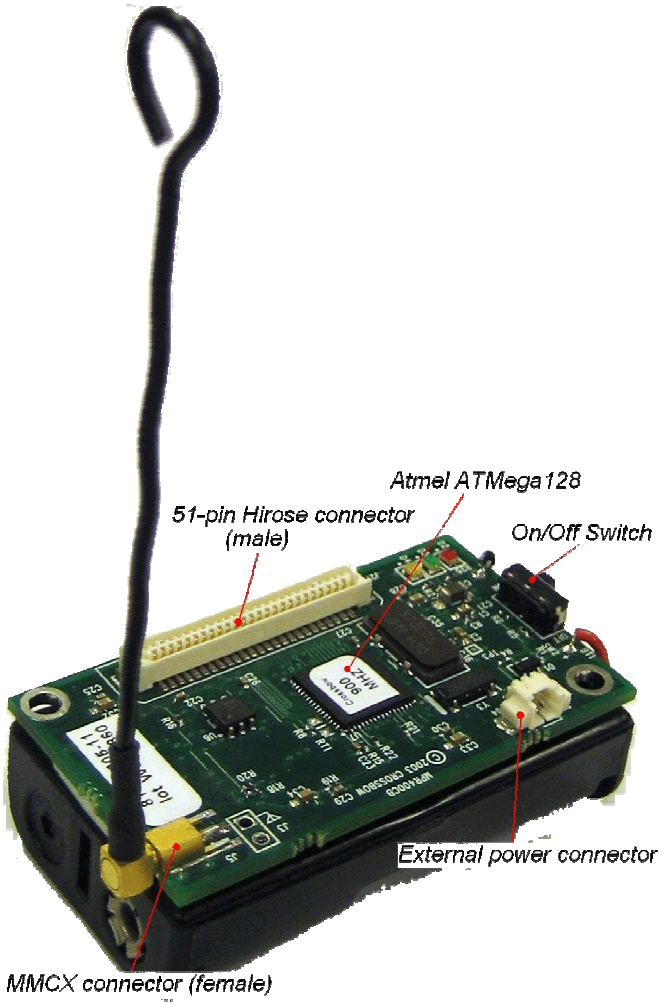


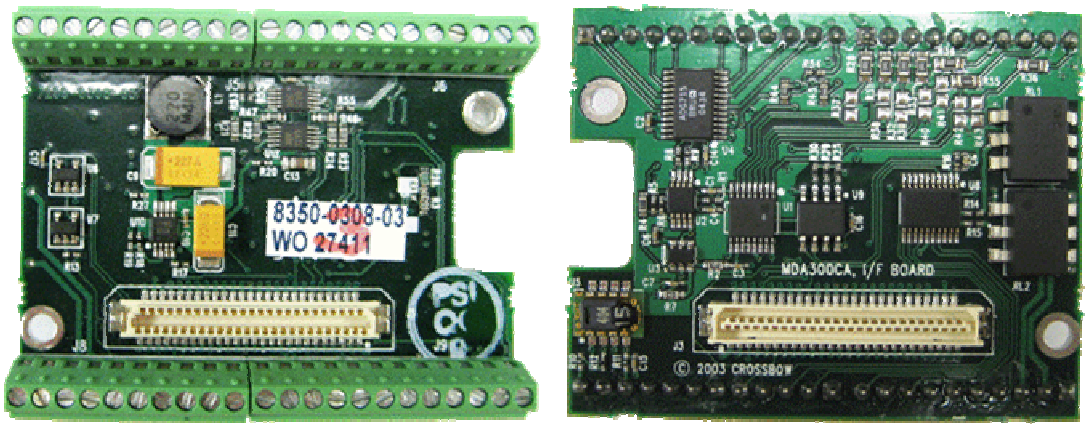
Figure 4.4 Top view of a MICA2 mote

In later experiments, we also used a new mote product MICAZ. Functionalities of MICAZ are similar to or slightly enhanced from MICA2, but they use a 2.4 GHz, direct sequence spread spectrum radio for communications. The maximum data rate for MICAZ is increased to 250 kbps. This was especially beneficial to us when the raw data for velocity measurement needed to be transmitted. Figure 4.5 shows an example of MICAZ mote. The antenna was obviously shorter than that for MICA2 (Figure 4.4) because of the higher radio frequency was used.

Figure 4.5 Top view of a MICAZ mote

4.2.1.4 Data acquisition board

A DAQ board (MDA300CA, Crossbow Technology Inc., San Jose, CA) was used to process the analog outputs from the sensors. It was also used to control relays. The board has a 12-bit analog-digital converter (ADC) with an 8-channel multiplexer, the sampling rate is 50 kHz (ADS7828, Texas Instruments, Dallas, Texas). The DAQ board allows eleven analog inputs channels, six digital I/O ports and two relays. Figure 4.6 shows the top and bottom views of the MDA300CA board. The 51-pin expansion connector can be used to connect a mote seamlessly.



(a) **(b)**

Figure 4.6 (a) Top view of a MDA300CA board. (b) Bottom view of a MDA300CA board

On this board, seven channels (A0 to A6) are single-ended analog channels. The input range for the analog signals is 0-2.5 V. Four additional channels (A7 to A10) are differential analog channels. Three excitation channels and two LED channels occupy the rest slots of the analog channels. For input signals of higher than 2.5 V, a voltage divider is necessary to scale down the voltage levels.

The ADC reading can be converted to voltage by using following equation (Crossbow, 2007).

$$V_{out} = 2.5 \times D_{ADC} / 4096 \qquad \textbf{(4.1)}$$

where:

V_{out} = Output analog voltage reading

D_{ADC} = ADC digital reading

Six digital channels (D0 to D5) can be used for digital input and output. When used as output, it is an open-collector circuit, and needs an external pull-up circuit to adjust to desired output voltage levels. When they are set as inputs they have internal pull-up resistance so that they can be directly used (Crossbow, 2007). Two additional digital channels are used as relays. One relay is normally open and the other is normally closed. In this study, we used digital outputs to control sensors and a digital input as a counter. Relays were also used for sensor control.

4.2.1.5 The gateway

We used a Stargate (SPB400, Crossbow Technology Inc., San Jose, CA) as an unattended remote gateway for control, routing and communication. As a high-performance and low-power processing platform designed for sensor, signal processing, and controlling, the Stargate can be used for many wireless sensor networking applications. A 32-bit, 400 MHz, Intel PXA255 XScale RISC processor is used with 64 MB SDRAM and 32 MB flash memory. It has a standalone, open-source Linux operating system with kernel version 2.4.19 (Crossbow, 2006). When it is equipped with a compact flash (CF) with an appropriate capacity (bigger than 1 GB) and PCMCIA connectors, the Stargate can store application programs and more than one year's worth of data. This guaranteed that backup data are always available, even in the worst wireless transmission scenario.

With an additional daughter card (SDC400CA, Crossbow Technology Inc., San Jose, CA), the Stargate also supports a variety of interfaces, including RS-232, 10/100 Ethernet, USB port, and JTAG (Crossbow, 2006). A mote attached on the Stargate serves as a base station that can receive wireless data from one or multiple remote motes. Figure 4.7 shows a Stargate with a daughter board on it.

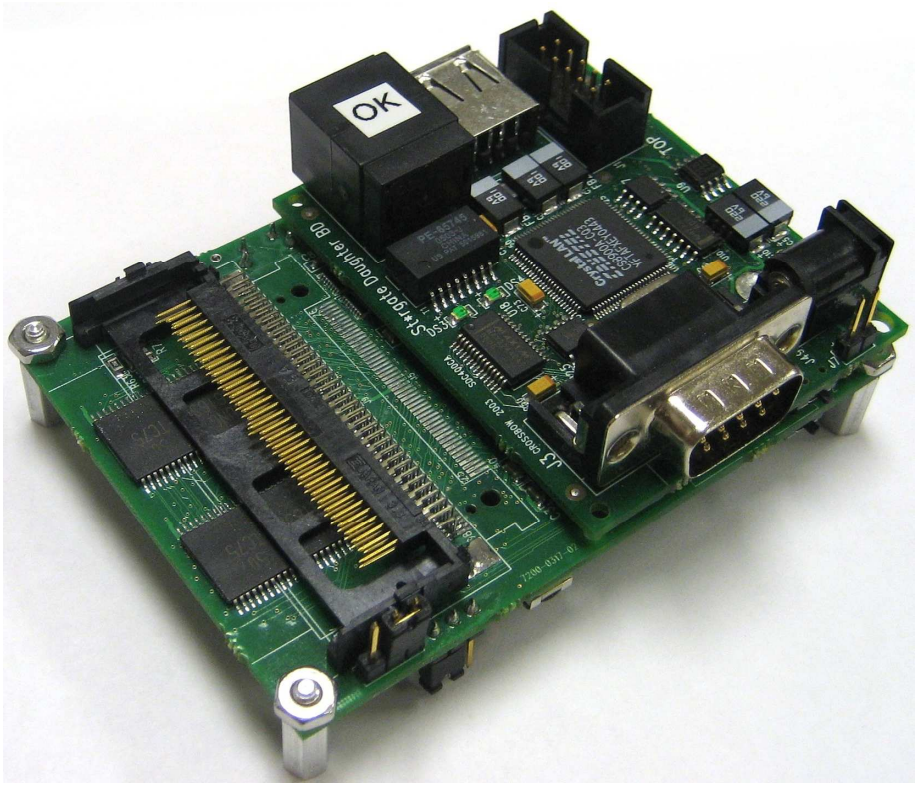


Figure 4.7 Stargate and daughter board as a gateway

4.2.1.6 Cellular communication using AirCard and GPRS

The cellular communication was designed as the last stage of wireless data transmission in the system. Mobile communication is feasible and accessible in most areas with human activities. A PCMCIA connector on the Stargate makes it possible to install any standard AirCard available in the market.

There are many different types of AirCards for mobile phone systems provided by different carriers, such as Code Division Multiple Access (CDMA) from Verizon Wireless and Global System for Mobile communications (GSM) from T-Mobile. We chose the AirCard 750 (AirCard 750, Sierra Wireless, Richmond, BC, Canada) based on the availability of the required data transfer service in the study area. The AirCard is essentially a cell phone built into a PCMCIA card. The AirCard greatly improved the mobility of our system, extending it, theoretically, to anywhere with compatible mobile data services. This is especially important to rural areas, where limited or no Internet access is available. Figure 4.8 shows an AirCard 750 with a whip antenna on it.

Figure 4.8 AirCard 750 with a whip antenna

General Packet Radio Service (GPRS) is a mobile data service for GSM system. The maximum speed of a GPRS connection is comparable to a modem connection in an analog telephone network (about 32-40 kbps). In today's market, all mobile service providers offer data service as well as voice service. We chose T-Mobile's GPRS simply because their data service had the same coverage as their voice service in our study area. We also used AT&T and Verizon data services for other locations. We will discuss these two data services in Chapter 5.

4.2.2 Power supply and conservation

A 12 V car battery was used as a power source for the remote system. Efficient power usage was vital for longevity of the outdoor systems. This study required appropriate power management for both sensor and wireless data communication.

For each sensor data scanning, the sampling interval was designed to be 30 seconds, this allowed a near real-time data sampling while saving energy for extra data communication in case a shorter sampling interval was applied. As designed, each LED on the optical sensor was turned on for 96 ms and then off within every 30 seconds period, and this time period could be controlled by the mote. This scheme minimized sensor's power consumption.

As programmed, the mote always stayed in a low power consumption mode after finishing the control and data transfer tasks. With a 30-seconds data scanning interval, two AA size batteries lasted an average of 4.5 days in the test. Instead of using two AA size batteries, a 3.3 V voltage regulator could be used for power supply. A 12 V car battery used for mote and sensor scanning lasted more than one month. In a later study, we used a 5 W solar panel to recharge the battery, extending its lifetime to a few months. The Stargate and AirCard were the two main power consumers in the entire system. The current drained by the Stargate could reach up to 500 mA while the AirCard was in operation. Because we used a 12 V battery as the power supply and the Stargate needed only a 5 V power supply, in the worst case, $(12V-5V)\times 0.5A = 3.5W$ power was wasted on the voltage regulator's heat sink. This also generated a heat dissipation problem in the small enclosure used to protect the Stargate and the AirCard for outdoor operation. To solve this problem, in addition to add air ventilation, we used a switching voltage regulator (DE-SW050, Dimension Engineering, Akron, OH). This voltage regulator drained current only when necessary. This design change reduced heat build up and significantly increased the battery life.

4.2.3 Data transfer protocols

The AirCard on the Stargate provides a point-to-point protocol (PPP) link across Internet via the cellular service. The PPP link is a method for transmitting datagram over serial links. It encapsulates datagram and many other networking protocols for transporting IP traffic, assigning and managing IP addresses. It does not impose any restrictions regarding transmission rate (Cisco, 2010). Once established the PPP link connection, we built a link between the central gateway (Stargate) and the server computer using a commercial GPRS data service. This link remained connected until we sent command to explicitly close it, or until other external event occurred (power outage, hardware failure, etc.). In the software design, we must have the PPP link to be checked to ensure none disruption data transmitting. We will discuss this further in section 4.2.5 .

After obtaining the PPP link, we used a file transfer protocol (FTP) to transmit data from the Stargate to the server computer. FTP is a standard network protocol used to copy a file from one host to another over a TCP/IP-based network, such as the Internet.

FTP is built on a client-server architecture. It utilizes control and data connections separately between the client and server. FTP is used with user-based password authentication or with anonymous user access (Wikipedia, 2010a).

A freeware software (Markov, 2006) was chosen to setup FTP server on the server computer. The FTP client, Stargate, did not require to installing any additional FTP software. It used the existing ftp command to make TCP connection to the server on port 21 to establish a control connection. This connection remained open for the duration of the communication session. From its port 20, the FTP server machine made a data connection to the Stargate and required the start of file transfer. At the same time, the control connection was used for session management information exchange between the client and the server.

The server responded on the control connection with three-digit status codes in ASCII, e.g. “200” represented that the last command was successful. A file transfer in progress over the data connection could be aborted using an interrupt message that was sent over the control connection. We used these FTP status codes to determine the progress of data transfer as well as the FTP connection.

FTP can be run in either the active or the passive mode. The difference between these two modes is the way that the data connection is established. In the active mode, the client sends the server the IP address and a port number on which the client will listen, and the server initiates the data connection. In situations where the client is behind a firewall and is unable to accept incoming data connections, the passive mode may be used. In this mode the client sends a PASV command to the server and receives an IP address and a port number in return. The client uses these to open the data connection to the server (Wikipedia, 2010a). Since we could adjust the firewall to open the necessary ports on the FTP server computer, we used active mode for the FTP connections.

4.2.4 Database and Internet presentation

For the two-tier WSN we developed in this study, the numbers of records generated were limited. We chose Microsoft Access 2003 as our data storage database. Compared with other sophistic databases such as Microsoft Structured Query Language (SQL) server, Oracle, the Microsoft Access database could be managed with lower

maintenance. The database programming was simple and the technical support was strong.

Microsoft Access database saved records into tables. Table could be designed based on the structure of records. In this study, raw data received from the FTP connection were saved into a table named “tblOriginal. Saved raw data not only kept the original data format, but also provided help information for troubleshooting. Immediately after the raw data were written into the database, a few relational database SQL programs, so called stored procedures, were used to validate the new data and to parse or convert the raw data into separate user-readable values, and then to save them to a result table named “tblResults” automatically. We will discuss the details of the stored procedure programs in section 4.2.5.3 .

An Internet information service (IIS) was set up in the server computer as a web server, which provided the real-time data access on the Internet. We used Active Server Pages (ASP) to run the web publication tasks. Basic access authentication was enforced to provide a secured website. Web server-side programming helped to realize dynamic graphics refreshing in real-time. The web showed the newest data and graphics on its first page. Selected data could be viewed and downloaded from anywhere in the world.

4.2.5 System software design

The system software included mote programs, the Stargate data processing and communication programs, server computer data receiving and processing programs, and web server and publication programs. These programs were written in various languages, including nesC, Linux shell script, Java, C, SQL and Active Server Pages (ASP). Language selections were based on their availability on different operating platforms, also considered the compatibility between different software languages.

4.2.5.1 Mote program

There were basically two types of programs installed on the mote devices – the programs for transmitting or receiving data. The original programs were developed by Intel Corporation. We further developed these programs based on their open source license. The programs were all written in nesC to realize the structuring concepts and execution models of TinyOS. The nesC program could be compiled and debugged using

Cygwin, which was a Linux-like environment for Windows. It provided substantial system call functionality, and a collection of tools, offering a Linux look and feel (Red Hat, 2010).

The program for transmitting data controlled the optical sensor sampling every 30 seconds. The LED lights were turned on and off in sequence with a duration of 96 ms. As we tested under the laboratory condition, this duration could be further reduced to 6 ms without causing significant signal distortion, such as reduction in magnitude. However, one second on and off period used on CR5000 datalogger was already satisfactory for optical sensor sampling (Zhang, 2009), to give a safe margin to avoid signal distortion due to a short LED's on or off duration for the real-time sampling, we chose 96 ms in the program. By using multiple timers in a chain connection manner, i.e. one parent timer called the next timer before it expired, the sample reading instance was designed precisely in the middle of each LED's on or off time period. This ensured that no sample could be taken at the moment of LED's on and off transient. The sampling timing scheme is illustrated in Figure 4.9.

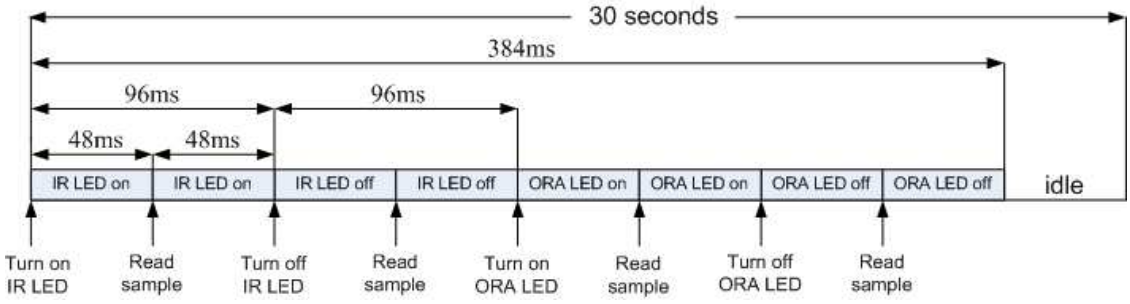


Figure 4.9 Sampling timing scheme for one sensor

The program also wrote digital data into a predefined record, also called packet, as shown in Figure 4.10, and then sent them to a base mote on the Stargate. We defined each field in the date payload as a "channel". The digital data was saved in a hexadecimal format, and each channel in data payload had 4 hex digits which represented a maximum number of 65,535 in decimal. Considering the data range of measurement, this was large enough for our transmitted records.

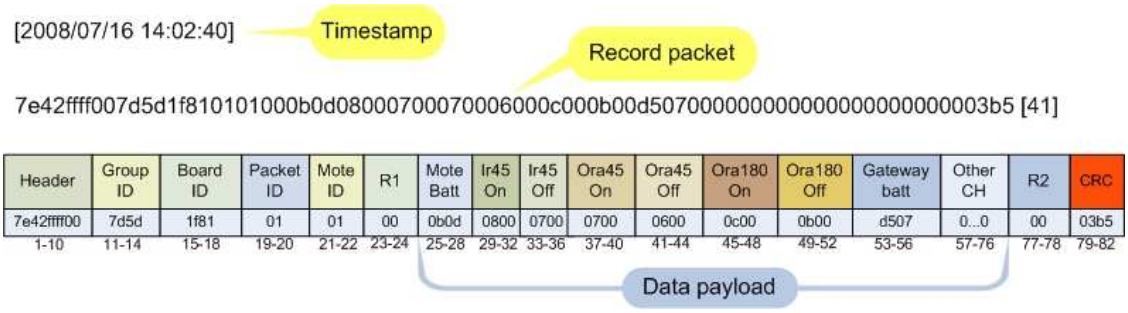


Figure 4.10 Layout of a typical record and its timestamp

Description of each field in the record was given in Table 4.1.

Table 4.1 Description of each field in the record

Name	Description	Length (bytes)
Header	Including record beginning number, MDA300 board ID	5
Group ID	Identification number of a mote group (Group ID "125" was used for all motes in this study)	2
Board ID	Identification number of sensor or DAQ board	2
Packet ID	Identification number of a packet	1
Mote ID	Identification number of a mote	1
R1	Reserved	1
Mote Batt	Mote battery voltage	2
Ir45 On	Reading received at 45° when IR LED was turned on	2
Ir45 Off	Reading received at 45° when IR LED was turned off	2
Ora45 On	Reading received at 45° when ORA LED was turned on	2
Ora45 Off	Reading received at 45° when ORA LED was turned off	2
Ora180 On	Reading received at 180° when ORA LED was turned on	2
Ora180 Off	Reading received at 180° when ORA LED was turned off	2
Gateway batt	Gateway battery voltage	2
Other CH	Five other channels (2 bytes each)	10
R2	Reserved	1
CRC	Cyclic redundancy check (CRC) for error detection	2

Defined by the mote program, the header of the record was always started with 7e in hex (0x7e). To avoid confusion with the data following the header, 0x7e should not be allowed to show up inside the payload data. Consequently, the program automatically checked and changed any value equal to 0x7e to 0x7d5e. This was called a number escape. In this way, 0x7e never appeared after the first two bytes in the packet. Again, 0x7d used for the number escape could be confused with 0x7d used for data in the payload. Similarly, the program changed any 0x7d used for data in the payload to 0x7d5d. In this way, 0x7d was used for number escape only in the payload. As a consequence, in the receiving side, we must always find any number escape and change them back to the original values, and then perform the rest of data processing tasks. Because of this number escape algorithm, the total length of each record varied between 41 and 43 bytes.

There was only one application program that could be executed in a given time on the TinyOS environment. However, the TinyOS's concurrency model allowed multiple threads to run within one application program (Crossbow, 2005). In our sensor control nesC program, after initiating the control component, there were two threads running at the same time. The first thread was called main thread, which took care of collecting data and sent them to the radio channel. The second thread was called timer thread. It was started by the main thread, and took the responsibility for controlling the LED lights and sampling at appropriate time instance. Once started, these two threads ran forever until the mote was stopped by the user or when an unexpected event occurred.

Figure 4.11 gives a flowchart for the mote sensor control and data transmitting program. The source codes of the program are given in the Appendix A.

For receiving program on the base mote, we directly used the original nesC programs from Intel.

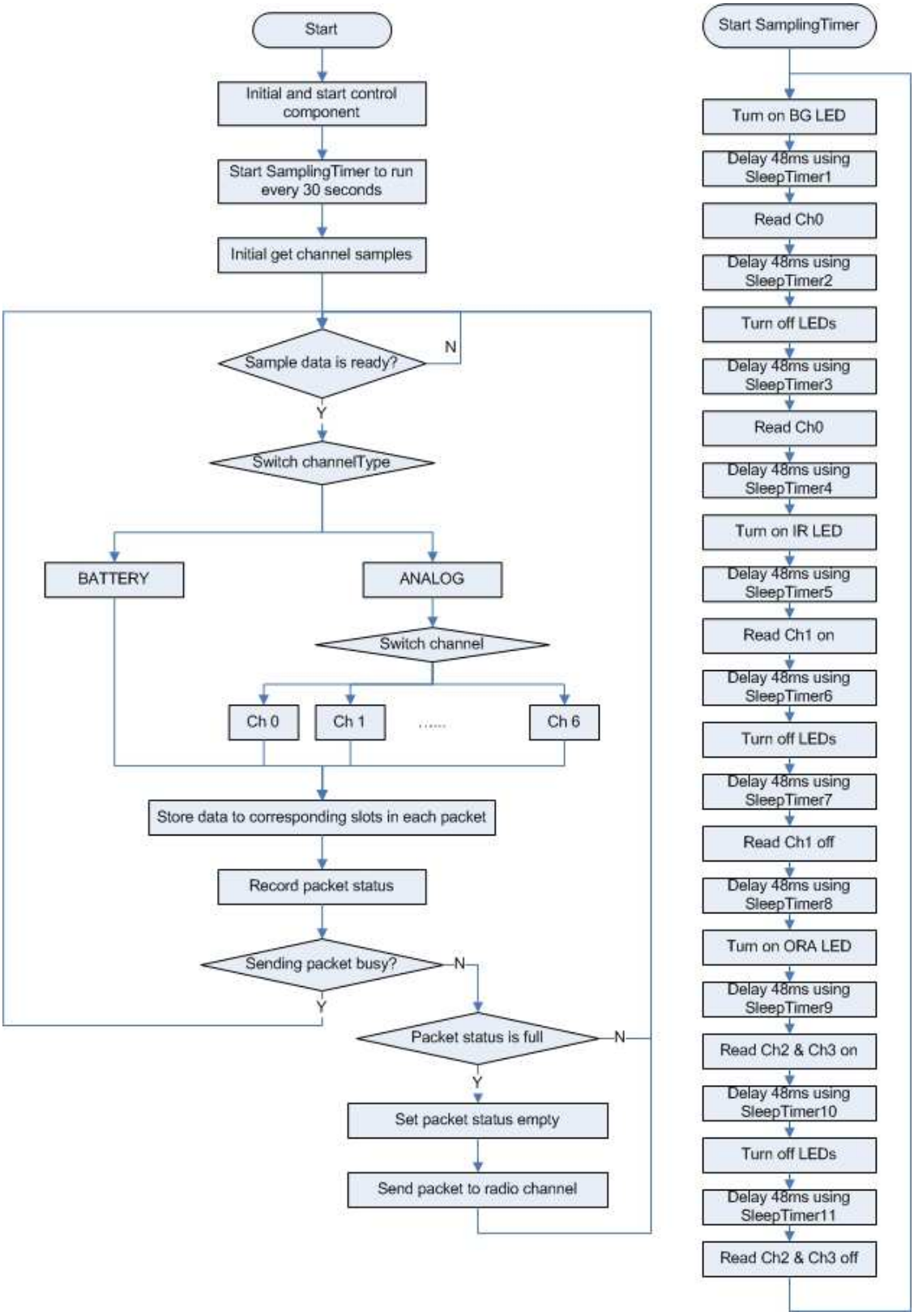


Figure 4.11 Flowchart for the mote sensor control and data transmitting program

4.2.5.2 Stargate program

As mentioned in section 4.2.1.5 , the Stargate had an open source Linux operating system on it. Hence, we could use Linux shell script to program for data collection and transmission. Shell script was written for command line interpreter of the Linux system, and it could be used for manipulating files, executing programs and printing texts.

Once the Stargate was started, it automatically executed a few programs in the sequence based on their priority settings. The first program called was a program (PPP.sh) that used the “pppd” command to call AirCard 750 and establish a PPP link. It also requested a 30 seconds time delay in the Stargate to wait for the AirCard’s response. Next to it, another program (cfcard.rc) checked the system’s date and time. The time could be synchronized with a time server (tick.ucla.edu) through the Internet connection. After that, a watchdog timer (WDT), a hardware timing device that triggered a system reset if the program hung or neglected regular services, was initiated. The program for starting WDT was named as “stargate-watchdog”. Now, the Stargate was ready for receiving data and sending them back to the server computer. These programs were very small and simple programs. The source codes are listed in Appendix B.

There was a script program called “xfer.rc”. The function of this program was to call data collecting script “Xlisten.sh”, FTP dead connection clean script “killID.sh”, FTP connection checking script “do-clean-ftp.sh”, and FTP file sending script “ftp_test_02.sh”. It also stopped these programs when the system was stopped. The flowchart of the starting data collection and FTP program is shown in Figure 4.12. The source code is listed in the Appendix B.

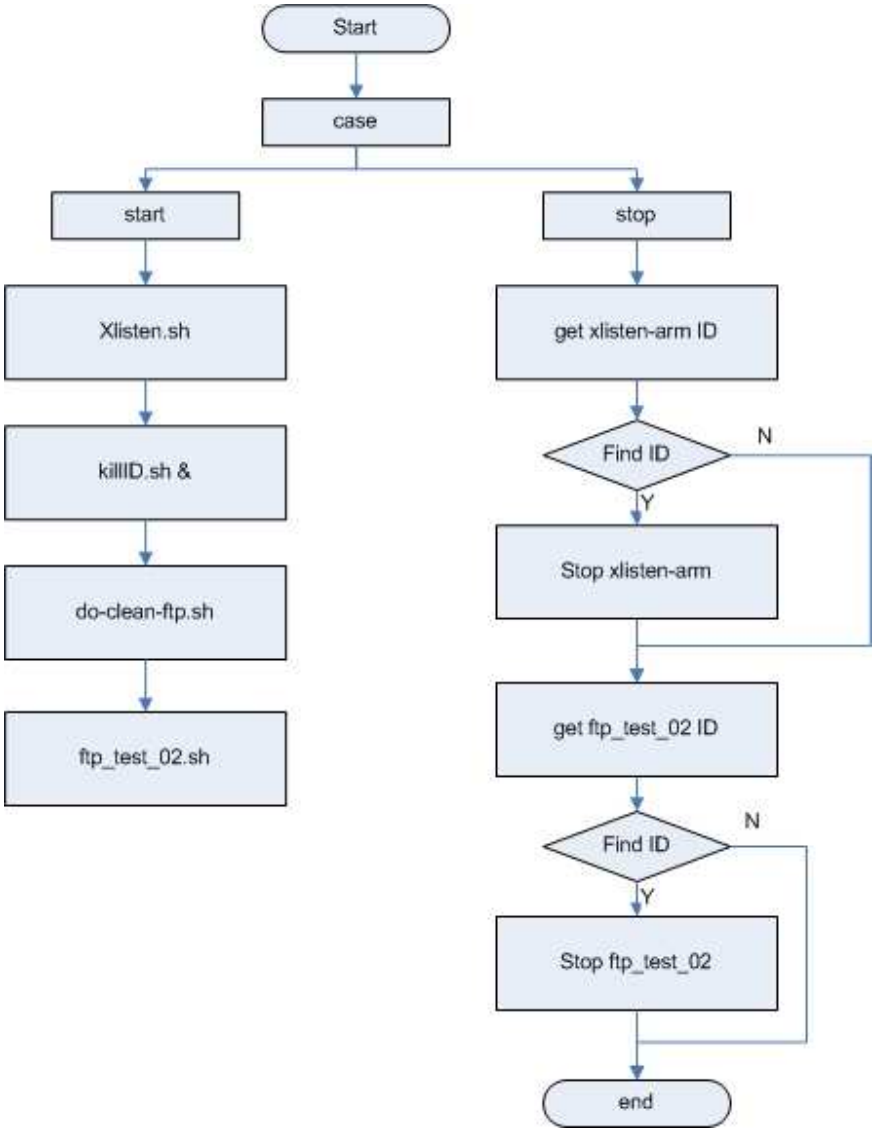


Figure 4.12 Flowchart for starting data collection and FTP program

The data collecting script (Xlisten.sh) actually called a C program (xlisten.c) developed by Crossbow. Based on given input parameters, the "xlisten.c" program listened to assigned serial port and printed out received data in a defined format. In this study, this C program saved all raw data into a text format file on the Stargate. We also used this C program to obtain time information from the Stargate and add timestamps for each received record. The flowchart for the data listening and printing C program is shown in Figure 4.13.

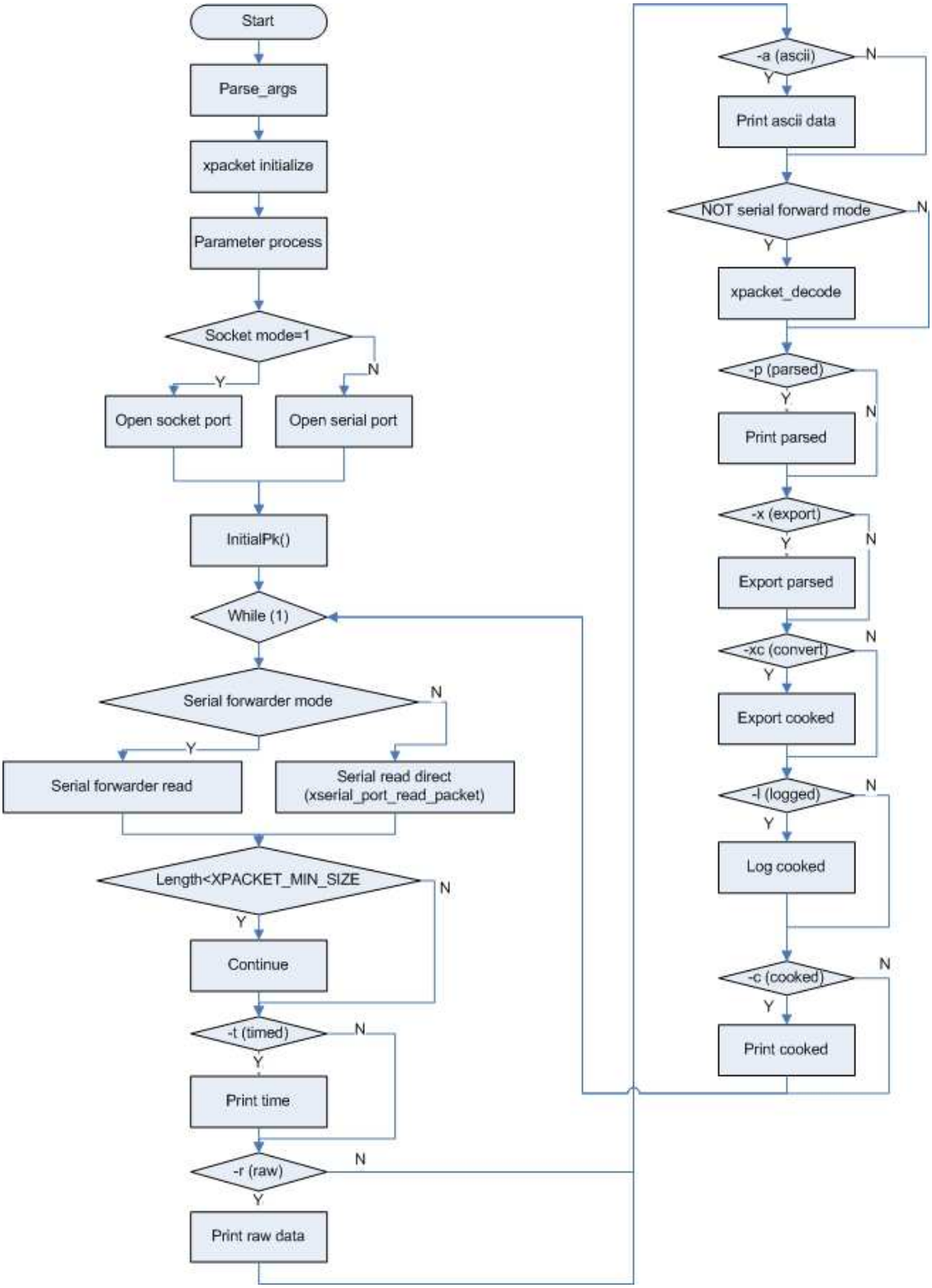


Figure 4.13 Flowchart for the data listening and printing C program

The FTP service was supported by a client-server architecture, which requested both client and server in good operating conditions. In case any part of this architecture failed, the connection could not be maintained. This was also true for wireless communication when the signal was interrupted by other noise sources. In this study, a FTP client was used on the Stargate. It requested connection and sent data to the server computer. Once the FTP client started a connection, it occupied the resources such as memory, network port on the Stargate. This FTP client and connection were closed by the user program when the file transmission task was finished. However, in some situations such as losing transmission signal and noise interference, before being properly closed by the user program, this FTP client remained in the system even when the actual connection was broken. This dead FTP client must be cleaned up so that we can reuse the resources for new FTP connections. A script program (KillID.sh) monitored the FTP client on the Stargate. If a FTP client was found to stay for an extraordinarily longer time than normal, the “KillID.sh” program forced this FTP client to exit and release the resources occupied by it. In this study, it normally needed less than 20 seconds for a FTP client to complete a FTP transmission. If a FTP client stayed more than 2 minutes, it would be cleaned up by the “KillID.sh” program. The flowchart for this script program is shown in Figure 4.14. The source code is listed in the Appendix B.

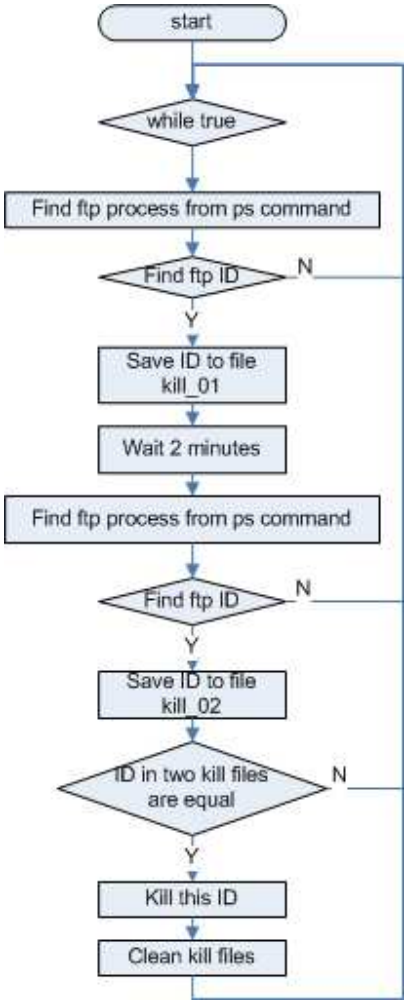


Figure 4.14 Flowchart of the dead FTP clean up program

Before sending data files to the server computer, we used a FTP connection checking script program (do-clean-ftp.sh) to establish the first connection without sending any data. This was a safeguard program to ensure that both the client and server were ready for data communication. The flowchart for FTP connection checking program is shown in Figure 4.15.

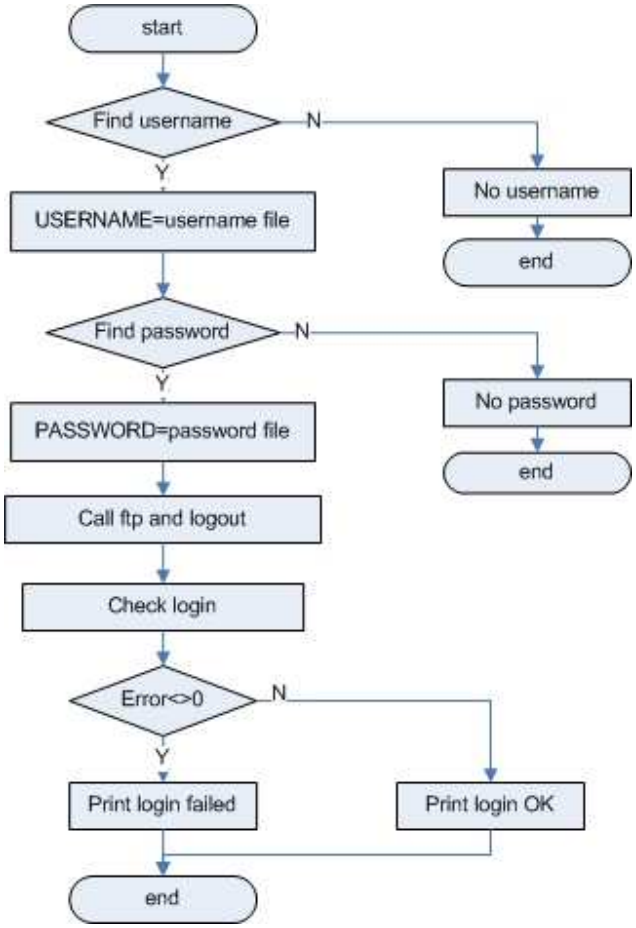


Figure 4.15 Flowchart for FTP connection checking program

After the transmission from the mote(s) to the Stargate was established, the incoming data from the mote were accumulated in a data-log backup file one line after another. The data-log backup file was a file to save all incoming records from the sensor motes on the Stargate. In case of wireless transmission failure, a data backup could be retrieved from it. We used a line reader script program (Readxlisten.sh) to read the line numbers and trace any changes in the line number. Each record had two lines of data: line of timestamp and line of raw data. Hence, if the line number was increased by 2, we knew that there was a newly added record in the file. By checking changes in line number every second, a call FTP script program (ftp_test_02.sh) could call a FTP transfer script program (do-file-xfer-02.sh) when we had new record in the data-log backup file. The flowchart of the call FTP program is shown in Figure 4.16.

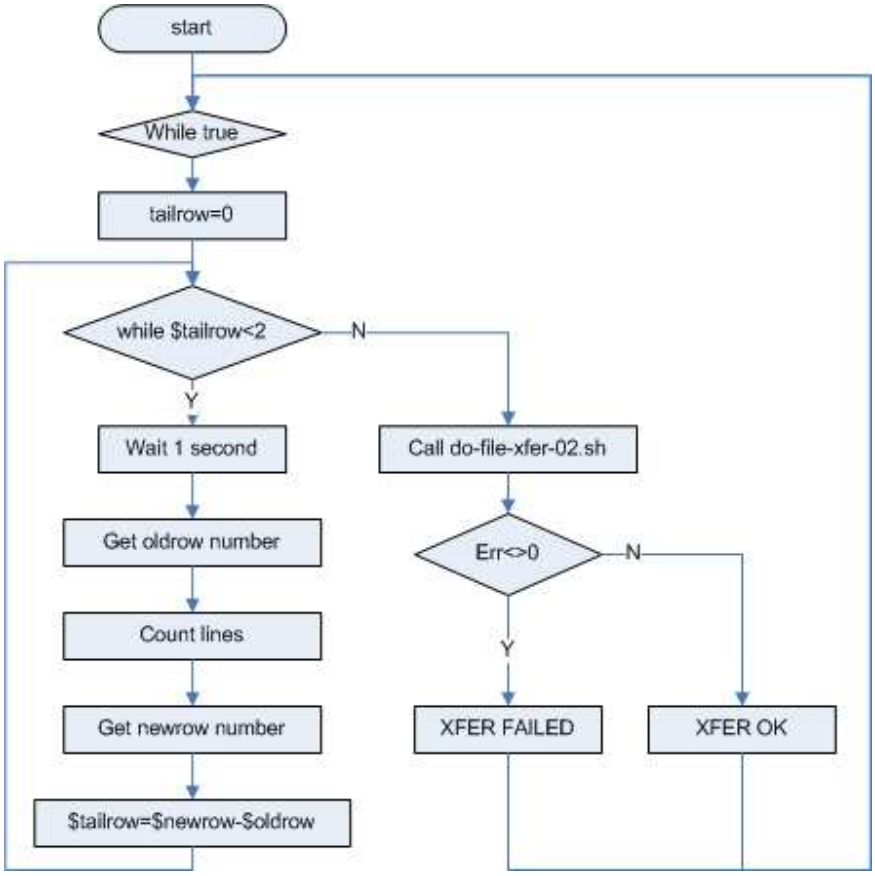


Figure 4.16 Flowchart of call FTP program

Upon a data sending request, the FTP transfer script program (do-file-xfer-02.sh) prepared a data-sending file for transmitting based on the newly generated line number. The data-sending file was a file to temporarily store data. The program then called the FTP command to send the data-sending file to the server computer. If the file was transferred successfully, the program received a three-digit status code, “200”, from the FTP remote server to indicate that the FTP command was successful. It then updated the line number to indicate that those lines were transferred. If the transfer failed, the program checked the FTP connection for three times before restarting the Stargate. The durations between these three checking points were called Waiting Interval (WI). The WI was doubled after each restarting of the Stargate until the length of WI reached 1 hour. The checking algorithm eliminated any unnecessary system restart due to temporary connection problems such as transmission traffic congestion and low transfer rate caused by weak signal. It also reduced the frequency for restarting the Stargate when there was a real connection problem from the GPRS service. In case the FTP connection was lost for

more than 5 hours in a gateway that was communicating with two motes, the accumulated new records could generate a huge data-sending file. When the FTP connection was recovered, it usually took more than 2 minutes to accomplish the file transfer. However, a FTP client executing more than 2 minutes could be ended by the "KillID.sh" program. As a consequence, the Stargate repeatedly failed to send this huge file, and it blocked the normal FTP transmission. To avoid this predicament, the FTP transfer program copied the most recent records from the data-log backup file to the data-sending file, and made the data-sending file size less than 550 k so that it could be sent out within 2 minutes before the FTP client was terminated by the "KillID.sh" program. Figure 4.17 shows the flowchart of the FTP transfer program.

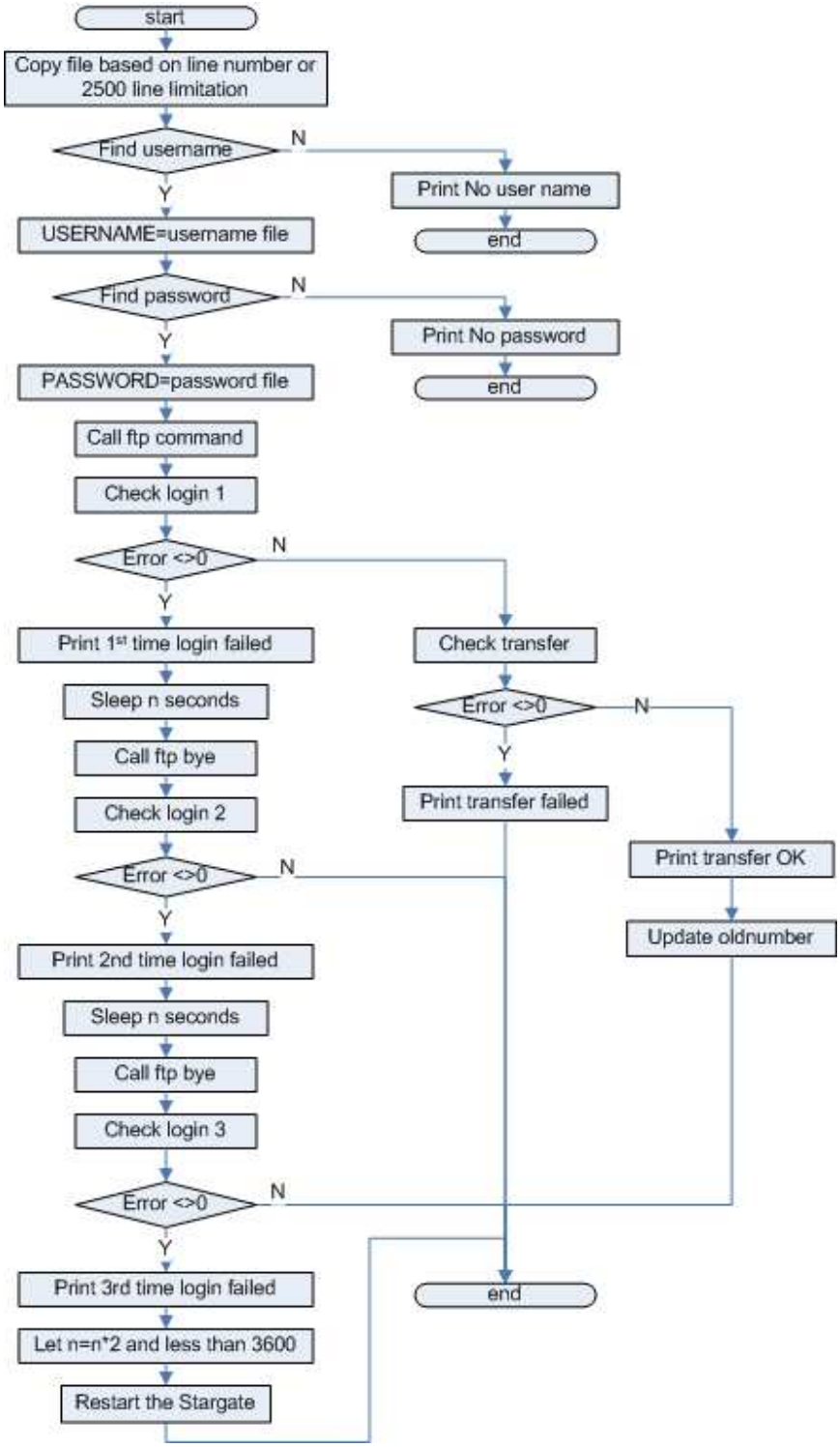


Figure 4.17 Flowchart of the FTP transfer program

The last program, an initial checking script program (check_init.sh) checked if four processes - the data listening and printing C program, the watchdog timer program,

the call FTP program, and the line reader program - were running. If any of these processes was not working, it rebooted the Stargate. This initial checking program only executed for two minutes from the time the Stargate was started or restarted. The source codes for above program can be found in the Appendix B.

4.2.5.3 The server computer program

The server computer received sensor data, and stored them into a Microsoft Access database. A backbone java program (backbone.java) running on the server computer continuously monitored the received data file. The program obtained the latest date and time from the database and saved them for a data render C program (bkj_render.c) to use later. It then started the data render C program which detected and prepared data when new records came in. As the new data was ready, the backbone java program inserted them into the database, and then used stored procedures to convert the raw data into user-readable values in the database. Based on the new data's timestamp in the database, the backbone java program gathered the data of the last 24 hours and called a graphic generating C program (bkj_gd.c) to draw images for Internet presentation. The flowchart of the backbone java program is shown in Figure 4.18.

The data render C program, interacted with the backbone java program, was used to read received sensor raw data, make a copy of them, calculate the SSC predication from one record for each mote and save them to a text file (render.txt) for Internet display. This program also told the backbone java program when a data was ready for insert into the database based on its timestamp, so that no duplicate records were entered. This maintained the consistency of the database. Figure 4.19 shows the flowchart of the data render C program.

The graphic generating C program used a GD library to draw PNG format pictures for Internet display. GD is an open source code library for the dynamic creation of images by programmers. GD creates PNG, JPEG and GIF images, among other formats. The most common applications of GD involve web site development (Boutell, 2007). The flowchart of the graphic generating C program is shown in Figure 4.20.

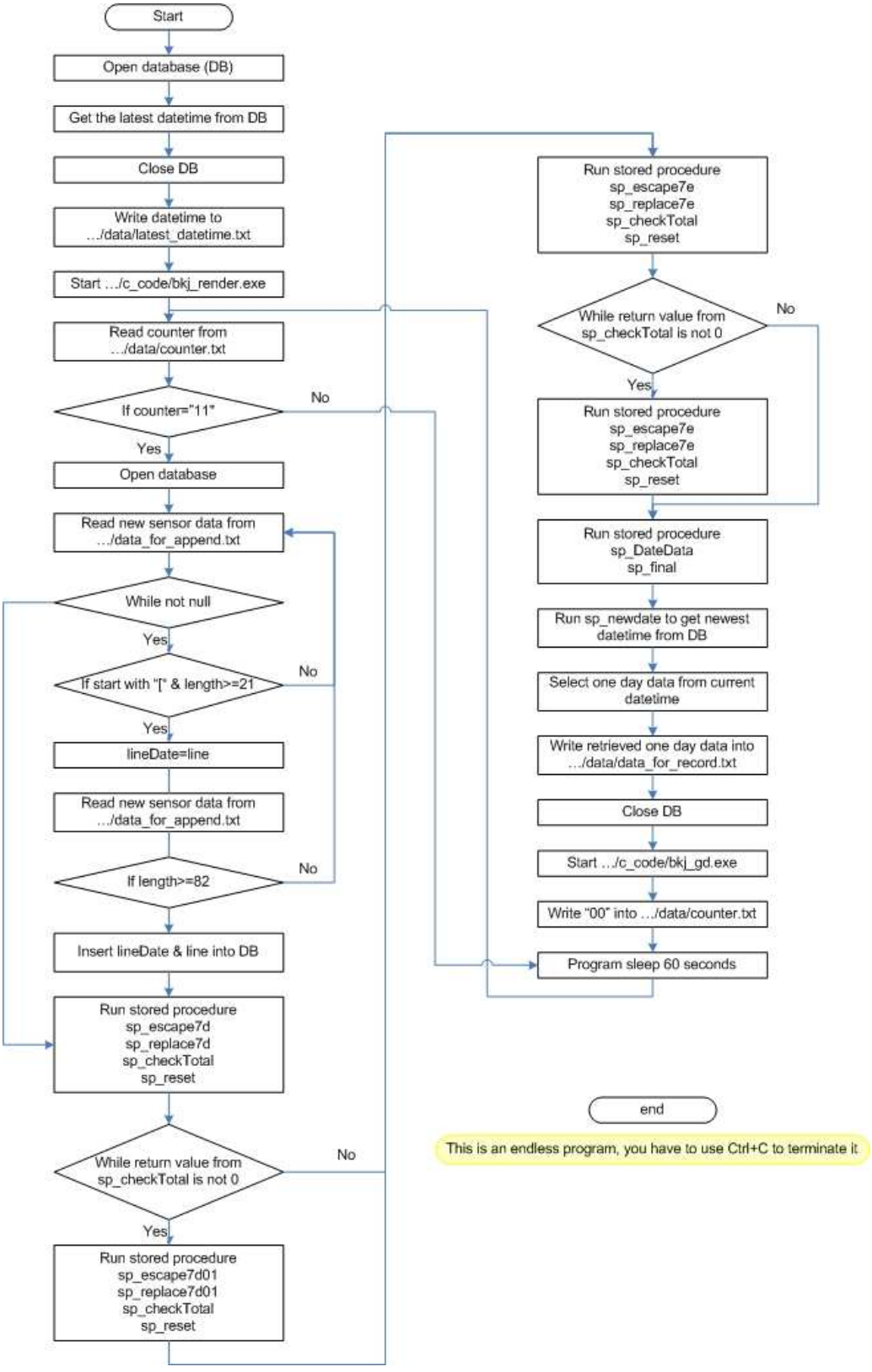


Figure 4.18 Flowchart of the backbone java program

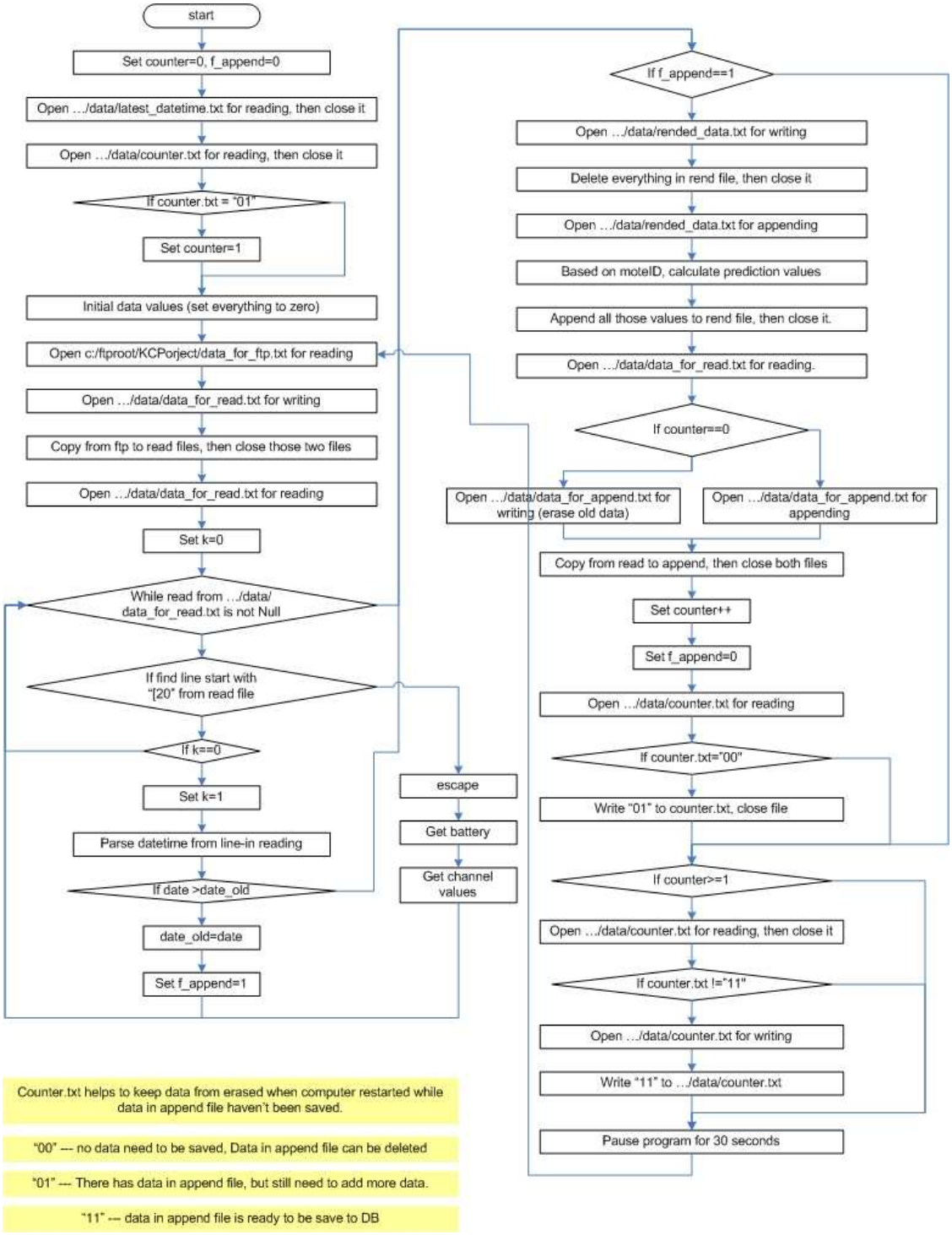


Figure 4.19 Flowchart for the data render C program

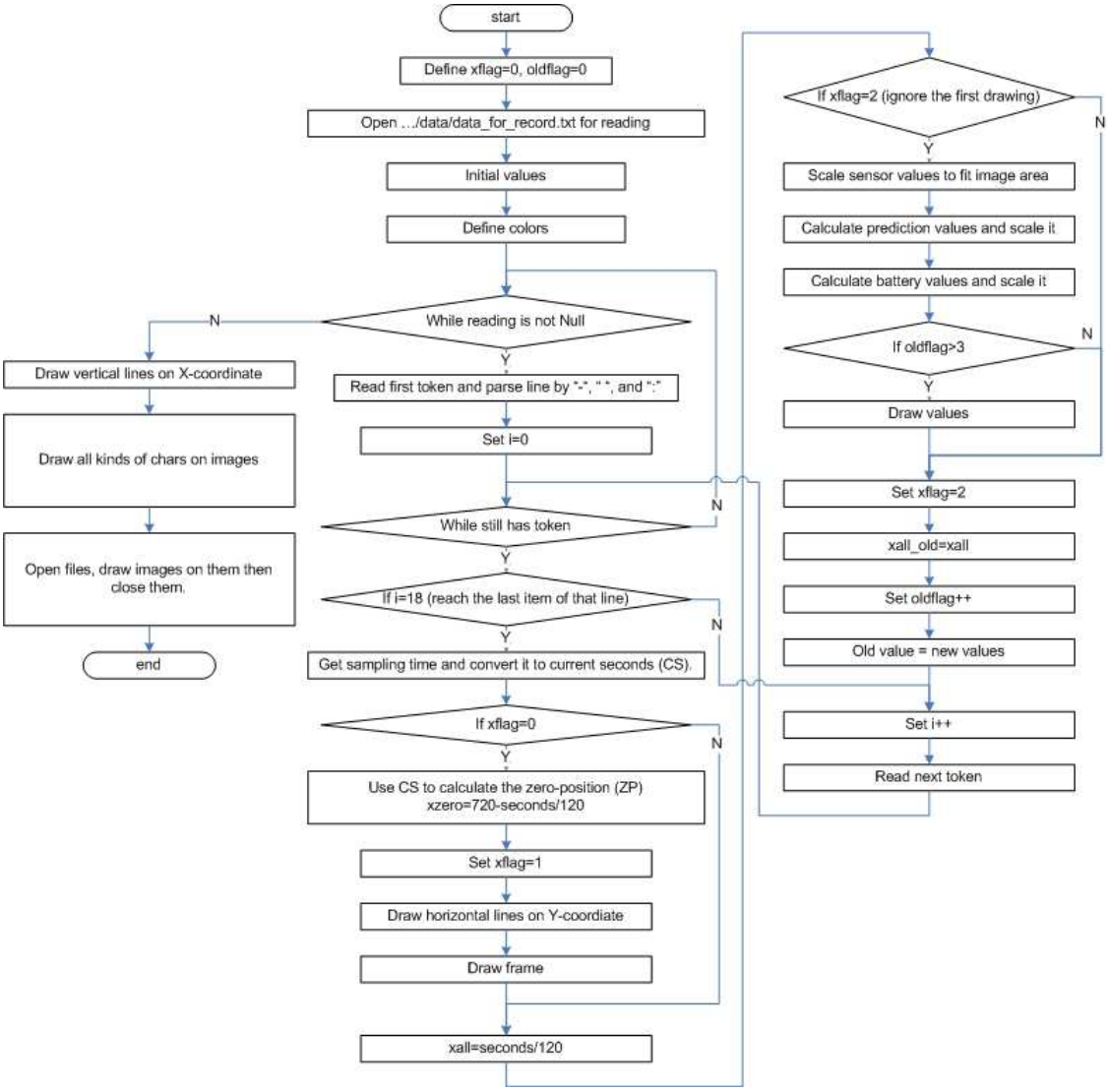


Figure 4.20 Flowchart of the graphic generating C program

Stored procedures were used in the Microsoft Access database to manipulate data. Stored procedure is a subroutine program written in SQL, Java, and C etc. It is saved in the database. The stored procedure can be called by other applications and run directly within the database. In this study, all the stored procedures were written in SQL language. By using these stored procedures, raw data were validated, timestamp were added to each record, and number escapes were detected and changed back to their original values in the "tblResults" table. Based on the record layout shown in Figure 4.10, raw data were parsed and inserted into the final results table (tblResults). Finally, for Internet display purpose, the data of the latest 24 hours were retrieved from the database.

4.2.6 Deployment and installation

4.2.6.1 Fort Riley test site

A preliminary experiment was conducted at Fort Riley, Kansas, in August 2006. Figure 4.21 shows the components of the remote sensor system. A sensor was connected to a water pipe. A water pump propelled the water with soil sediment from a nearby water tank into the pipe. The sensor measured the sediment concentration in the running water. An MDA300 data acquisition board attached to a MICA2 mote collected data from the sensor, and a Stargate transferred the data via the AirCard 750. A laptop was used for debugging and temporary on-site monitoring. A LabView program was used to display real-time data on the laptop.

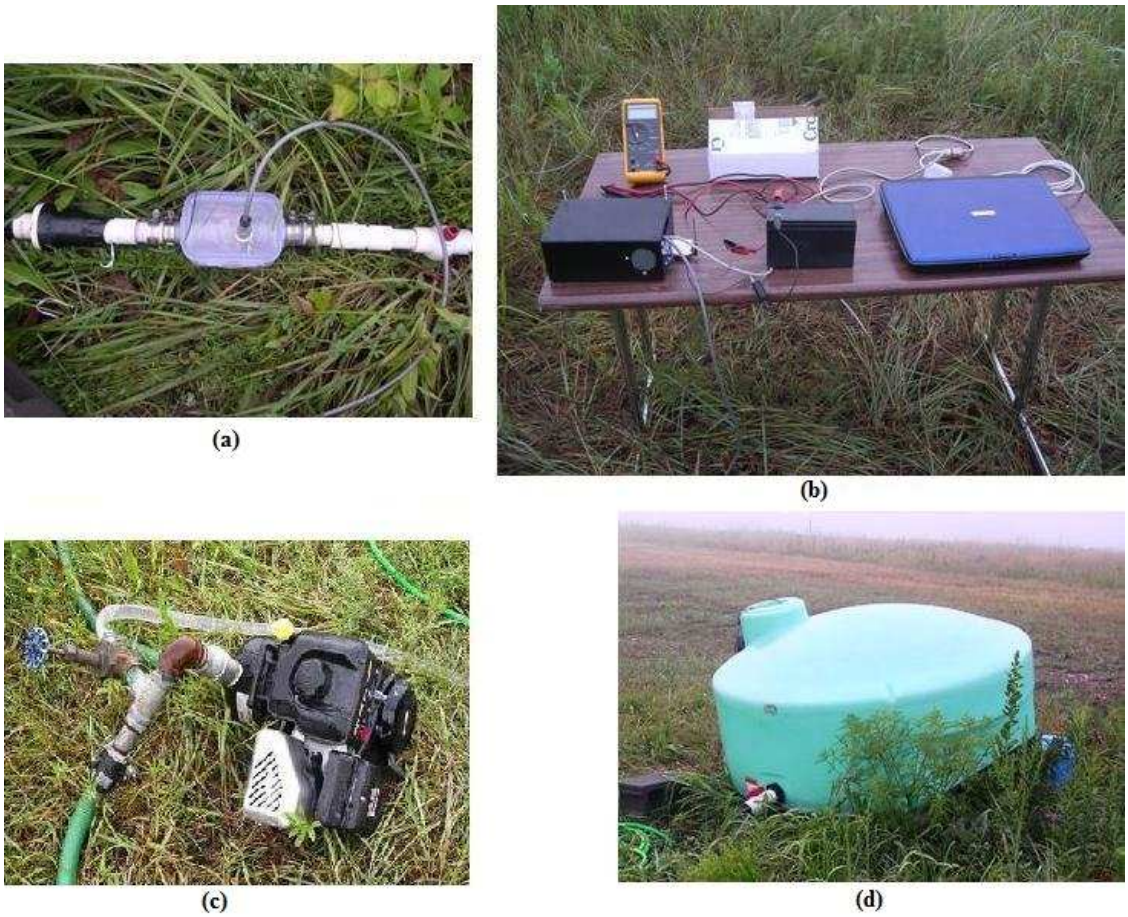


Figure 4.21 Components of the field test at Fort Riley, Kansas: (a) Sensor, (b) Stargate gateway, (c) Water pump, (d) Water tank

We established signal transmission and received data from both the in-field remote computer and the indoor server computer. However, at that time, we encountered several problems—the data transfer signal was sometimes weak, the power source was overheated, and the Stargate occasionally stopped responding. These problems reduced the system reliability, which was detrimental for an unattended remote system.

Due to the weakness of the GPRS data transfer signal in rural areas, the AirCard 750 on the Stargate can lose its signal and disconnect from the system. The AirCard 750 used in this study cannot restore the connection by itself. To solve this problem, we mounted a range extended antenna (YSC-RE1905U-SNP, Sharper Concepts Inc., Boca Raton, FL) on a 1 meter pole to improve the signal power level. We also added a checkpoint in the Stargate data transmitting program to determine whether the FTP connection was still alive. If the connection was found not maintained, the program would recheck the FTP link two more times before it restarted the AirCard to restore the connection.

We solved the overheating problem by using a power management algorithm in the shell script program on the Stargate. Basically, we closed the transmission channel on the AirCard whenever possible. As a result, the power consumption was reduced significantly. Moreover, by using a switching voltage regulator, we significantly reduced the heating problem and prolonged the battery life.

The Stargate was a single-board computer. Like most computers, it can stop responding after operating for an extended time period. This freeze of the Stargate might be due to errors in a third party driver we used for the AirCard. However, Crossbow did not offer a driver for AirCard 750 on the Stargate. To improve the reliability of the system, we started a watchdog timer (WDT). It would restart the Stargate after the processor stopped responding for a certain period of time.

During the Stargate's restarting period, the sensor data were saved on the remote mote's 512-Kb flash memory, which, in this study, can save up to 40 hours of data without sending them out. Therefore, restarting the Stargate would not cause data losses.

4.2.6.2 Little Kitten test site

We conducted the second experiment in Little Kitten Creek (Manhattan, Kansas) from August 2006 to March 2007. System block diagram are illustrated in Figure 4.22. A sensor was located in the creek, and the wireless communication system was located on the bank. A 12 meter long cable connected the signal conditioner on the bank to the sensor. The sensor was protected by a waterproof enclosure. A protective fence was also built to protect the sensor and cable from damage by floating logs and other debris.

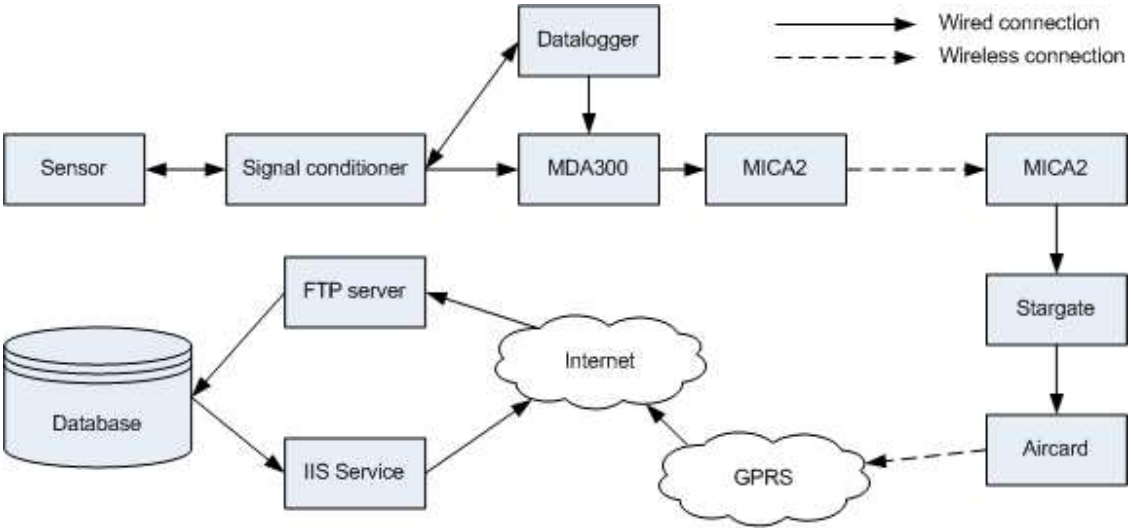


Figure 4.22 Block diagram of a two-tier WSN system at the Little Kitten Creek site in Manhattan, Kansas.

We used a CR10 datalogger (CR10, Campbell Scientific Inc., Logan, UT) to control the sensor scanning. Sensor data were stored in the datalogger every 10 seconds. Saved data were used later to verify the accuracy of wireless data transmission. The datalogger also sent a synchronization signal to the MDA300 board to initiate its onboard sampling every 30 seconds. Thus, the sampling rate for the MDA300 was set three times lower than that of the CR10 datalogger.

Data were sent immediately from the MICA2 mote to the Stargate and transferred to an indoor server computer through GPRS using an FTP service. A 10-12 second transmission latency due to FTP connection and data transfer was observed. After the resuming algorithm with a checkpoint in the mote program and a WDT for the Stargate was implemented, data transmission became more reliable and robust. The system sent data continuously for 15 days and stopped due to a battery failure. The system was

powered by a 12 V car battery. Voltage levels at the battery and each MICA2 mote in the system were monitored in real time via the Internet.

4.2.6.3 Mission City test site

We applied solar panels in our latest two-tier architecture experiment. The experiment was set up in Mission City, Kansas. On August 17, 2007, we installed two sensors and a two-tier wireless sensor network to measure sediment concentrations in storm drainage water. Two MICAZ sensor stations collected and sent data to a central station. As the second infrastructure tier, the central station stored data and communicated with a cell phone system. By using a FTP protocol, we retrieved data at our backbone computer system. We did not use datalogger in this experiment. Backup data were stored on a CF memory card with a capacity of storing one year's worth of data. Real-time monitoring data were published on the Internet. Figure 4.23 shows the deployment map of the sensor nodes and central station.

Figure 4.23 Map of the wireless network deployed in Mission, Kansas (Map from Google Imagery, Digital Globe, Sanborn, Map Data)

The power supply was enhanced by installing a 15 W solar panel (Northern Tool Equipment) for the central station and 5 W solar panels for each sensor station. Reprogramming the software to further reduce the activation time on remote devices also

improved power conservation. We tested the system from Jan. 16 to March. 19, 2008, and observed that the power supply was sufficient under normal weather conditions.

Radio-hostile environments were another challenge we experienced in this study. Signal loss due to absorption by nearby water and lawns greatly reduced the transmission range. Trees and vehicles also impeded signals. Moreover, interference caused by obstacles near the path of transmission, usually referred to as a “Fresnel Zone” (Rappaport, 2007) problem, also hindered the stableness of data transmission. The results were either no signal or a severe packet loss.

We achieved more than 100 meters transmission by using MICA2 mote with 900 MHz radiofrequency under laboratory conditions. However, for the field test in Mission City, we could not attain this distance. We tested three carrier frequencies - 433 MHz, 900 MHz, and 2.4 GHz - under the same environmental conditions. The effective radio transmission ranges these frequencies were found to be approximately 30 m, 10 m, and 20 m, respectively. The distance between one of the sensor stations to the central station was 96 meters. Therefore, changing carrier frequency did not solve the problem. Eventually, we decided to use a 2.4 GHz MICAZ mote which, compared with motes with other frequencies, has a higher data rate, lower power consumption, cheaper external antenna, and better anti-interference ability.

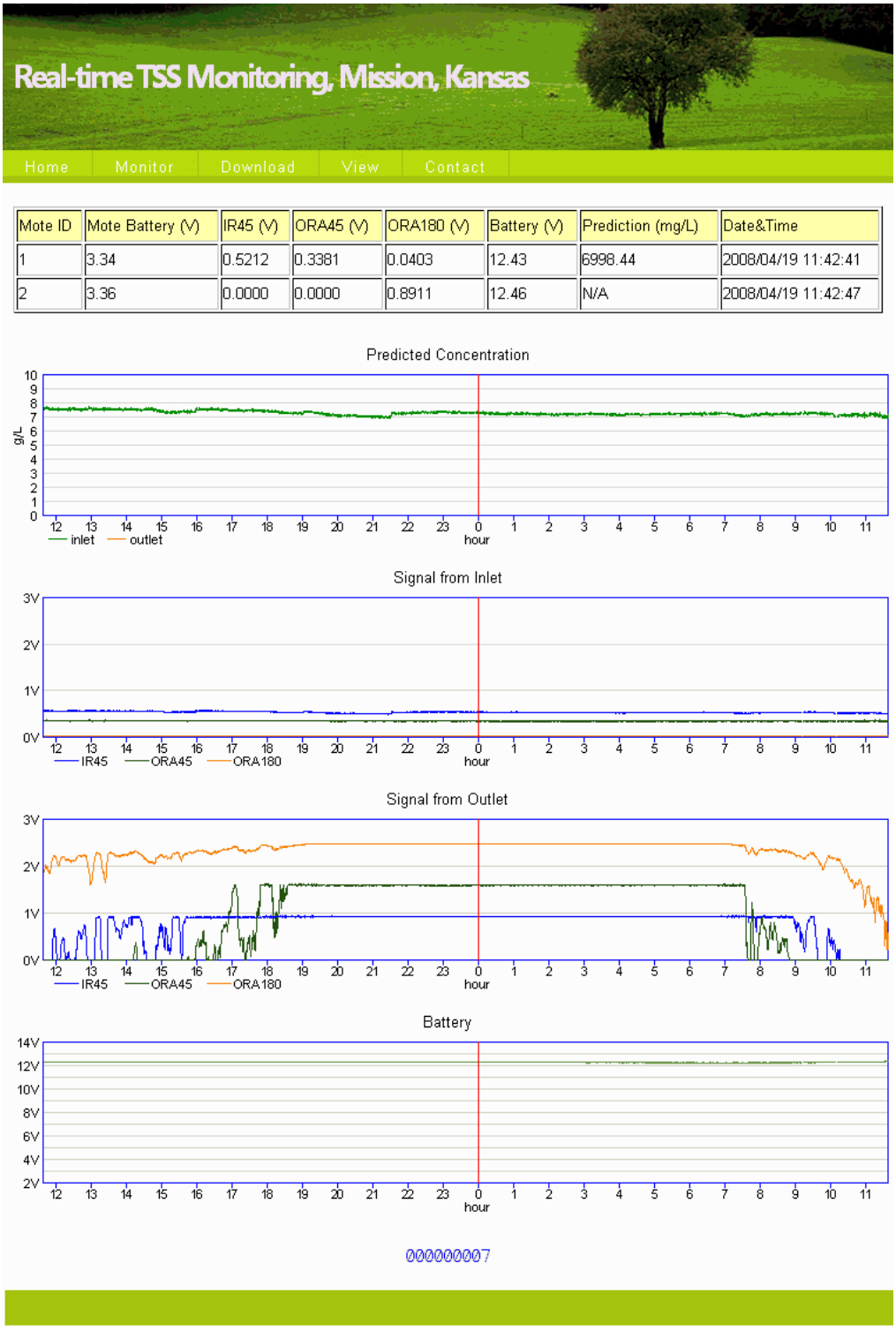
We also tested multi-hopping relays by adding additional motes between the sensor stations and the central station. After the field test, this multi-hopping relay algorithm was proved to be feasible. However, as we added more components, such as motes, solar panels and car batteries, into the system, the system became more complicated, and the implementation cost increased.

Finally, we solved the transmission range problem by using both omnidirectional and Yagi directional antennas. An omnidirectional antenna (HyperLink Technologies Inc.) with 8.5 dBi gain was used at the central station. Each sensor station has a Yagi directional antenna (HyperLink Technologies Inc.) with gain of 14.5 dBi. Setting antennas at a height of 3 meters (Figure 4.24) minimized packet loss. For more than two months of continuously observation, the average packet loss rate was 3.1%.

Figure 4.24 Two sensors and a remote system in Mission City, Kansas

4.3 Results and discussion

After setting up the system in Mission city, Kansas, we transferred sensor data to a database on an indoor server computer. These data can be observed on the Internet and are updated every 1 minute. One example of the real-time monitoring interface on the Internet is shown in Figure 4.25. The table shown on the top in Figure 4.25 gives readings from two sensors and sediment concentrations calculated from these readings. Signal levels from each sensor were shown in the following graphs in different colors. Battery voltage was also displayed to help monitor the power supply condition.



Mote ID	Mote Battery (V)	IR45 (V)	ORA45 (V)	ORA180 (V)	Battery (V)	Prediction (mg/L)	Date&Time
1	3.34	0.5212	0.3381	0.0403	12.43	6998.44	2008/04/19 11:42:41
2	3.36	0.0000	0.0000	0.8911	12.46	N/A	2008/04/19 11:42:47

Figure 4.25 A screenshot of the real-time monitoring interface at the Website.

After collecting data from Jan. 24 to Feb. 15, 2007, we compared the wirelessly transferred data with data stored in the datalogger. The results are illustrated in Figure 4.26 and in Figure 4.27.

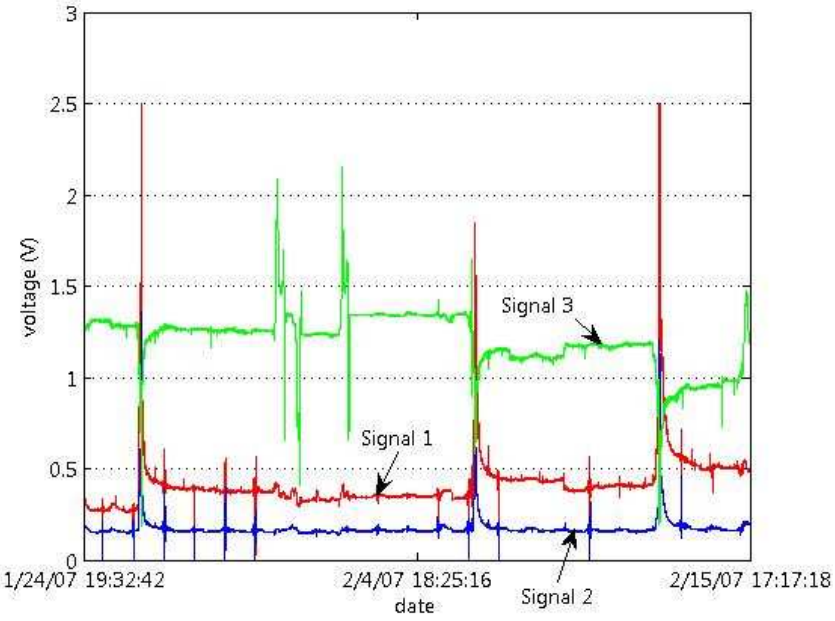


Figure 4.26 Wirelessly transferred data extracted from the database stored in the server computer

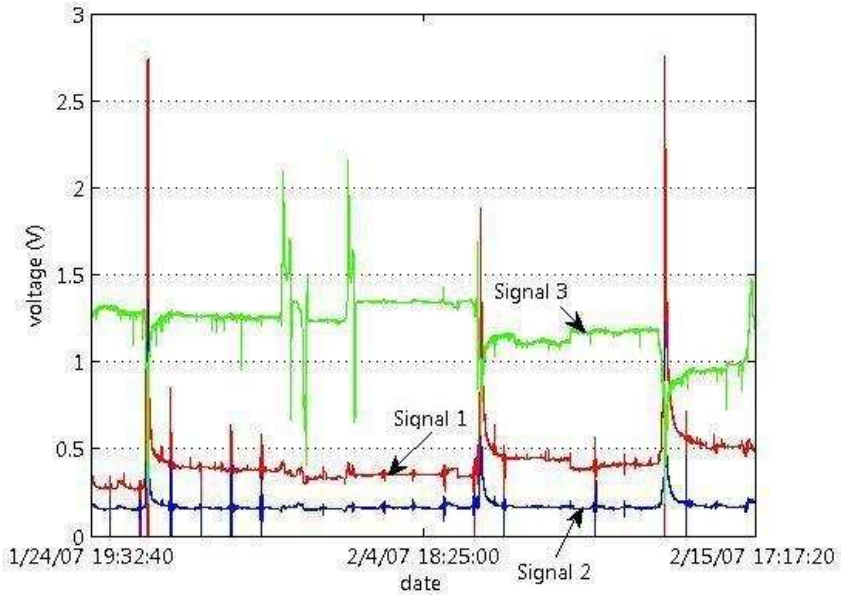


Figure 4.27 Sensor data logged by CR10

The analog to digital converter (ADC) on the data-acquisition board mounted on the wireless mote and the ADC on the datalogger were different. The difference was removed by establishing a regression model of the datalogger data against the wirelessly received data. Using this model, data received from the WSN were first converted to "equivalent" datalogger data and then compare with the actual datalogger data. The result is shown in Table 4.2.

Table 4.2 Percentages of outliers in transmitted signals

	Signal 1	Signal 2	Signal 3
Mean of absolute difference between signals (mV)	4.2	1.8	1.8
Confidence interval (2σ) (mV)	16.3	9.5	5.7
Number of outliers	673	527	1959
% of outliers in data	1.09	0.85	3.16

In Table 4.2, the "outliers" were defined as the wirelessly transmitted signals that were outside the confidence interval of the datalogger-recorded signals. The confidence interval was defined by $\pm 2\sigma$ from the datalogger-recorded values, where σ was the standard deviation of the difference between the wirelessly transmitted and datalogger-recorded signals, which was assumed to have a normal distribution. As observed in Table 4.2, signal 3 presented the worst case, where 3.16% of the wirelessly transmitted data were considered as the "outliers". Further observation of these outliers revealed that, most of them occurred around 10:40 AM each day, which was the time when the ambient light condition for the optical sensor experienced a sharp change from shadow to direct sun light. This transition took place within a very short period of time. Because the sampling times of the mote and the datalogger were not completely synchronized, a small difference in sampling time may yield significantly different signal readings within this transition period. Thus, the outliers may not necessarily represent wireless transmission errors.

From the experiment in Mission City, Kansas, we recorded wireless data from Jan. 16, 2008 to Aug. 19, 2008. In order to study the data loss during wireless signal transmission, we defined the packet loss rate as follows:

$$\text{Packet Loss Rate } L_R = \left(\frac{N_T - N_S}{N_T} \right) \times 100\% \quad \textbf{(4.2)}$$

where N_S is the actual number of data packets received in database during a time period.

N_T is the number of data packets actually transmitted during the same time period, and

$$N_T = \left\lceil \frac{T_D}{T_S} \right\rceil \quad \textbf{(4.3)}$$

where $\lceil x \rceil$ is the ceiling function that returns the smallest integer not less than x,

T_D is the duration of the observation period in seconds,

T_S is the sampling interval in seconds.

From March to May, 2008, we received 197,650 data records from the wireless system. The actual data packet loss rate was 3.1%.

A severe ice storm occurred in mid-December of 2007. The equipment in Mission City, Kansas, survived without damage and data loss. However, due to a power outage, we could not retrieve data from the backbone database on the indoor computer in real-time. After the power was resumed, the system was able to recover the data and completely return to its normal operation.

Long-range wireless communications using microwave link or satellite were expensive and difficult for deployment. In this study, we explored the use of a commercial cell phone service to achieve low-cost, long-range transmission. The cost for the two-sensor system is summarized in Table 4.3.

Table 4.3 Hardware cost for two sensors in a real-time sediment runoff monitoring system

Components	Price
Wireless devices	$1,300.00
Antenna devices	$350.00
Power supply devices	$650.00
Cellular service	$620.00
Total	$2,920.00

The sensor we used was at a much lower cost than similar, commercial sensors. Once the system was installed, it saved a great amount of maintenance cost that was necessary when human operators were sent to the experimental sites to take water samples manually. With the WSN, data can be obtained in near real time, which would allow researchers to study the water samples in a timely manner and, in some cases, respond to emergencies without delay.

CHAPTER 5 - THREE-TIER WIRELESS SENSOR NETWORK

As the area of coverage increased, two-tier WSN was not an appropriate solution for transmitting the sensor data. Considering the situation where many sensor nodes were spread over a large observation area, we needed to find a mid-range wireless communication module that connected the local sensor nodes with the central station. Therefore, designing a three-tier WSN was essential for the monitoring system's further development.

A typical three-tier WSN included sensor nodes and a gateway, which made up the first tier of wireless communication. The transmission distance within the first tier was determined by the motes, which were responsible for short-range radio transmission. The wireless communication between gateways and the central station represented the second tier. The transmission distance may vary from a few hundred meters up to several kilometers, depending on the radio devices used. The third tier was the wireless communication between the central station and a server computer via a long distance data transmission service, such as a commercial cellular service and a Meteor Burst Communication (MBC) service.

5.1 Network Architecture

As the name implied, the three-tier WSN included three levels of networking: the local wireless sensor network (LWSN), the mid-range sensor network (MRSN) and the long-range cellular network (LRCN).

The LWSN was in charge of transmitting data from sensors to a gateway station. On this tier, the area the wireless transmission had to cover was usually near a stream or under a bridge, surrounded by trees and other vegetations. The vegetations and water were in general hostile to wireless transmission because they often absorb, refract, or reflect the signals. Moreover, commercial cellular coverage in these areas were generally

poor or nonexistent, short range wireless devices (up to 100 m) were needed to relay sensor signals to a gateway station, where signals could be further relayed.

The MRSN relayed the data, through a moderately long distance (up to 16 km), from the signal-unfriendly sensor sites to a location with a satisfactory commercial cellular coverage. Because of the longer transmission range, the MRSN allowed multiple LWSNs within larger areas to send data to the same central station, where they shared a single cellular service to transmit the data to the Internet. Repeater station could also be added to the MRSN to further enlarge its coverage area.

The last tier in the three-tier WSN was the LRCN. It used a commercial mobile wireless data service to further transmit data to the database server through the Internet. A “Web-GIS” system developed in this project gave access to the database via the Internet. A block diagram for the three-tier WSN system is shown in Figure 5.1.

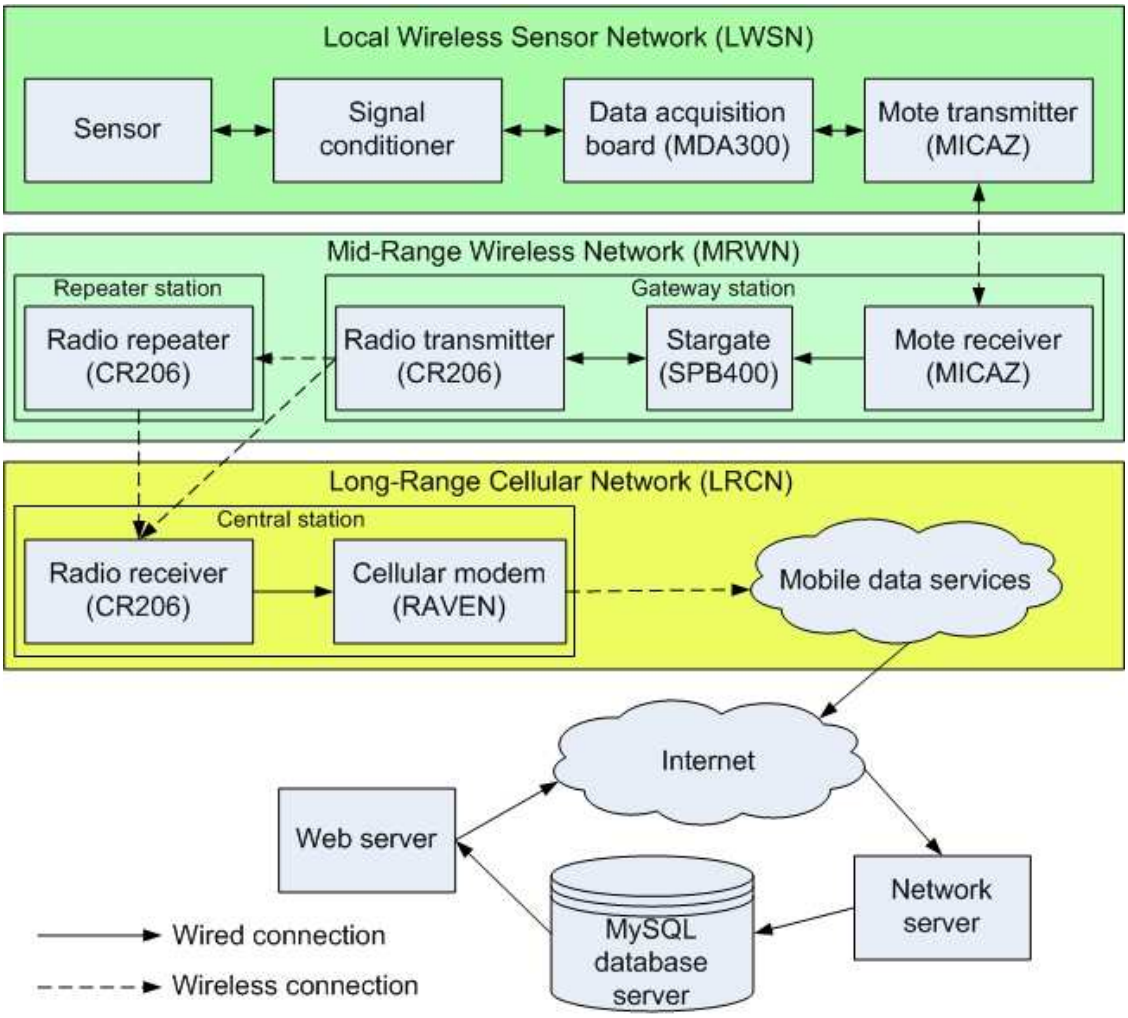


Figure 5.1 Block diagram for the three-tier WSN monitoring system

5.2 Methodology

5.2.1 Radio propagation model

Before further discussion of the three-tier WSN, we need to investigate a few propaganda theories to better understand features of various wireless communication systems. This would also help us in planning the topology of the network system.

Performance of wireless communication has some elementary boundaries imposed by its radio path. The transmission path between the transmitter and the receiver varies from simple line-of-sight to the one that is severely blocked by objects such as buildings, mountains, and foliage. Unlike most wired paths, wireless paths are usually

affected by variations in the environment surrounding the path, which often add complexity to the analysis.

Using propagation models to predict the average received signal strength at a given distance from the transmitter is a traditional mechanism for electromagnetic wave propagation analysis. One of the fundamental models is a free space propagation model. This model is used to predict received signal strength when the transmitter and the receiver have a clear, unobstructed, line-of-sight path between them. The free space equation (Rappaport, 2007) is shown below:

$$P_r(d) = \frac{P_t G_t G_r \lambda^2}{(4\pi)^2 d^2 L} \qquad \textbf{(5.1)}$$

where P_r = the received power (W)

P_t = the transmitted power (W)

G_r = the receiver antenna gain

G_t = the transmitter antenna gain

d = the transmitter – receiver separation distance (m)

L = the system loss factor not related to propagation ($L \geq 1$)

λ = the wavelength of the carrier signal (m)

From equation 5.1, we found that the average received signal power decreased when the distance between the transmitter and the receiver increased. Therefore, in order to help extend the system coverage, we could either reduce the transmission distance whenever possible or install repeater stations. Based on equation 5.1, a lower carrier frequency, which corresponded to a longer carrier wavelength λ, would yield a larger received power P_r. Therefore, among carrier frequencies of 900 MHz and 2.4 GHz, 900 MHz was selected for mid-range signal transmission. Also from equation 5.1, the antenna gains for both the transmitter and the receiver contributed to the enhancement of the received signal strength. Thus, we added several types of high-gain antennas in the system to improve the system. Obviously, a higher transmission power in the transmitter would result in higher received signal strength in the receiver. However, in order to reduce the power consumptions and avoid interferences caused by our system to other radio signals in the area, we limited our transmission power in accordance with the

Federal Communication Commission (FCC)'s regulation (FCC, 2011). The signal powers of all the radio devices we used are listed in Table 5.1 below:

Table 5.1 Signal power of all radio devices in the system

Radio devices	Frequency band	RF power
Mote (MICAZ)	2400 MHz to 2483.5 MHz	Maximum 1 mW
Datalogger (CR206)	915 MHz	0.9 W (Transmit), 0.24 W (Receive), 0.036 W (Idle)
Modem (CDMA for Verizon)	800 MHz	3.24 W (Transmit), 2.868 W (Receive), 1.248 W (Idle)
Modem (Quad-Band GPRS/EDGE for T-Mobile or AT&T)	850/900/1800/1900 MHz	5.4 W (Transmit), 4.2 W (Receive), 1.248 W (Idle)

In reality, the radio transmission often takes place over an irregular terrain. The terrain of an interested area needs to be taken into account for estimating the signal strength. The presence of trees, buildings, and other obstacles also must be taken into account.

In study of the radio transmission, three basic propagation mechanisms must be evaluated. They are reflection, diffraction and scattering (Rappaport, 2007). Reflection happens when a signal is imposed onto an object which has a very large dimension compared to the signal wavelength. Ground surfaces, buildings and walls are typical objects for reflection. Reflections on the radio wave path should be avoid or minimized by carefully selecting the location of each radio station.

Diffraction occurs when the radio path between the transmitter and the receiver is obstructed by a surface that has sharp edges (Rappaport, 2007). When the line-of-sight condition is not met, diffraction may actually increase the signal strength behind the obstacle. The Fresnel zone (Figure 5.2) is an example of signal diffraction.

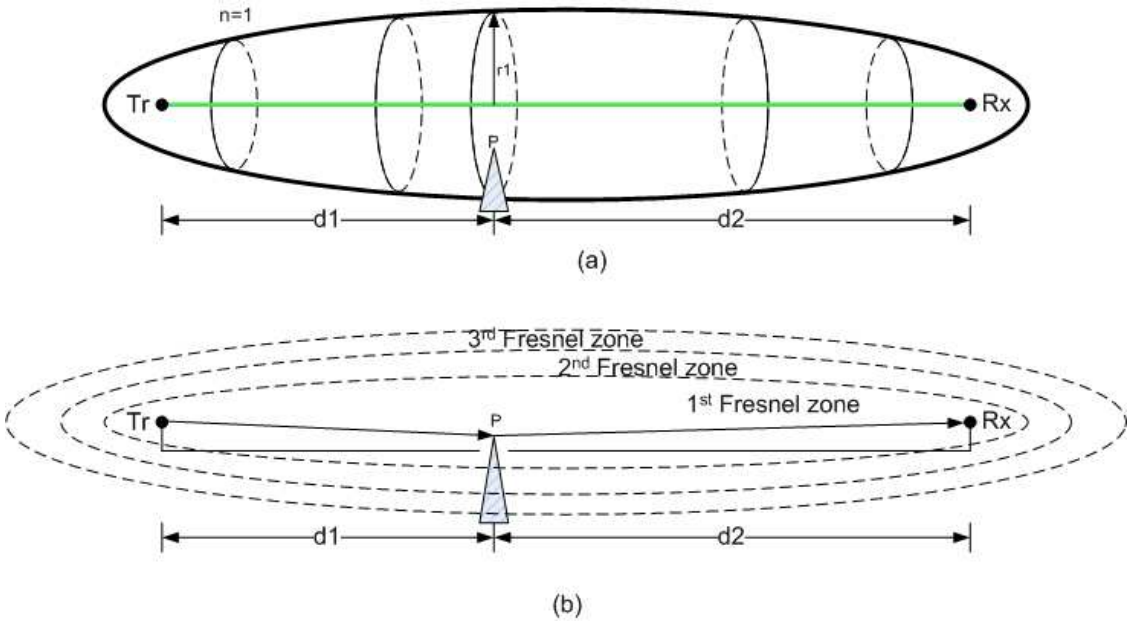


Figure 5.2 The boundaries of Fresnel zones: (a) The 1st Fresnel zone (b) Illustration of 1st to 3rd Fresnel zones.

The Fresnel zone is defined as successive regions where the propagation signal waves caused by diffraction have a path length $n\lambda/2$ greater than the total path length for a line-of-sight path. The radius of the nth Fresnel zone circle (see the circle in Figure 5.2 (a) when n=1) can be calculated as:

$$r_n = \sqrt{\frac{n\lambda d_1 d_2}{d_1 + d_2}} \qquad \textbf{(5.2)}$$

where r_n = the radius of the nth Fresnel zone circle (m)

n = an integer starts from 1 to infinity

λ = the signal wavelength (m)

d_1 = the distance from the transmitter to obstacle (m)

d_2 = the distance from the receiver to obstacle (m)

If an obstacle moves between the transmitter and the receiver, we get a serial of circles which generate many ellipsoids. For case of n = 1, 2, 3 …, we call these ellipsoids as the first Fresnel zone, the second Fresnel zone, the third Fresnel zone and so on. The Fresnel zones are in elliptical shape with the transmitter and the receiver antenna at their foci.

Usually, if the obstacles do not block most volume of the first Fresnel zone, the signal loss could be minimized. For a rule of thumb, if we could maintain 60% volume of the first Fresnel zone clear, we should have a good communication path. For this reason, we built antenna towers with different heights to create a larger clearance volume in the first Fresnel zone.

Scattering occurs when there are a large number of objects with dimensions smaller than the wavelength of the carrier signal. Objects like foliage, street signs and lamp posts are typical objects that cause scattering (Rappaport, 2007). Higher carrier frequency produces shorter wavelength. For carrier signals of higher than 2400 MHz, which correspond to wavelengths of shorter than 0.125 m, it has fewer objects with dimensions smaller than its wavelength when compared with carrier frequency of 900 MHz. In the areas for mote signal transmission, we chose 2.4 GHz over 900 MHz as the carrier frequency so that the scattering effect may be reduced.

5.2.2 System components

5.2.2.1 Sensor nodes

In this study, a fully equipped sediment/velocity sensor node contained four modules. They were sensory module, sensor cleaning module, control and communication module, and power supply module. The sensory module included an optical sensor for SSC and velocity measurement, a dye bottle or canister for velocity measurement, a thermocouple for temperature measurement, a rain gauge for precipitation measurement, and a metal cover to protect the light sensor from interference caused by the ambient light. The sensor cleaning module included an air-blast or an ultrasonic device for automatic cleaning of the optical lenses. The control and communication module included a mote, a data acquisition board, a printed circuit board (PCB) board and a 2.4 GHz Yagi directional antenna. This module set the sampling rate of the sensor and the interval for sensor cleaning. It also sent the measurement data wirelessly to a mote mounted on the gateway station. These devices required considerable electrical power to operate, which was provided by the power supply module. The module included the solar panels and one or two deep-cycle, 12 VDC batteries. Depending on the location, terrain and distance to the gateway station, the

height of the antenna tower for the sensor node varied between 1 and 3 meters. Some of the modules were placed in weather-proof enclosures for protection. Cables between the sensory module and the control and communication module were protected by PVC pipes for outdoor deployment. The system configuration for a sensor node is illustrated in Figure 5.3. For field installation, one or more components may not be included.

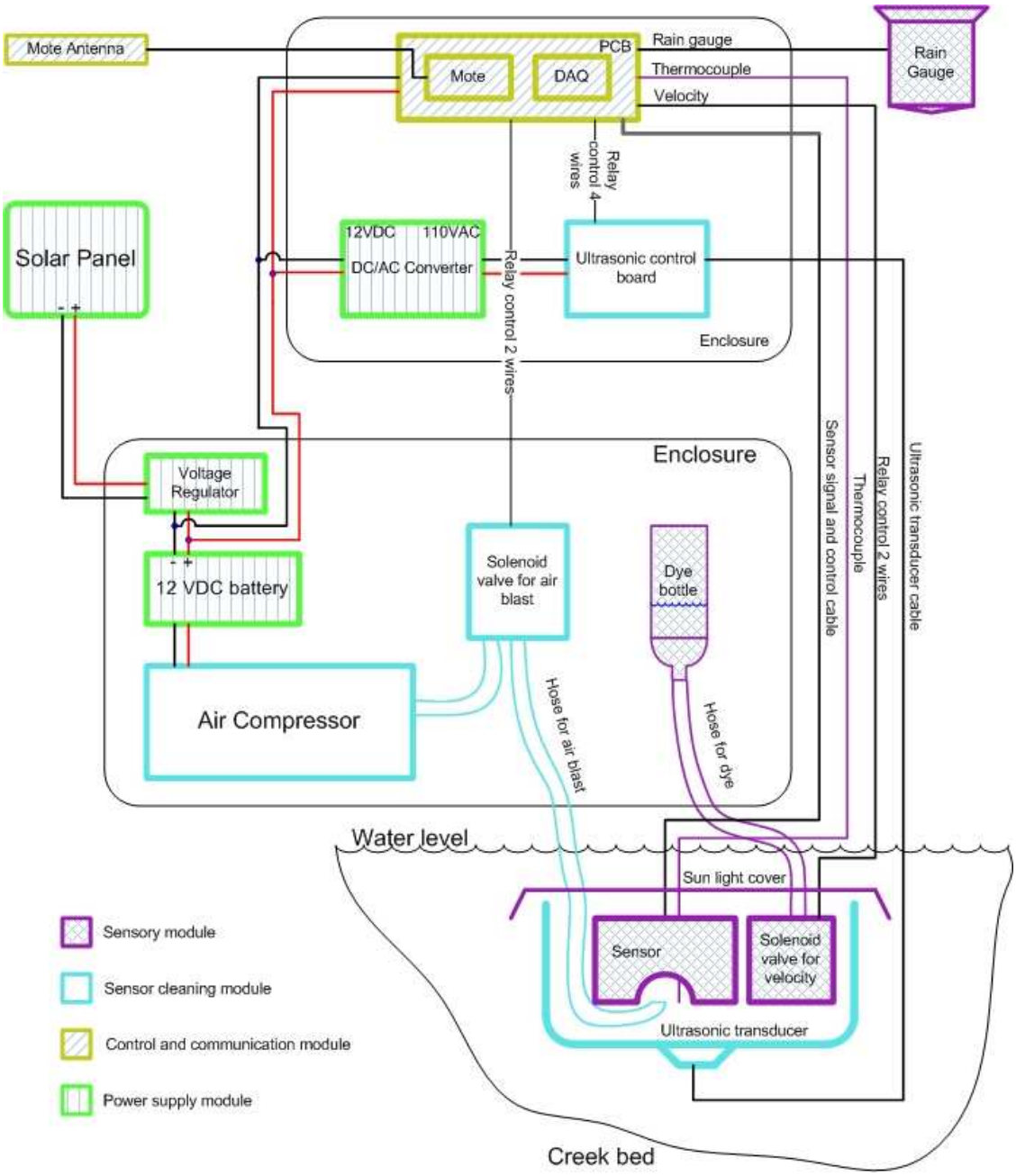


Figure 5.3 System configuration for a sensor node

In addition to the sediment measurement, the flow velocity was measured by two sets of orange LEDs and the corresponding phototransistors arranged on two rings on the optical sensor, with a distance of 0.04 m. The flow channel on the sensor was aligned with the direction of the water flow so that the water flow sequentially passed through the two rings, measuring the “upstream” (ORA45 1 or ORA180 1) and “downstream” (ORA45 2 or ORA180 2) signals, respectively. The “upstream” signals were also used for sediment measurement.

We injected blue food dyes into the water before velocity measurement to produce measurable color contrast. When released dye traveled through the stream, similar signal patterns could be observed from both the upstream and downstream with a time delay. After receiving the two sets of signals, the cross correlation was calculated between the upstream and downstream signals. The time delay at which the cross correlation was maximized was then used to calculate the flow velocity (Zhang, 2009). The principle of velocity measurement is illustrated in Figure 5.4.

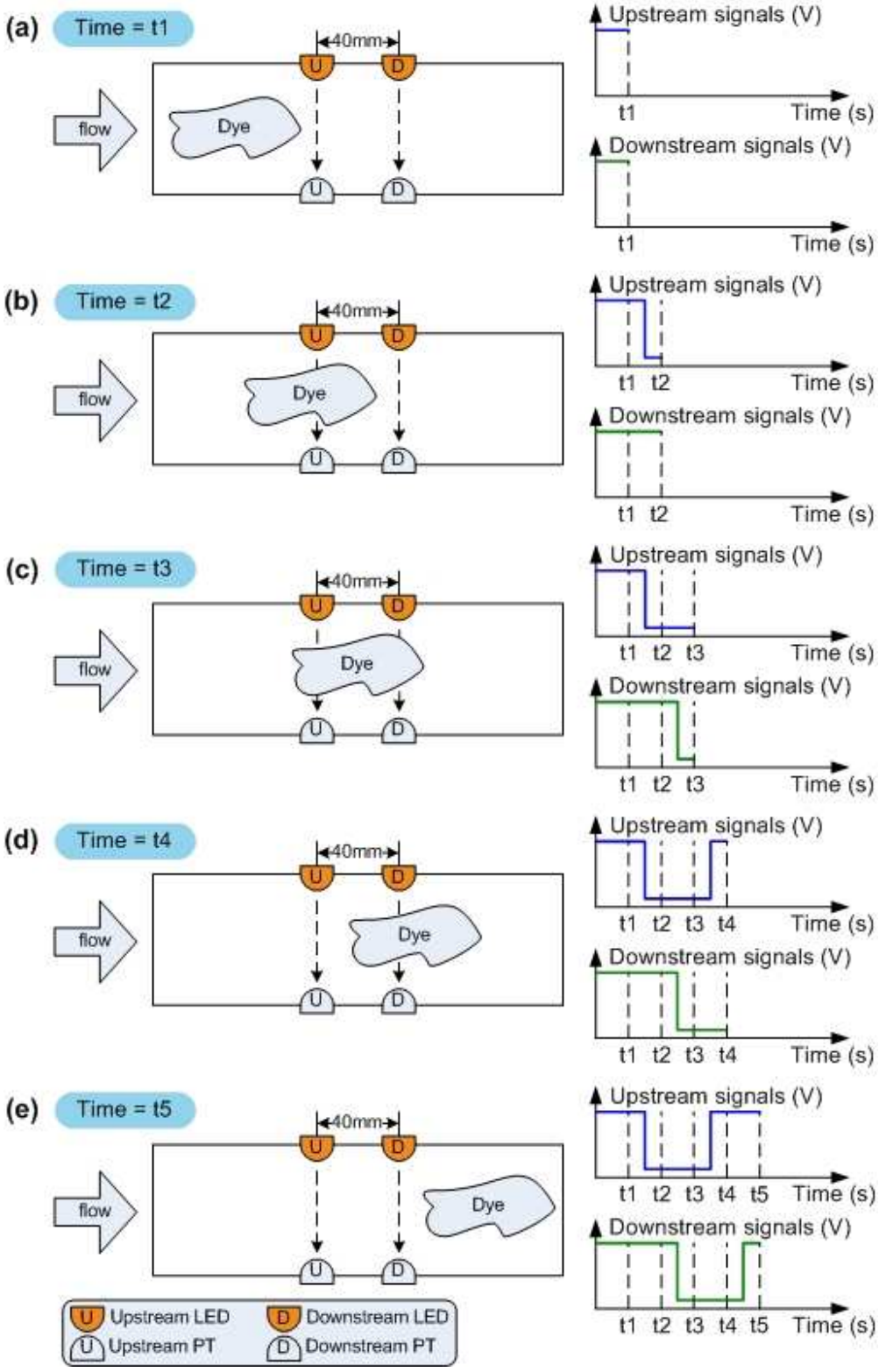


Figure 5.4 Principle of the velocity measurement

5.2.2.2 Gateway station

The gateway station included three modules. The Stargate module was used for collecting data from connected sensor nodes, saving and parsing data, and sending data to a radio communication module. The radio communication module included a CR206 datalogger (CR206, Campbell Scientific Inc., Logan, UT) and an antenna tower. This module is responsible for sending data to the central station or a repeater station. The power supply module included a solar panel, voltage regulators and 12 VDC batteries.

From Figure 4.10, a record had an 11-byte timestamp followed by 41 bytes, or occasionally more than 41 bytes, but generally not exceeding 43 bytes (Section 4.2.5.1) of data. A sensor without velocity measurement could transmit 2880 records per day. Using 43 bytes as the data length, the data rate was calculated as:

$$2880\times(43+11)\times 8bits/86400s = 14.4bits/s$$

For a sensor with velocity measurement, it needed to transmit an additional 1920 records per day. The data rate was calculated as:

$$(2880+1920)\times(43+11)\times 8bits/86400s = 24bits/s$$

As we mentioned in section 4.2.1.5 , a compact flash (CF) card was used to store the programs and the sensor data on the Stargate. We could calculate the capacity of a 1 GB CF card based on our data rate and storage method.

When data were saved on the Stargate, the characters were saved in ASCII (American Standard Code for Information Interchange) format. Each character occupied one byte. A carriage return (1 byte) was also added to each line in the text file. One sensor data contained $(43+11)\times 2 = 108$ characters and two lines (timestamp and record). As a result, we used totally 110 bytes to save a timestamp and its corresponding record into the data-log backup file.

A sensor node produced one sediment record every 30 seconds, hence a sensor node should generate the following amount of data within a year:

$$110bytes\times\left(\frac{60\,\mathrm{sec}}{\mathrm{min}}\times\frac{60\,\mathrm{min}}{hour}\times\frac{24hour}{day}\times\frac{365day}{year}\Big/30\,\mathrm{sec}\right)\Big/\frac{1024^3 bytes}{GB} = 0.1076GB/year$$

If velocity was to be transmitted along with the sediment data, additional 80 records were added each hour. Hence the number of bytes to be stored for the velocity data within a year can be calculated as:

$$\frac{110 bytes}{record} \times \frac{80 record}{hour} \times \frac{24 hour}{day} \times \frac{365 day}{year} \Big/ \frac{1024^3 bytes}{GB} = 0.072 GB / year$$

The total space required to store the programs on the CF card was less than 120 MB. Thus, the storage spaces available on a 1 GB CF card to store the data and the time lengths the card can be used to sore data under different scenarios can be found as follows.

Table 5.2 CF card capacity

Scenarios	Capacity needed to store one year of data	Years of usage for a 1 GB CF card
1 sensor node for sediment data only	0.1076 GB	8.1 years
1 sensor node for 1sediment and velocity data	0.1796 GB	4.9 years
2 sensor nodes for sediment data only	0.2152 GB	4.1 years
2 sensor nodes for both sediment and velocity data	0.2872 GB	3.1 years

The CR206 datalogger has an on-board 915 MHz, frequency hopping spread spectrum (FHSS) radio. The radio transmission range is 1.6 kilometers with 0 dBd, ¼ wave antenna (line-of-sight) and up to 16 kilometers (line-of-sight) with a high-gain antenna. The CR206's input channel configuration and small size are optimal for connecting a few sensors in an outdoor environment. It has a RS-232, 9-pin interface for communication between the datalogger and a computer. It has 512 k byte flash memory formatted for 4 byte per data point. The program flash memory allows a maximum storage of 6.5 k byte. The maximum operation speed is one scan per second. The communication protocol used for CR206 datalogger is PakBus. It is a packet-switched network protocol with routing capabilities. The CR206 datalogger uses a 12 VDC power supply with an average current drain of 20 mA in a radio-always-on condition (Campbell

Scientific, 2010a). Figure 5.5 shows a CR206 datalogger with a FHSS radio. We used the CR206 datalogger mainly as our mid-range radio device. In our system, the only measurement taken directly from CR206 was the battery voltage from datalogger's power supply.

Figure 5.5 CR206 datalogger with spread spectrum radio from Campbell Scientific, Inc.

The FHSS used on the CR206 datalogger is a signal transfer method. It rapidly switches the carrier frequency among several frequency channels to transmit signals. The sequence of the frequency switching, called pseudorandom sequence, is known by both the transmitter and the receiver. FHSS signals have a high interference resistance and are difficult to be intercepted. Also, the frequency bandwidth can be utilized more efficiently (Wikipedia, 2010b). When we configure the radio of the CR206 datalogger, we must give the same frequency hopping sequence for both the transmitter and the receiver.

As described in equation 5.1, high-gain antennas for both the transmitter and the receiver are essential to obtain high level of received signal strengths. A Yagi directional antenna is one of the high-gain devices that can pick up very weak signals in a specific direction. The available 900 MHz Yagi antenna has a manageable size and requires a fairly simple installation.

An omnidirectional antenna normally does not have higher gains on the particular direction than a Yagi directional antenna with similar size and cost; however, it covers 360° surrounding areas which give the advantage for transmitting and receiving from all directions simultaneously. Unlike the Yagi directional antenna which requires accurate direction adjustments, the omnidirectional antenna's installation is quite simple.

In some circumstances, if the numbers of transmitters are limited, and the locations of the transmitters are known, we can use 2-3 Yagi antennas with a signal splitter to replace an omnidirectional antenna at the receiver. The signal splitter at the receiver can split/combine signals for transmitters from different locations. Two types of antennas and a 2-way signal splitter are shown in Figure 5.6.

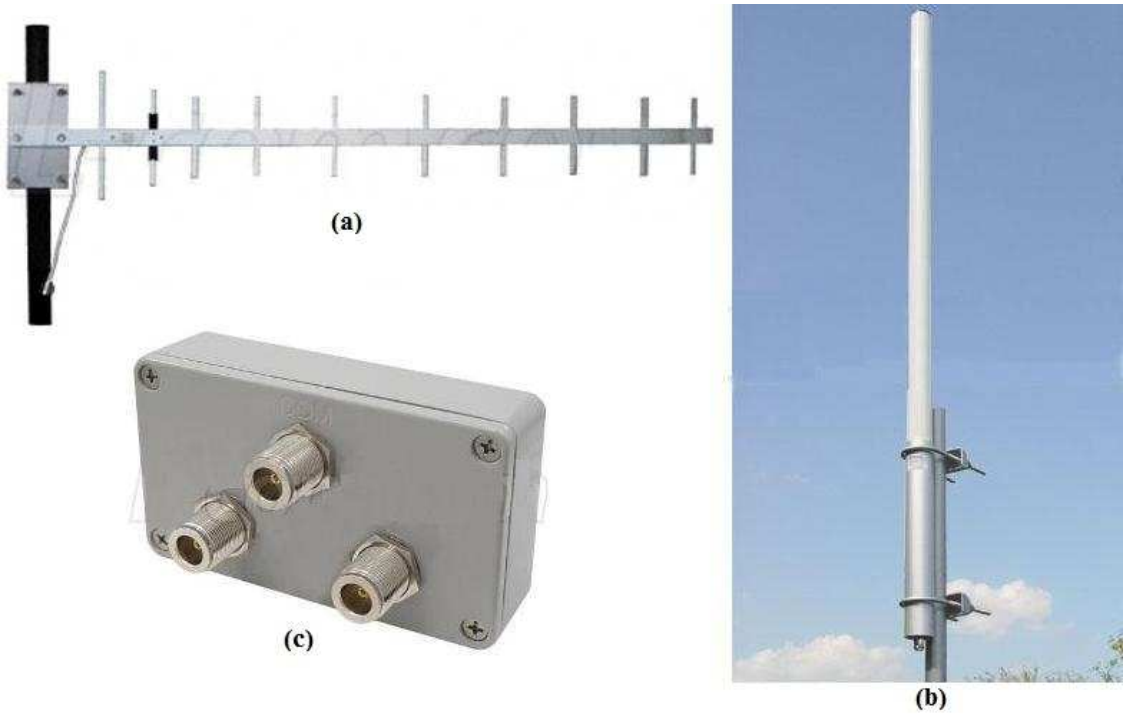


Figure 5.6 900 MHz antennas and signal splitter: (a) Yagi 14dBi directional antenna, (b) 8dBi omnidirectional antenna (c) 2-way signal splitter

In this study, a 3-meter high antenna tower was used in the gateway station. Most other components were properly installed in secured enclosures. Figure 5.7 shows the system configuration for the gateway station.

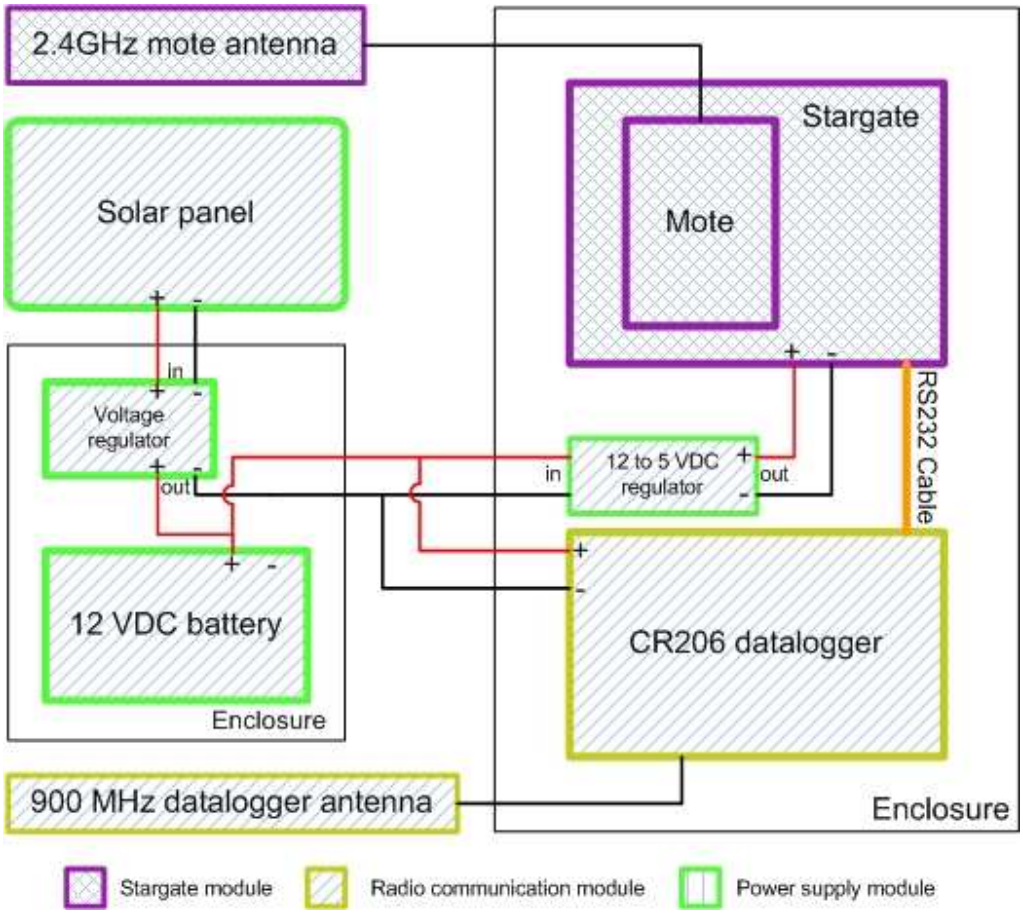


Figure 5.7 System configuration for the gateway station

5.2.2.3 Repeater station

The repeater station had only two modules: a radio communication module and a power supply module. It took the responsibility to receive signals from a gateway station and forward them to the central station. If multiple repeaters were used to relay signals, a repeater station may receive signals from or transmit them to another repeater. Selection of the repeater location was very important. Ideally, the repeater station should be located on an upland area, with a good line-of-sight to the central station. For the repeater station, omindirectional antenna was the common choice. However, Yagi antennas with a signal splitter were preferable. The power consumption at repeater station was generally very low (20 mA in average) and it required a small solar panel for recharging the battery.

5.2.2.4 Central station

The central station was the final data sink in the remote region. It included three modules: a radio communication module, a cellular modem module and a power supply module.

The cellular modem we used was a Raven modem, model Raven XT (Sierra Wireless, Richmond, BC, Canada). It provided sophisticated remote monitoring and controlling functions. For Code Division Multiple Access (CDMA) cellular system, the modem used 800 MHz carrier frequency, and for Global System for Mobile Communications (GSM) cellular system, the modem used 900 MHz carrier frequency. The average transmit/receive current drain at 12 VDC was 239 mA. It had a mini USB port and a RS232 port (Sierra Wireless, 2007). The Raven modem in the cellular modem module was connected to the CR206 datalogger in the radio communication module by a null modem RS-232 cable.

There were different types of Raven modems manufactured for major cellular carriers in the US, such as AT&T, Verizon, and T-Mobile. We selected the cellular carrier based on a comparison of signal coverage. Figure 5.8 shows the compact form of a Raven XT.

Figure 5.8 The AirLink Raven XT cellular modem

Before deployment, Raven modem should be configured. This involves setting up the serial port, assigning the IP address of the server computer, specifying the machine

port on the server computer, choosing TCP protocol, and disabling the TCP timeout. These configuration settings can be saved in the modem and do not need to be re-configured unless parameter is changed.

5.2.2.5 PCB board and cleaning system

The PCB board needed a major reversion since many new functions had been added. Based on these newly added functions, the mote programs also needed to be modified. The main changes included adding different circuits on the board and re-arranging the layout for the mote unit (MICAZ and MDA300). Three double pole double throw (DPDT) relays were added. We used one relay to control the air-blast cleaning device, another relay to control the dye releasing for velocity measurement. The third relay was a backup relay. A K-type thermocouple (TT-K-24-500, OMEGA Engineering, Inc. Stamford, CT) was used to measure the water temperature in the creek. We added a conditioning circuit for the K-type thermocouple measurement. We provided a 3.3 V power supply for the MDA300 DAQ board. We added resistor arrays for calibration of the sediment sensor, also added the circuit for the rain gauge. A PCB functional diagram is shown in Figure 5.9.

The cleaning system included two types of mechanisms: the air-blast and the ultrasonic. The air-blast cleaning mechanism used an air-compressor to fill a tank of air to a certain pressure level. When the pressure dropped below a threshold, the air-compressor started to refill the tank. An air hose from the tank conducted the air to the air outlets on the sensor. A solenoid valve was used to release pressurized air through the air hose to the sensor, thus the air bubble could blow away dirt and biofouling materials. The mote controlled a relay on the PCB board to open and close the solenoid valve for the air hose. When the air-compressor refilled the tank, it needed 240 W (20 A at 12 VDC) instance power supply, which caused a significant voltage drop on the battery. The voltage drop could be used to verify the control function of the mote.

Another cleaning method was using an ultrasonic transducer to generate mechanical waves in a container that enclosed the sensor. The water movement caused by the waves in the container cleaned the sensor lenses. The ultrasonic transducer was controlled by a control board, which also allowed adjustment of the duration of the

cleaning operation. The mote used two built-in relays on the MDA300 board to start and reset the ultrasonic control board, respectively.

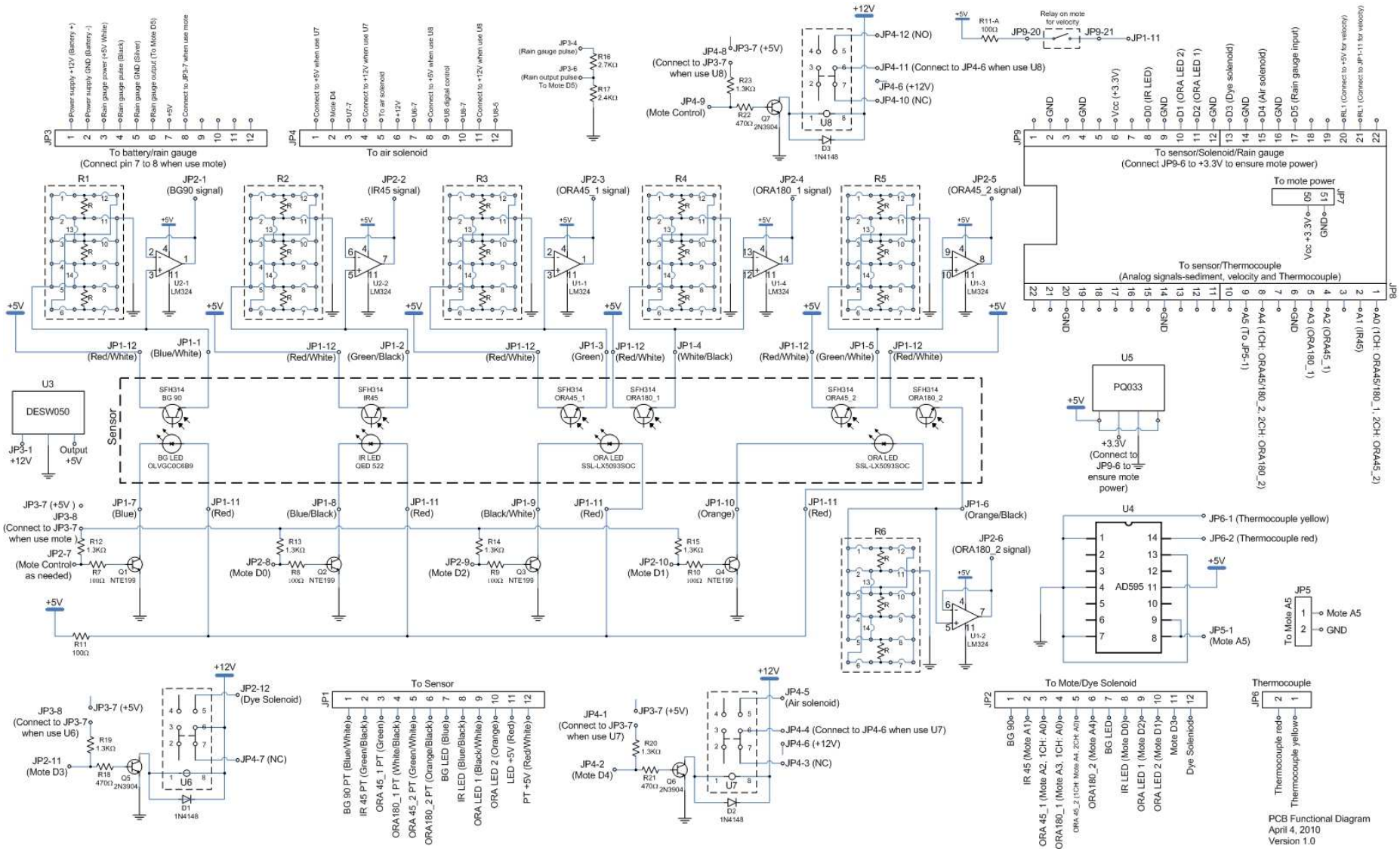


Figure 5.9 Functional diagram of the PCB control board

5.2.3 Energy harvesting and saving methods

Energy harvesting is also known as power harvesting or energy scavenging, it is the process by which energy is derived, captured and stored from external sources, e.g., solar energy, thermal energy, wind energy, salinity gradients, and kinetic energy. Frequently, the term “energy harvesting” is used when talking about electrical power needed for small, wireless, autonomous devices, such as those used in wearable electronics and wireless sensor networks.

5.2.3.1 Solar panel

In this study, we used solar power as the main power source in the field. To design and build a solar power system, we needed to consider the amount of energy on demand, gauge of wire connectors, types and capacities of batteries, locations of the power components, and installation issues.

To calculate the power consumption of the system, we needed to find the power usage of each component in the system. In order to explain the design principle and method, we used the Little Kitten site as an example. This site had a sensor node with air-blast cleaning and velocity measurement. The power system also supported a nearby gateway station. Table 5.3 shows the current drains and durations of power usages of the electrical components in a sensor node and a gateway station.

Table 5.3 Current drain and duration of usage for electrical components in a sensor node and a gateway station at the Little Kitten site

Component	Current drain	Duration of power usage
Mote (MICAZ)	27 mA (Average)	24 hours
Optical sensor with MDA300 and PCB board	29 mA (Average)	24 hours
Air-blast cleaning system	20 A	24 minutes per day
Stargate	200 mA (Average)	24 hours
Datalogger (CR206)	20 mA (Average)	24 hours
Air-blast solenoid valve	1.5 A	10 seconds per hour
Velocity solenoid valve	1.5 A	30 ms per hour

The daily power consumption by each component can then be calculated as.

$$E_{daily} = I_{Avg} \times T_{use} \times V \quad \textbf{(5.3)}$$

where:

E_{daily} = Daily power consumption (Wh)

I_{Avg} = Average current drain (A)

T_{use} = Expected daily use (hr)

V = Voltage of the power supply (V)

Table 5.4 Daily power consumption for the Little Kitten site

Component	Daily power consumption (Wh/day)
Two motes (MICAZ)	15.55
Optical sensor with MDA300 and PCB board	8.35
Air-blast cleaning system	96
Stargate	57.6
Datalogger (CR206)	5.76
Two solenoid valves	1.21
Total	184.47

The Little Kitten experimental site is located at longitude 96.6339º W and latitude 39.2057º N. To design a year-around, reliable power supply, the average daily insolation (4.6 hr) during the coldest months in a year - from December to February – was used in the calculation (Roberts, 1991) of needed power:

$$P = \frac{E_{daily} \times 100\%}{S_{Avg} \times Eff_{batt}} \quad \textbf{(5.4)}$$

where

P = Power need (W)

E_{daily} = Daily power consumption (Wh)

S_{Avg} = Average daily insolation (hr)

Eff_{batt} = Charging efficiency of battery (%)

A typical charging efficiency for a lead-acid battery was 80%. Thus, the power need at the Little Kitten site was calculated as 50.13 W. Considering the availability and cost, a solar panel rated at 60 W was selected for this site. To charge a 12 V battery, a voltage regulator with 5 A current rating was also selected.

To maximize solar energy absorption, the solar panel was mounted towards the south, with an inclination angle close to the area's latitude.

5.2.3.2 The battery

As solar panels give only instant power, batteries are needed to provide sustainable power storage. The recommended battery for a solar panel power system was deep-cycle battery. Different from a shallow-cycle battery such as a car battery, deep-cycle battery can be discharged with more stored energies and maintain a long life. When we start a vehicle, the car battery is discharged at a large current within a short time period and then immediately recharged once the car engine is started. On the contrary, a deep-cycle battery is discharged slowly at a smaller current, usually during the night, and is gradually recharged, usually during the day.

In this study, the battery used was a deep-cycle, marine battery with a full capacity of 200 Ah. This capacity was specified for an operating temperature of 20ºC. If lower operating temperatures were expected, the capacity would be reduced by 1% per ^{o}C. According to Country Studies US (Country Studies US, 2008), the lowest temperature for Manhattan, KS, usually occurred in January, and the average value was around -8.33 ºC. Therefore, the total usable capacity of the battery was reduced to

$$C_{batt} = 200 \times 71.67\% = 143.34Ah \qquad \textbf{(5.5)}$$

The “worst-case scenario” for a solar-charged battery was minimum to none insolation on consecutive days. This may be caused by rain, snow, or clouds. To sustain the system operation under the “worst-case scenario”, the minimum battery capacity A had to meet the following condition:

$$A = \frac{E_{daily} \times D}{12V} \qquad \textbf{(5.6)}$$

where

A = Minimum battery capacity in a 12V system (Ah)

E_{daily} = Daily power consumption (Wh)

D = number of days without charging (Day)

Typically, a stationary deep-cycle battery is designed for 80% depth-of-discharge. However, using shallower depth-of-discharge can prolong the battery's life. Assuming 60% depth-of-discharge, the number of batteries needed for the Little Kitten site can be calculated from the following equation:

$$N = \frac{A \times 100\%}{C_{batt} \times M_{dc}} \quad \textbf{(5.7)}$$

where

N = The number of batteries required

A = Minimum battery capacity in a 12V system (Ah)

C_{batt} = Total usable capacity of a battery (Ah)

M_{dc} = depth of discharge (60%)

For the Little Kitten site, equation 5.7 gives $N = 1.79$, indicating that two batteries are needed for a sustainable power supply system.

The following Table gives the estimation of daily power consumption, solar panel power need and number of deep-cycle batteries needed for all installation sites in US. Based on this table, we can specify the solar panel and the number of batteries.

Table 5.5 The estimated daily power consumption, solar panel power need and number of deep-cycle batteries needed for all installation sites

Site	E_{daily} **-Daily power consumption (Wh)**	***P*-Power need (W)**	***N*-Number of 200Ah batteries needed**
Little Kitten sensor node and gateway	184.48	50.13	2
Wildcat bridge sensor node and gateway	184.48	50.13	2
Wildcat /Silver Creek sensor nodes	200.61	54.51	2

and gateway			
Above Keats repeater	5.76	1.57	1
Cico tank repeater	5.76	1.57	1
Colbert Hills central	74.59	20.27	1
Pine Knot sensor nodes and gateway	297.82	77.56	3
Upatoi Creek sensor nodes and gateway	297.82	77.56	3
Buena Vista turn repeater	5.76	1.50	1
Buena Vista central	74.59	19.42	1
Gunpowder sensor nodes	129.47	35.18	1
Gunpowder gateway	71.14	19.33	1
Anita sensor nodes, gateway and central	275.20	74.78	3

5.2.4 System software design

The system software included programs for data acquisition, relay control, air-blast and velocity control, the Stargate data process and communication, the datalogger routing protocol, time synchronization and MySQL database.

5.2.4.1 Data acquisition program

Based on the mote program described in section 4.2.5.1 , which was designed for acquiring data from the optical sensors, we added solenoid control for air cleaning and velocity test, as well as pulse counting from a rain gauge and temperature measurement from a thermocouple.

In the mote program, we used an unsigned 8-bit integer variable “lights” to assign the digital ports on the MDA300 board. From the most significant bit (MSB) to the least significant bit (LSB), each bit in this variable controlled a digital port or a relay. The

ports controlled were: RL2, RL1, D5, D4, D3, D2, D1, and D0. RL1 and RL2 were two built-in relays on the MDA300 board and D0-D5 were six digital I/Os. RL1 was a normally-open relay and RL2 was a normally-closed relay. The initial status of the digital ports on the MDA300 board were output low, RL1 open and RL2 closed. To keep the initial status of these digital ports, the "lights" must have an initial value as 0xc0. Table 5.6 gives the detailed values of 'lights' for functions that it controls.

Table 5.6 Digital port control in the mote program

Instruction on MDA300	Operation on variable "lights"	Function
D0 high	OR 0x01	Turn on IR LED
D2 high	OR 0x04	Turn on ORA LED
D0, D1 and D2 low	AND 0xf8	Turn off all LEDs
D4 high	OR 0x10	Turn on air solenoid valve
D4 low	AND 0xef	Turn off air solenoid valve
D1, D2 and D3 high	OR 0x0e	Turn on velocity test
D3 low	AND 0xf7	Turn off dye solenoid valve
D1, D2 and D3 low	AND 0xf1	Turn off velocity LEDs

* The initial "lights" value was 0xc0.

** D5 was set up as an input. The program blocked the output value of "lights" on the input digital port.

To take the samples for velocity measurement, the mote program turned on the orange LEDs of the upstream and downstream at the same time for 3 seconds. Simultaneously, it opened a solenoid valve to release dye and then closed it after 30 ms. At a sampling rate of 279 Hz, the program acquired 512 samples at both the upstream and downstream locations. The readings were voltage levels resulting from current-to-voltage converters that converted the current signals from the phototransistors at the upstream and downstream locations, 45º or 180 º from the LEDs. The user can choose to use either the 45º or the 180 º signal by changing the wiring to the analog and digital ports without changing the program. Because these voltage data were used to calculate the flow velocity later, we called them "velocity raw data". The velocity raw data were saved in the mote until all 1024 samples were taken. At that time the program put the data into

payloads of records based on their incoming sequence. From Figure 4.10, it can be seen that each record contained 13 channels in the payload. Thus, 40 records were needed for the 512 samples from either the upstream or the downstream raw data. We used “02” and “03” in “Packet ID” to distinguish samples from the upstream and downstream, respectively, and used “R1” to give a sequence number for each record of the velocity data. Figure 5.10 shows the structure of the velocity raw data record.

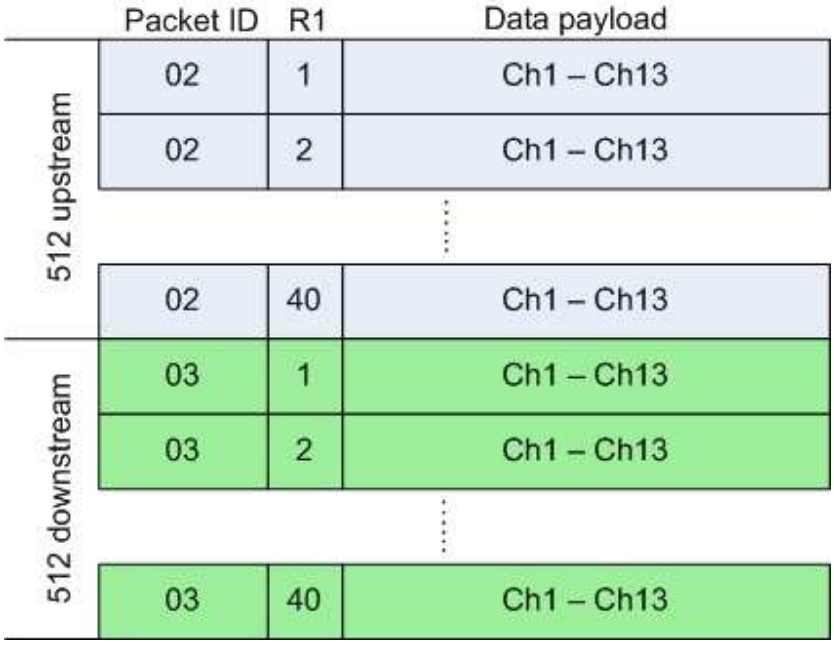


Figure 5.10 Structure of the velocity raw data records

A tipping bucket rain gauge (TE525, Campbell Scientific Inc., Logan, UT) was used at the sensor node. It needed a 5 V excitation voltage to produce pulses at bucket tipping. Each time the bucket was tipped, a pulse of approximately 135 ms was generated due to switch closure (Campbell Scientific, 2010b). In the mote program, we setup a digital port (D5) as an input, and predefined it as a totalizer counter to check the rising-edge of the incoming pulse. The program used a mask-able interrupt to monitor the rising-edge from the digital port. Whenever there was a pulse generated from the rain gauge, the program monitored the changes in the counter.

Components and their relationships within a nesC program for rain gauge reading are illustrated in Figure 5.11.

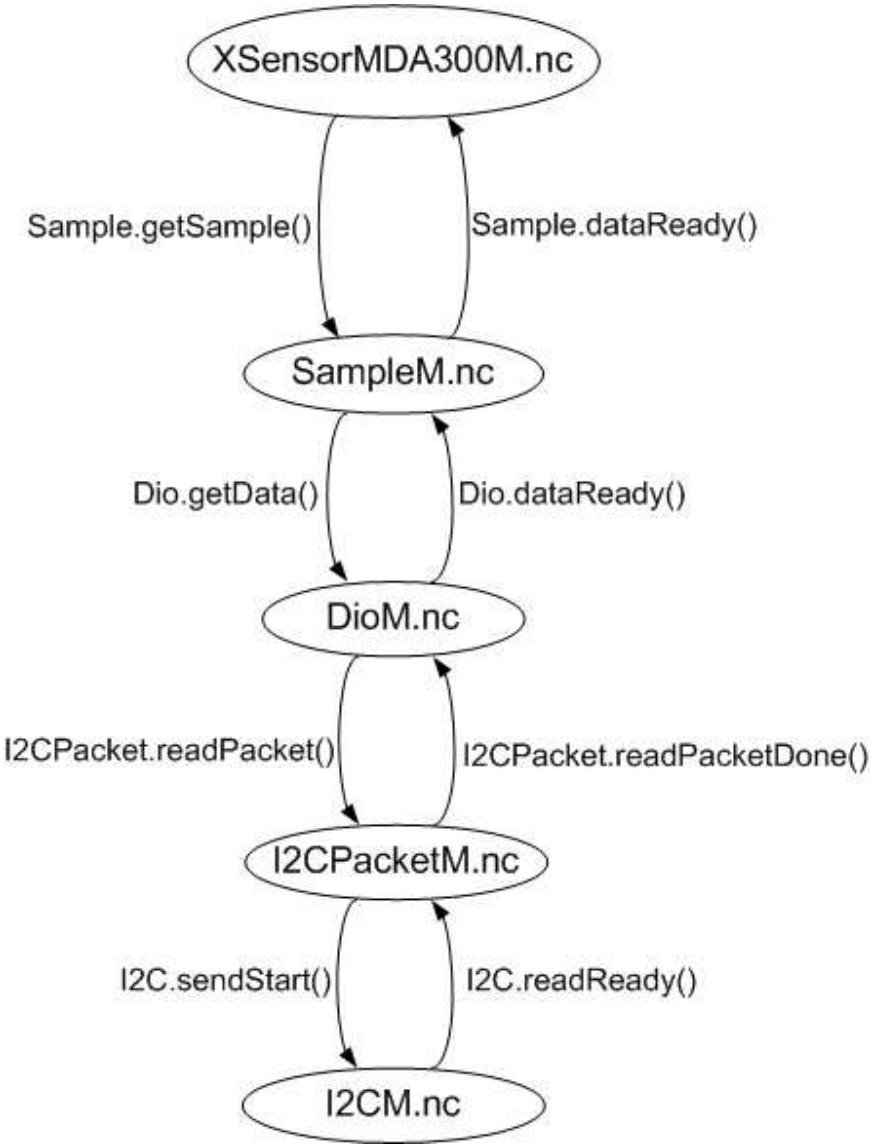


Figure 5.11 nesC program components for measuring a rain gauge counter

5.2.4.2 Control program

Digital I/O ports were used to control the relays in the system. As described in Section 5.2.4.1 , RL1 and RL2 were two built-in relays on the MDA300 board. They were single pole single throw (SPST) optical solid state relays that required a very small control signal to control up to 100 V voltage or 150 mA current. In the mote program, we controlled these two relays as we did for other digital outputs. The default values for these two relays were set to high.

For three external relays on the PCB board, since the digital output ports on the MDA300 were the open-collector type, pull-up resistors were used to increase the output voltage to the desirable levels. In the mote program, we used “lights” to control them.

In order to save energy and cost, in some experimental sites, we used one air compressor to support air-blast cleaning for two sensors. In this case, only one mote was used to control the solenoid valve. To avoid cleaning a sensor while it was taking sediment readings, time synchronization among participating sensors was needed.

In this “one air compressor for two sensors” configuration, for time synchronization purpose, the timers on these two sensor motes were no longer started automatically and repeatedly when we turned on the power on these motes. Instead, the timers were waiting for a beacon signal from the base mote on the Stargate. The base mote on the Stargate sent a beacon signal to both sensors every 30 seconds. The beacon signal was a normal signal with a unique packet ID (0x07) and an empty data payload. Motes on the sensors identified the beacon signal by checking its packet ID. Upon receiving the beacon signal, the timer on each sensor mote was started for timing. The sensor mote that controlled the air solenoid valve started to sample the sediment data one second after received the beacon signal. The other sensor mote took sediment samples 16 seconds after receiving the beacon signal. The air-blast was turned on right after the first mote finished taking sample data to start a 10-second cleaning. Thus, the cleaning would be completed before the second mote started to take the samples. This schedule guaranteed that no sample data was taken during the air cleaning operation, and the mote at the gateway station received a sample data every 15 seconds. The interval for air-blast cleaning was set at 1 hour. Figure 5.12 illustrates the mote control synchronization mechanism.

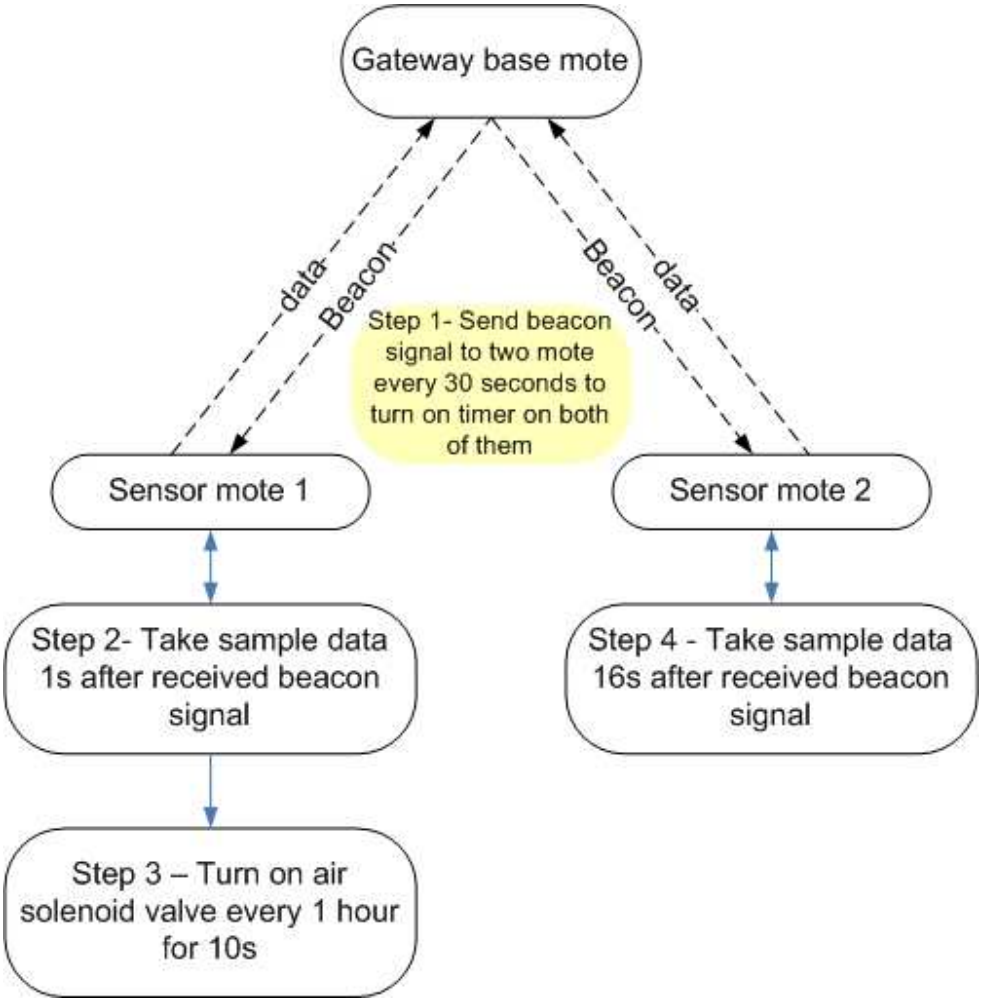


Figure 5.12 Illustration of mote control synchronization mechanism

5.2.4.3 The Stargate program

In the three-tier WSN, the AirCard was no longer used on the Stargate for data transmission; instead, the data was sent to a CR206 datalogger via a RS-232 cable. As a result, we did not need to use the FTP programs. We added a few C programs to handle the data processing and transfer tasks.

A Linux line reader script program (Readxlish.sh) was further developed to adopt the changes as velocity raw data were also required to be transferred for experimental purpose. The velocity measurement was scheduled to be taken every 1 hour. Before the velocity data were taken, the program copied 12 new records from a data-log backup file and saved them to a data-sending file every six minutes. The data-log backup file was a file to save all incoming records from the sensor motes. The data-sending file was a file

to temporarily store data before the data was sent out to the datalogger. The velocity data, on the other hand, were generated by the sensor in 1.84 seconds. The line reader script program constantly checked the total number of new records every 10 seconds. Receiving more than 12 records within the past 10-second interval was an indication of the receipt of a set of velocity raw data. In this case, the program would store both received velocity and sediment data in the data-sending file.

After receiving the sediment and velocity raw data for three months, there were approximately 690,000 records in the data-log backup file. When the line reader script program tried to get the number of total records from the data-log backup file, we started to observe a slower response from the line reader script program, and it became even worse as the number of records further increased. A larger number of records in the file prevented us from reading and retrieving data quickly, which eventually caused the program to stop reading. This problem was solved using the method described below.

Whenever the number of records in the data-log backup file exceeded 216,000, the line reader script program sent out the newly received records as usual, and then used a subroutine to copy the data-log backup file into a new file, which was archived with a file name that contained the current date and time. The program then removed the current data-log backup file, and created an empty backup file for the incoming data. Figure 5.13 shows the flowchart of the script control program.

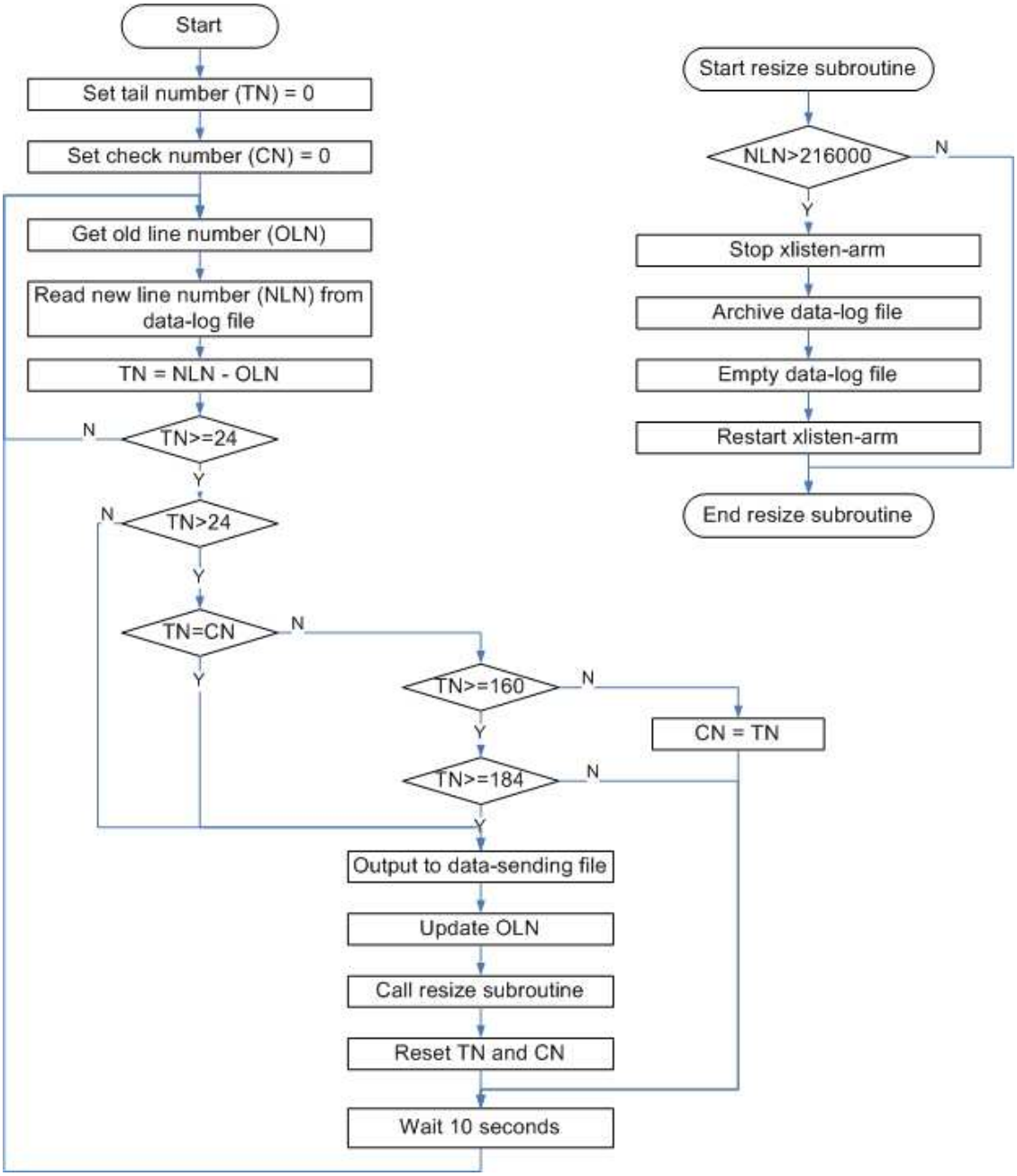


Figure 5.13 The flowchart for the line reader script program

A data processing and controlling C program (prelogger.c) was used to read the data-sending file. When the program found date and time that were newer than the time previously retrieved from the data-sending file, it started to retrieve and validate data from the data-sending file. For each received record, the program removed the “Header”, “PacketID” and “Reserved”, and added “day”, “time” and “GroupID” at the beginning of the record. This newly created record was called a logger record. Based on the datalogger

package structure which we will discuss in section 5.2.4.4 , the program filled these received logger records into a predefined package and saved them to a queued, first-in-first-out (FIFO) buffer, a circular array. The package would be sent out when there was no new data waiting for reading or the queue was full. The program requested time synchronization information from the CR206 datalogger whenever the queue was empty and the time from the last synchronization checking had passed 12 hours. Normal sediment data and the velocity data were distinguished by their group ID. This is shown in Table 5.7.

Table 5.7 The group ID ranges used for different data types

Data type	**Group ID range**
Sediment data	0 to 127
Velocity upstream data	201 to 240
Velocity downstream data	301 to 340

The program calculated the cross-correlation between the upstream and downstream data for velocity measurement. It also calculated the cyclic redundancy check (CRC) of each record for error checking before sent them out. CRC-16 was used for the calculation. The divisor value was “0x8005” and the generator polynomial was $x^{16} + x^{15} + x^2 + 1$. The flowchart for this program is shown in Figure 5.14.

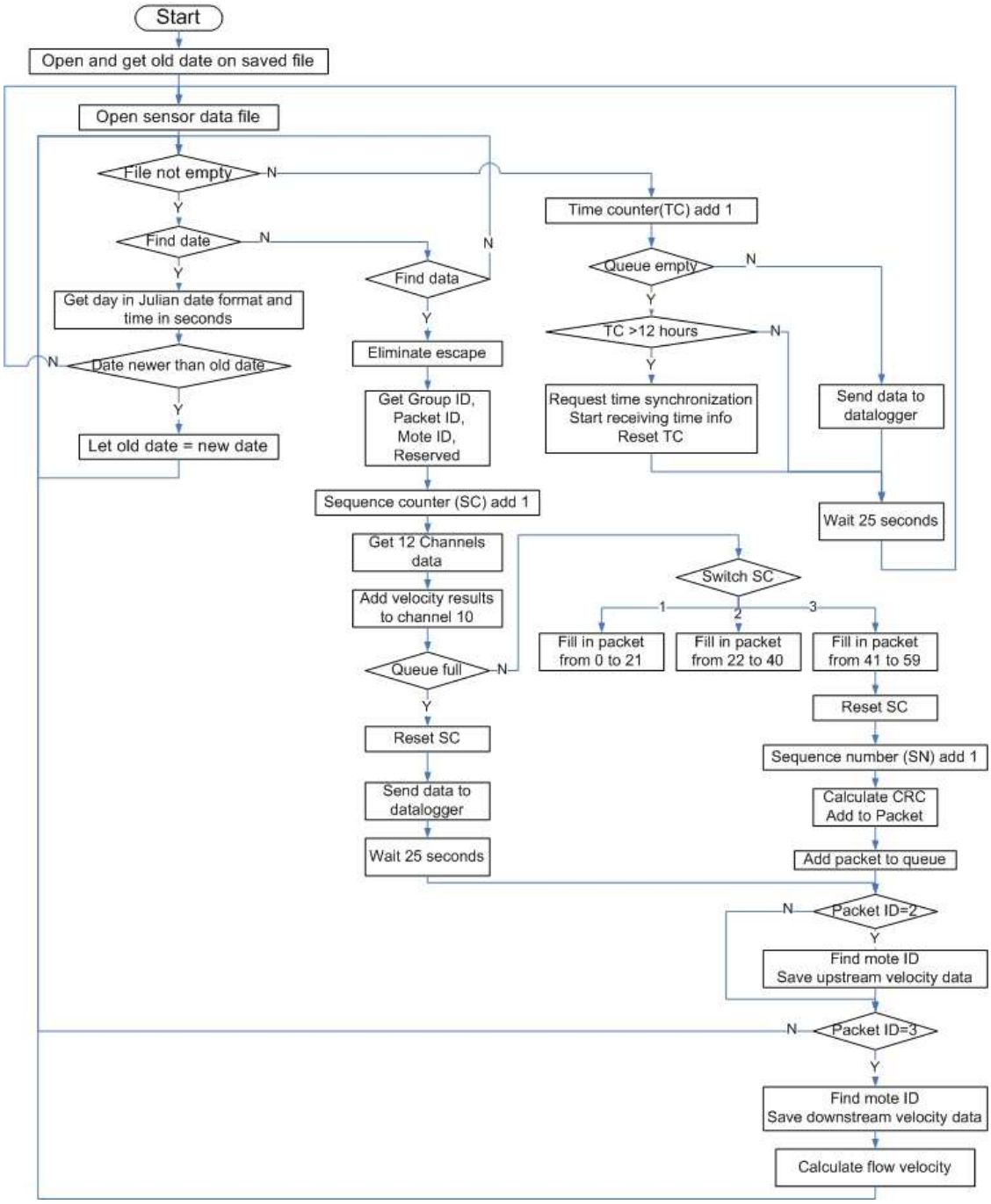


Figure 5.14 Flowchart for data processing and control C program "prelogger.c"

5.2.4.4 Data flow control in the MRWN

CRBasic is a programming language for Campbell Scientifics'datalogger software design. All programs on the datalogger are programmed in the CRBasic language. To use the serial input functions on the datalogger, we must install a special S operating system

on the datalogger, and use version “v06S” of compiler on the computer for software design.

The CR206 datalogger has limited variables that can be used for data transmission. We used totally 60 variables for the purpose of sending data. The layout of a package structure organized in the datalogger at the gateway stations is shown in Figure 5.15. Each package must have a PakBus address to identify the origin of the data, and a sequence number to tell the order of the packages. In each package, the gateway and the central station battery voltage levels, as well as the repeater’s battery voltage level if applicable, were included in addition to the sensor data. Each package contained three logger records. The logger records could be distinguished by their group ID, mote ID and timestamps. Each logger record had its CRC at the end for error checking, as shown in Figure 5.15.

PakBusAdd	Seqnum	Gateway Batt	day	time	Group ID	Mote ID	Mote Batt	Ch1 IR45 on	Ch2 IR45 off
Ch3 ORA45 on	Ch4 ORA45 off	Ch5 ORA180 on	Ch6 ORA180 off	Ch7	Ch8 Rain	Ch9 Temp	Ch10 Velocity	Ch11	Ch12
Ch13	CRC	day	time	Group ID	Mote ID	Mote Batt	Ch1 IR45 on	Ch2 IR45 off	Ch3 ORA45 on
Ch4 ORA45 off	Ch5 ORA180 on	Ch6 ORA180 off	Ch7	Ch8 Rain	Ch9 Temp	Ch10 Velocity	Ch11	Ch12	Repeater Batt
CRC	day	time	Group ID	Mote ID	Mote Batt	Ch1 IR45 on	Ch2 IR45 off	Ch3 ORA45 on	Ch4 ORA45 off
Ch5 ORA180 on	Ch6 ORA180 off	Ch7	Ch8 Rain	Ch9 Temp	Ch10 Velocity	Ch11	Ch12	Central Batt	CRC

Figure 5.15 The structure of a package formed in the datalogger at the gateway stations

The CR 206 dataloggers were used in the mid-tier of the three-tier WSN, the MRWN, as transmitters at the gateway stations, transceivers at the repeater stations, and receivers at the central stations. The transmitter datalogger at the gateway stations received data from the Stargate through a 9-pin, RS232 serial port, and was responsible for time synchronization between the Stargate and the datalogger. When both the datalogger and the Stargate were started or restarted at the same time, the datalogger needed very short transition time from its initiate status to its normal operating status,

while the Stargate took longer time. The program on the datalogger waited 3 minutes until the Stargate was ready, and then sent time information to synchronize the Stargate. The datalogger also responded to synchronization request from the Stargate every 12 hours.

In the datalogger used at a gateway station, when data was ready to be sent to an upper level station, i.e. a repeater or a central station, the datalogger program waited for a beacon signal “receiving flag IDLE” from the upper-level station. The “receiving flag IDLE” signal was a one variable package. Once the datalogger at the gateway station received this beacon signal, it changed a flag in the datalogger from 0 to 1, giving the green light to start transmitting data to the upper level station. At the same time, in the upper level station, the beacon signal was changed to “receiving flag BUSY”. This beacon signal would set the flags on other gateway stations from 1 to 0, thus preventing simultaneous data transmissions from multiple gateway stations. During data transmission, the datalogger at the gateway station sent the data in six 40-byte segments to the upper-level station. For each of the segments, the datalogger looked for a SUCCESS acknowledgement from the upper-level station to confirm the success of the transmission. For each segment, if the transmission failed, the datalogger at the gateway station would repeatedly transmit the segment until a SUCCESS acknowledgement was received. After data transmission of all six segments was completed, the datalogger at the upper-level station would reset the “receiving flag” to “IDLE”, which gave chances to other gateway stations to send data. Figure 5.16 gives the flowchart of the CRBasic program for the transmitter datalogger.

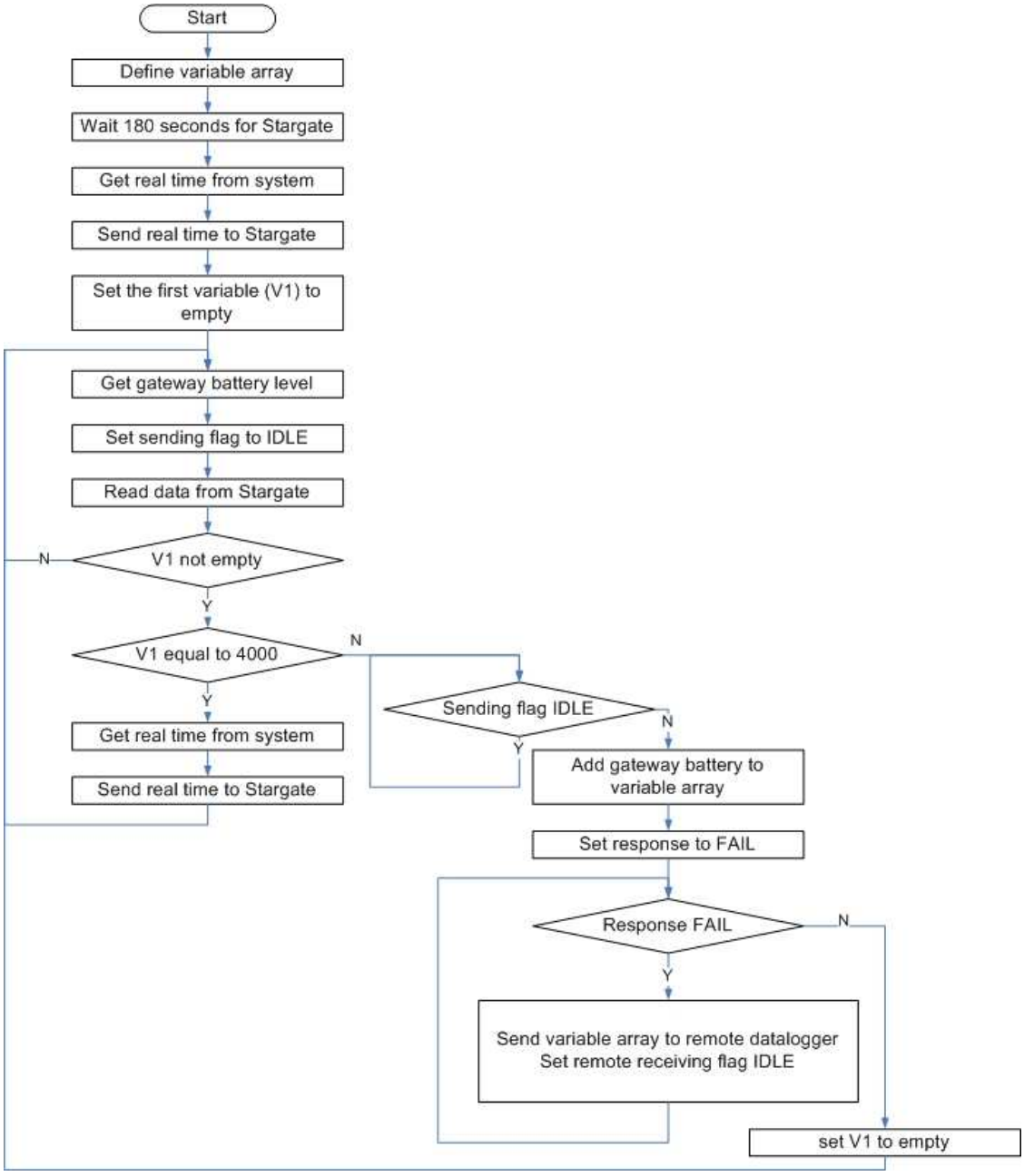


Figure 5.16 Flowchart for the CRBasic program written for the transmitter datalogger

When the datalogger at a repeater station was neither receiving data from a lower-level station nor sending data to an upper-level station, it would send a “receiving flag IDLE” beacon signal to its lower-level stations, (gateway or repeater stations) in a sequence that was predefined based on their PakBus address. During a 3-second period after the beacon signal was sent, if any of the lower-level stations responded to this signal,

indicating the need for data transmission, the repeater program would first change the "receiving flag" to "BUSY" so that other lower-level stations could not send data to the repeater. After the data transmission was completed, the repeater station saved the received data locally and watched the "receiving flag" beacon signal from its own upper-level station (a central station or another repeater station). Once the flag from the upper-level station became "IDLE", the repeater station would forward the saved data, in six 40-byte segments, to the upper-level station. Differently from the gateway station, the repeater station gave only 3 seconds for transmission of each 40-byte data segment, regardless of the actual time needed for transmission and transmission error. If the transmission was completed within 3 seconds, the repeater would not start another transmission before the 3-second period ended. On the other hand, if the transmission failed or was incomplete at the end of the 3-second period, the repeater station would transmit whatever in the segment to the upper-level station, and send a new beacon signal to the next lower-level station on the predefined sequence. This "wait-and-go" scheme prevented time waste and associated data loss. Figure 5.17 shows the flowchart for the CRBasic datalogger program at a repeater station.

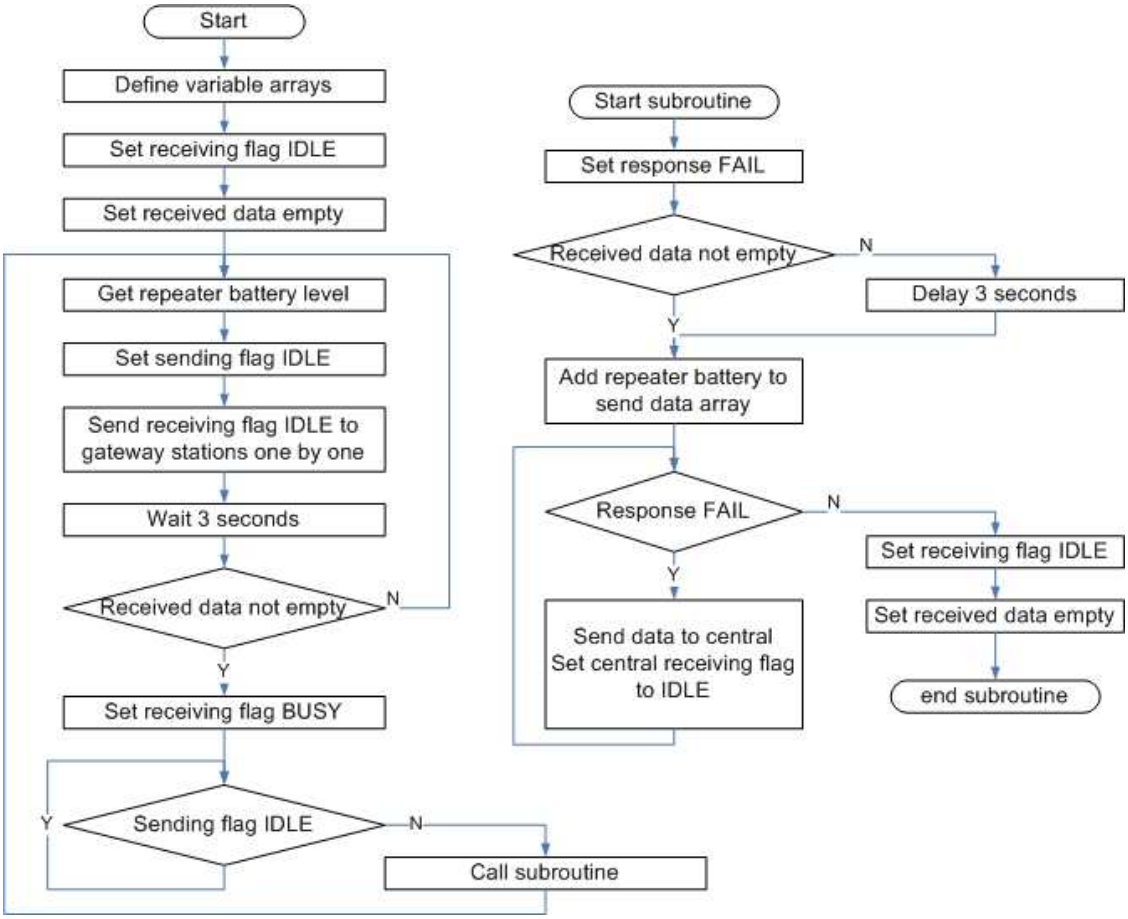


Figure 5.17 Flowchart for the CRBasic dataloger program written for the repeater station

The data receiving part of the program used for the datalogger at a central station was identical to that on a repeater station. The flowchart of the CRBasic program for the datalogger at the central station is shown in Figure 5.18.

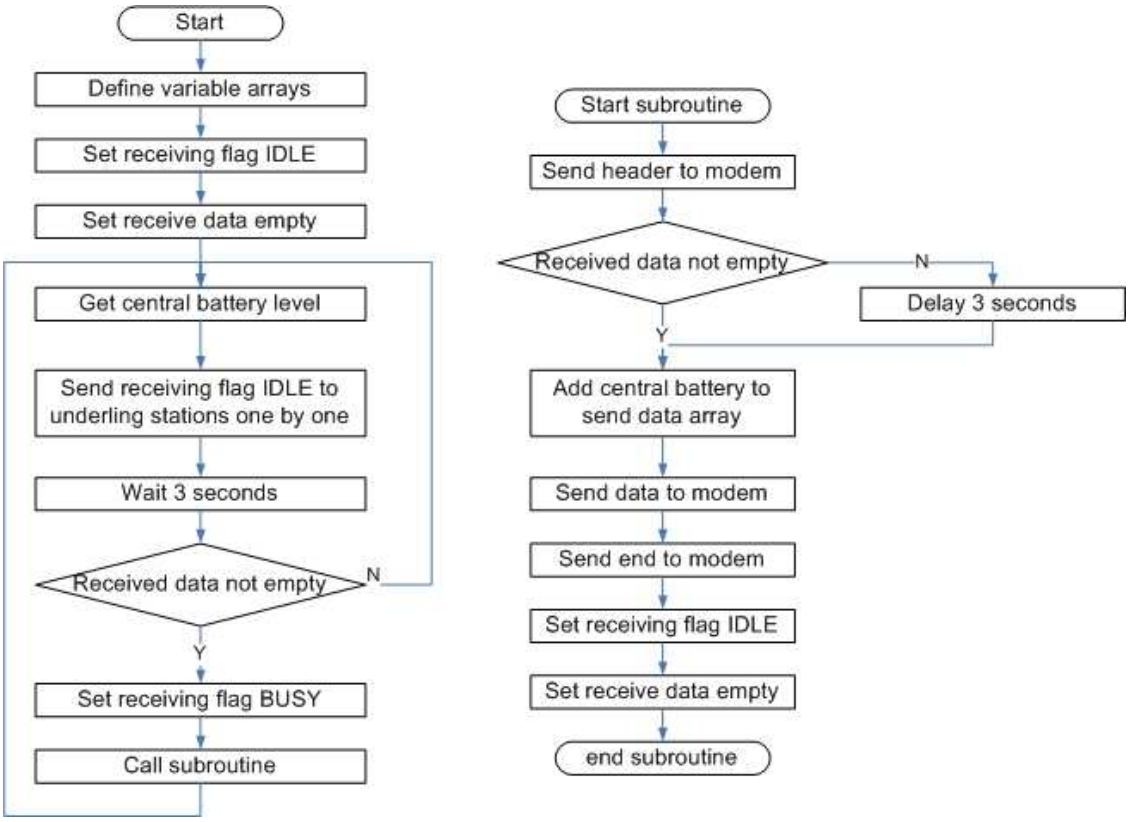


Figure 5.18 Flowchart for the CRBasic program for the datalogger at the central station

The data flow protocol among the gateway, repeater, and central stations is illustrated in Figure 5.19. The source codes are listed in the Appendix D.

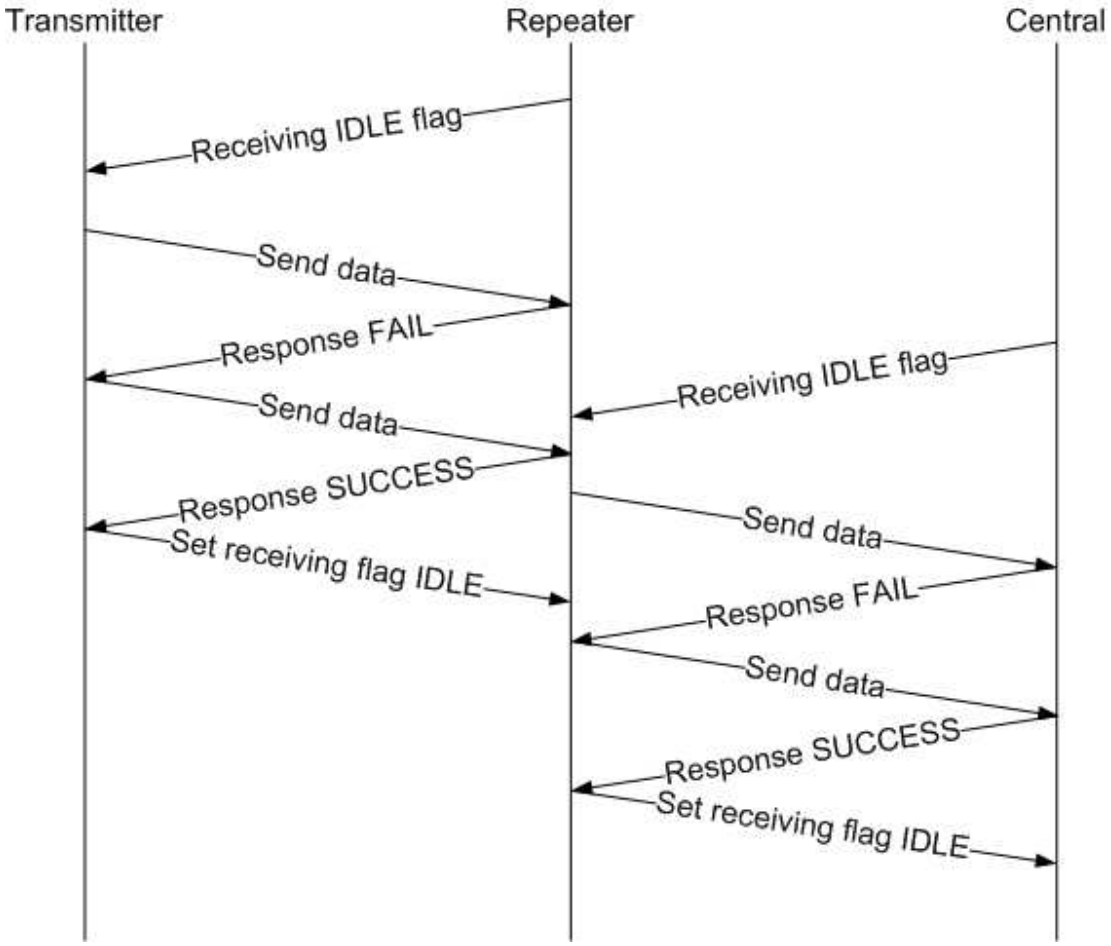


Figure 5.19 Data transmission protocol

5.2.4.5 Time synchronization program

The clock on the Stargate was stopped once it lost the power supply. On the contrary, the clock on the datalogger had an internal battery and kept running when it lost the power supply. To get system time on the Stargate, we synchronized the Stargate's clock with the datalogger. There were two situations where time synchronization was performed. The first was when the Stargate was starting up. Every 12 hours, if there was no data package waiting in the queue, the Stargate also would request time information from the datalogger to readjust its own time. This frequent adjustment guaranteed time synchronization between the Stargate and the datalogger.

In both cases, the Stargate program (prelogger.c) would issue a request to the datalogger for time information and call a time synchronization program (syncdate.c) while yielding the control of the RS-232 port, which was connected to the datalogger, to

this program. Thus, time information on the datalogger can be read by the Stargate via the serial port. The time synchronization program would be terminated whenever the time information was received, or 120 seconds after the program was called if no time information was ever received. Once the time information was received, a Linux script time synchronization program (syndate.sh) used this information to actually adjust the clock on the Stargate. Figure 5.20 shows the flowchart of the time synchronization program on the Stargate. The program source codes are listed in the Appendix E.

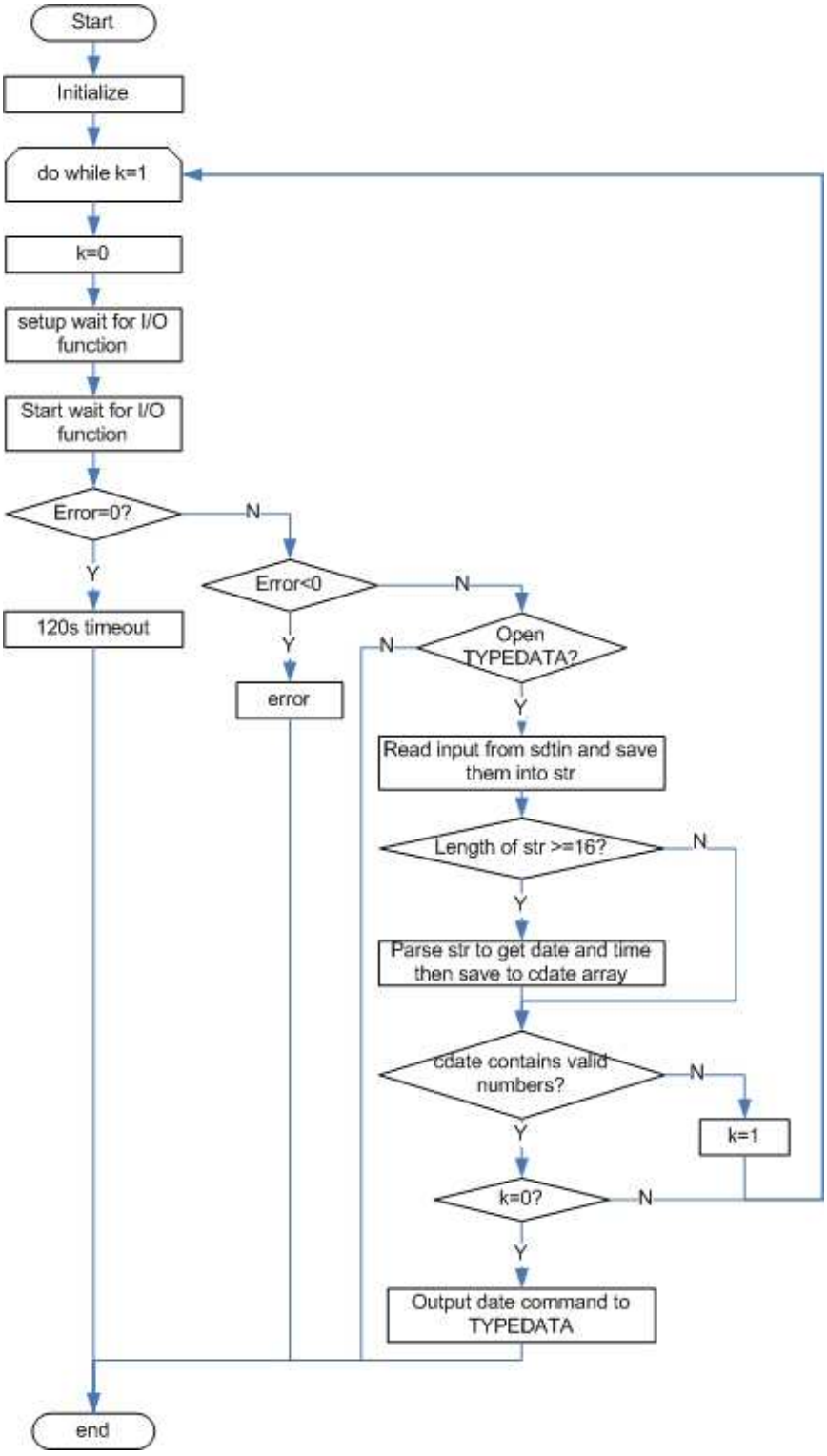


Figure 5.20 Flowchart of the time synchronization program

5.2.4.6 Server computer program

A java server program was used on the server computer to continuously receive data packages from the designated network socket. As predefined, each package used a $ sign as the header to indicate the start of the package, and a # sign for the end. Once the program found the header, it started to check the location of each data segment received in the package, and assigned it to corresponding column in the database. In the database, raw data were converted to engineering units and formats that were readable by the final user. Based on the received data package, each logger record was used to calculate the CRC again using the same polynomial formula used in the Stargate, and compared it to the received CRC code. If a discrepancy was found, a “CRC not pass” mark would be added to the record in the database, indicating errors in the received record. The flowchart for the Java server program is shown in Figure 5.21. The source code for the program is in the Appendix F.

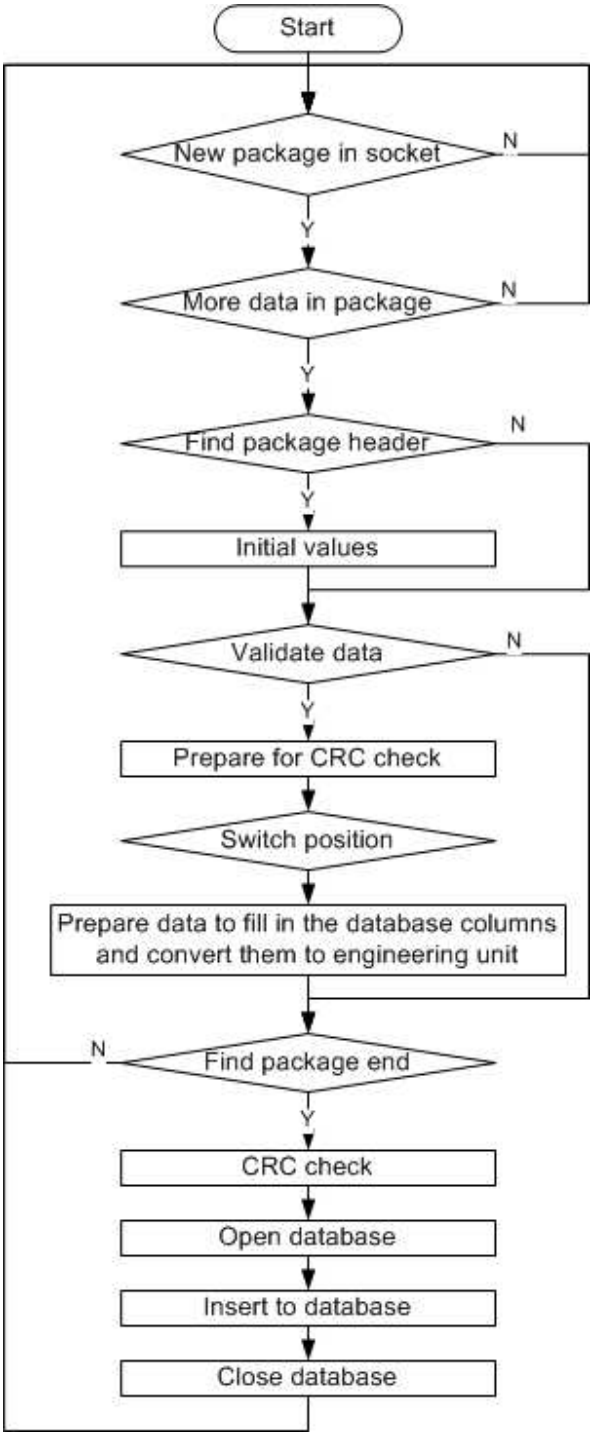


Figure 5.21 Flowchart of a Java server program

5.2.4.7 The MySQL database design

MySQL database is a popular, open-source relational database. With programming language specific APIs, many languages, including Java, can access the

database seamlessly. A MySQL server allows multi-user access, which was important as we collaborated with the Oklahoma State University (OSU) project team in design of the web GIS system for Internet applications.

In the database, we used one main table (tblresults) to hold all received data from the network, except the velocity raw data, which were saved in another table (tblvelocity_riley) separately. We used a table (tblsensors) to store information such as names, IDs, site descriptions and locations of all installation sites. The webGIS performance information, such as errors, warnings and user logins, was saved into table "tbllog". Descriptions of the columns in the main table are listed in Table 5.8.

Table 5.8 Description of columns in the main table of the MySQL database

Column	**Description**	**Range**
ID	A unique identification number for each record in the table. This number automatically incremented by 1 when a new record was added.	
UniqueCode	A unique number assigned to each sensor node based on its location.	
PakBusAdd	Address for PakBus protocol.	0-4000
SeqNum	Sequence number for packages sent from the gateway station	1-65535
GroupID	Group identification number for each mote.	0-127
MoteID	Identification number for each mote.	0-255
MoteBatt	Battery voltage on mote (Volt)	0-20
Ch1~Ch6	Ch1 ~ Ch6 are voltage reading for IR45 on, IR45 off, ORA45 on, ORA45 off, ORA180 on, ORA180 off. (Volt)	0-5
Ch7	Reserved	
Ch8	Velocity calculation results (meter/second)	0-30
Ch9	Thermocouple reading (Celsius degree)	0-50
Ch10	Rain gauge reading in counter numbers	0-65535
Ch11~Ch13	Reserved	
Ch14	CRC check mark (0 for pass, 1 for not pass)	0,1

GatewayBatt	Battery voltage at the gateway station. (Volt)	0-20
RepeaterBatt	Battery voltage at the repeater station. (Volt)	0-20
CenterBatt	Battery voltage at the central station. (Volt)	0-20
DateTime	Date and time in following format (yyyy-mm-dd hh:mm:ss)	

5.2.5 Site selection

For the three-tier WSN, selection of the sites for sensor nodes, gateway stations, repeater stations, and central stations was vitally important to the performance of the entire network. Candidate sites were first selected from maps. Field survey was then performed to measure important indices, such as received signal strength indicator (RSSI) between the candidate locations. One of the instruments used during the survey was a handheld spectrum analyzer: model N9340B, 100.0 kHz – 3.0 GHz (Agilent Technologies, 2011). High buildings, hills, and forests in the transmission path were avoided.

After the survey, extensive field tests were conducted to refine and confirm the selected sites. For this, we used two dataloggers, each installed with a signal strength test program provided by Campbell Scientific Inc. and a Yagi antenna. The program used the number of blinks of two LEDs on the datalogger to indicate the strengths of the received and transmitted signals. During the test, each datalogger was held by a person at a selected site. By changing the locations, heights, and directions of the antennas at the two sites, signal strengths can be measured, and the best antenna positions can be determined. With the help of compasses, accurate antenna directions were recorded.

5.2.6 Deployment and installation

The three-tier WSN was installed at three military installations in the US. We first set up a pilot experimental site at Fort. Riley, Kansas. As we tested the system and gained experiences on sensor deployment and system installation, we started to develop the second site at Fort. Benning, Georgia. Finally, we installed our last site at Aberdeen Proving Ground (APG), Maryland. At each site, we deployed four sensors. Thus, the total number of sensors in the network was 12. Measurement data from all 12 sensors were

transferred back to the database server at Manhattan, Kansas, through three different cellular carriers.

5.2.6.1 Issues of field deployment

Several issues were encountered during the deployment of the system. This section discusses these issues and the methods used to resolve them in order to improve the reliability, efficiency, and life span.

5.2.6.1.1 Enabling brown-out detection

A reliability problem detected was mote malfunctioning under unstable power supplies. A mote would malfunction when the supply voltage was too low, and would not resume normal operations after the power level was resumed. A bad mote required two trips of an operator to the field to retrieve and reinstall. For a solar charged battery that has been deployed for a long time, the voltage level may drop greatly during nights and cause this problem.

A solution to this problem was to enable the "brown-out detection function" available on the mote. This function monitored the power supply level during operation by comparing it to a fixed trigger level. When power dropped below the trigger level, the brown-out detection activated an internal reset to stop the microcontroller processor (MCU) immediately. When power rose again to a fixed restart level which was higher than the trigger level, the MCU was restarted after a certain delay (Atmel Corporation, 2006). The drawback of this function was that it consumed energy even in when the mote was in the deep sleep mode. After applying the brown-out detection function to all the motes for more than one year, no mote malfunctioning has been found even under the worst weather conditions, including winter snowstorms.

Figure 5.22 illustrates the brown-out detection in operation. As can be seen in the figure, there was a time delay after VCC (battery voltage) reached the restart level. This delay gave more time for the power supply to be recovered before the MCU started to work again.

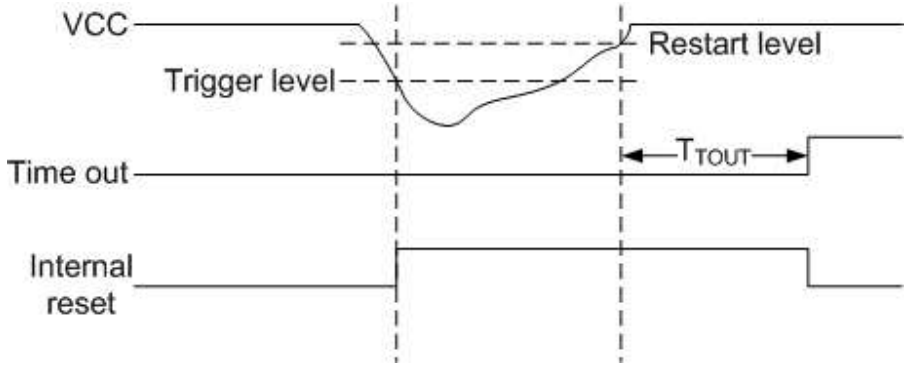


Figure 5.22 Illustration of brown-out detection (from Atmel Corporation, 2006)

5.2.6.1.2 Memory management for Stargate

One problem observed on the Stargate was that its operation slowed down after running for a few months. We investigated this problem and found that the memory usage on the Stargate was very high. This was due to the fact that the Linux system was automatically caching frequently accessed files in the memory to achieve a faster speed for reading and writing. Without the need to read from a hard disk (CF card for the Stargate), the frequently accessed files could be retrieved straightly from the memory in a much faster rate. These files eventually were written to the CF card. The system only used the unused part in the memory for caching files. Therefore, it did not affect the other applications in the memory. Since the memory had a finite size of 64MB on the Stargate, it was not possible to cache everything. The Linux system cleaned those less frequently accessed files and kept the more frequently accessed ones.

After we added raw data transmission for the velocity measurement, the amount of data accumulated in the data-log backup file (described in section 5.2.4.3) was larger than before, and the frequency of accessing this file was also increased. After a few months of continuous operation, the file in the CF card, as well as its copy in the memory, became very large, occupying most of the available memory in the system. Although this high memory usage did not cause system crash or malfunctioning owing to the memory management of the Linux system, it apparently slowed down the operation significantly.

To solve this problem, we activated the “virtual memory” in the Linux system by creating a swap file on the CF card. The swap file was an external memory storage on the CF card. With the swap file, files in the memory that were not presently used could be

saved to the virtual memory so that the memory could be used by other applications. We performed an extensive laboratory test to verify this method. We used a mote to send massive data (transmitted one record every 2 seconds) to the Stargate in a few days to simulate generation of a large amount of data to the data-log backup file in a few months. We monitored changes in the memory usage on the Stargate. Once the memory usage approached 64MB on the Stargate, the usage of virtual memory started to grow.

In addition, we used a line reader script program (Readxlisten.sh) to reduce the size of the data-log backup file when the number of records in the file reached a threshold. This program was discussed in section 5.2.4.3 . By using this program, when the number of records reached the threshold, the current data-log backup file would be renamed and archived in the CF card. A new, empty data-log backup file was then created to receive the incoming records. We conducted tests on this method, and found that, when the old backup file was archived, no immediate reduction in memory usage was seen. The copy of the previous data-log backup file in the memory was not cleaned until 2-3 minutes later. Overall, the memory usage dropped to a lower level after the Readxlisten.sh program was used.

5.2.6.1.3 Insect attack and vandalism

Damages by insects, especially ants, was found to be a major cause for failure after the system was deployed. We used ant repellent to protect the electronic components. Vandalism was another problem we encountered. Wires were cut several times, a battery was stolen, and a rain gauge was used as ash tray. We used chain locks had the wires underground or protected by PVC pipes.

5.2.6.2 System installation

The three-tier WSN was installed at three military installations across the country (Figure 5.23). For each installation, we chose a cellular carrier that provided the best signal coverage. They were T-mobile's Global System for Mobile communications (GSM) system and its General Packet Radio Service (GPRS) data service for Fort Riley, Kansas, Verizon's Code Division Multiple Access (CDMA) system and its Evolution-Data Optimized (EVDO) data service for Fort Benning, Georgia, and AT&T's GSM system and its Enhanced Data rates for GSM Evolution (EDGE) data service for

Aberdeen Proving Ground, Maryland.

Figure 5.23 Three military installations where the three-tier WSN was installed.

5.2.6.2.1 Fort Riley site

In October, 2008, we started to install a three-tier WSN to a pilot experimental site in the Fort Riley - Manhattan area in Kansas. Locations of four sensor nodes were carefully selected at the Little Kitten Creek, Wildcat Bridge, Wildcat Creek, and Silver Creek sites (Figure 5.24). From the sensor nodes, measurement data were transmitted to a nearby gateway station using MICAZ motes through their 2.4GHz radio channels. In the gateway station, a Stargate received these data and sent them to a datalogger with 915 MHz spread spectrum radio transceiver (CR206, Campbell Scientific Inc., Logan, UT) via an RS-232 cable. This datalogger then transmitted the data to the datalogger at a repeater station, where the data were relayed to the central station. Upon receiving the data, the datalogger at the central station sent the data to a cellular modem, which transmitted the data back to the server computer at Kansas State University. Data were stored in a robust open-source database server, MySQL server, which can be accessed from the Internet.

Figure 5.24 Map of the Manhattan-Fort Riley, KS Pilot Experimental Site

The central station was installed near the Colbert Hills water tower at longitude 96.6465º W and latitude 39.2299º N. The elevation of the location was 415 meters and it was at least 70 meters higher than any of the sensor nodes, gateway stations, and repeater stations. Although it was not a pure "line-of-sight" situation between the central station and the gateway/repeater stations, there were no obvious obstacles between their antennas. Two repeater stations were installed. One was on "Cico tank" at longitude 96.6288º W and latitude 39.2095º N for Little Kitten site. Another was on the "Above Keats" at longitude 96.7115º W and latitude 39.2125º N for both Wildcat bridge gateway and Silver Creek gateway stations. The gateway locations were chosen based on two principles: 1) close to the sensor nodes, 2) can communicate with the central station or a repeater station. The Little Kitten site had a gateway station and a sensor node installed at longitude 96.6339º W and latitude 39.2059º N. The Wildcat bridge gateway station and a sensor node were installed at longitude 96.6289º W and latitude 39.2188º N. The Silver Creek gateway station was installed at longitude 96.7206º W and latitude 39.2229º N. Two sensor nodes shared this gateway station, one at the Silver Creek, and another at the Wildcat Creek.

For all the gateway, repeater, and central stations in Fort Riley, we used three-meter high antenna towers. Each tower was secured with three guy wires with adjustable lengths. The guy wires were covered by PVC pipes and the pipes were painted in neon-orange color to warn approaching human beings or animals. 900 MHz Yagi directional

antennas (HyperLink Technologies Inc. Boca Raton, FL) with 14dBi gain were used for datalogger communications among all the gateway, repeater, and central stations at the Fort Riley site.

A central station needed to communicate with gateway and repeater stations in different directions. Thus, an omnidirectional antenna may be a good choice. However, omnidirectional antennas radiated and received unfocused signals in all directions. Thus, using an omnidirectional antenna at the central station may lead to low efficiency in signal reception (Yi et al., 2003). In this study, we used two Yagi directional antennas at the central station, pointing to the repeaters at "Cico tank" and "Above Keats", respectively. We also used a signal splitter/combiner to merge signals from the two repeater stations. This setup efficiently improved the communication. Figure 5.25 shows the central station at Ft. Riley site, KS.

Figure 5.25 Central station at Ft. Riley site, KS

Similar to the central station, we used two 900 MHz Yagi antennas at the Cico tank repeater station. One Yagi antenna was pointing to the central station at Colbert Hills, another was pointing to the gateway station at Little Kitten. Hence the signals from the Little Kitten gateway station can be relayed to the central station.

Three antennas were used on the Above Keats repeater station. A 3-way signal splitter was used so that three antennas could be connected to the datalogger at the same time. A 900 MHz omnidirectional antenna was used for communication with the central station at the Colbert Hills. Two more 900 MHz Yagi directional antennas were used to connect to the Wildcat bridge and Silver Creek gateway stations, respectively. Figure 5.26 shows two repeater stations at the Ft. Riley site, KS.

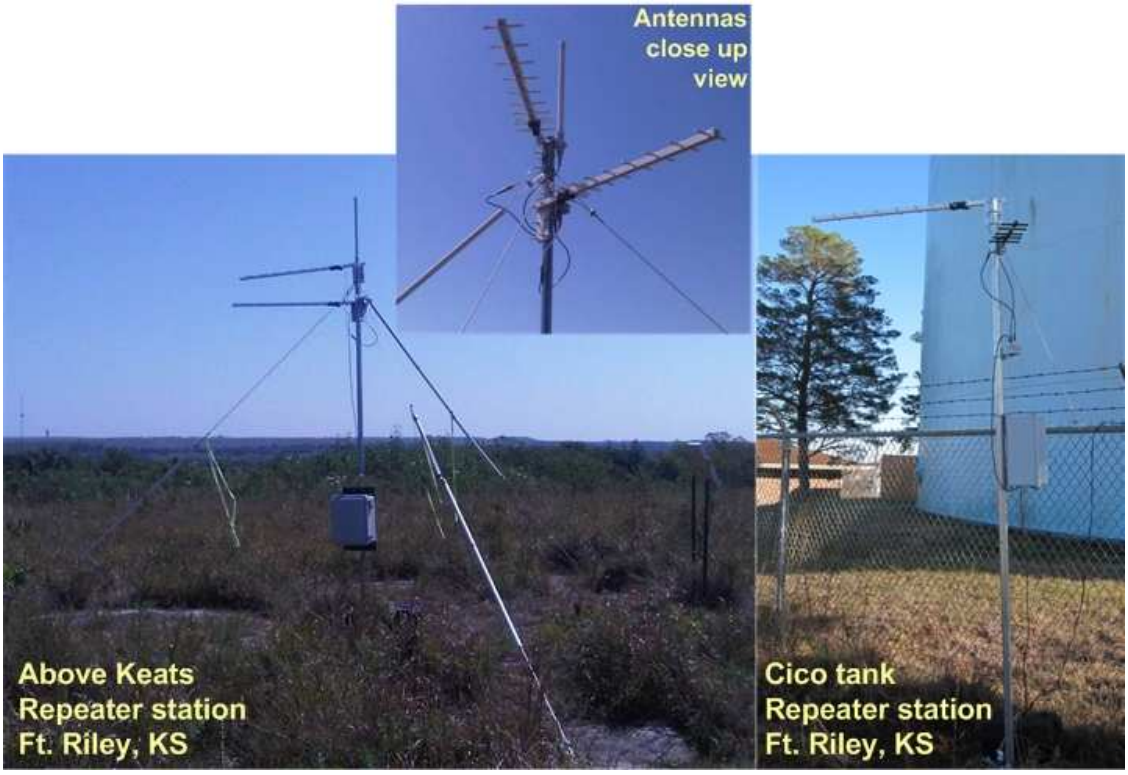


Figure 5.26 Repeater stations at Ft. Riley site, KS

At the Little Kitten site, we used 2.4 GHz Yagi antennas for both the gateway base mote and the sensor mote for LWSN communication. The distance between the sensor node and the gateway station was about 24 meters. Flow velocity was measured every hour. The optical sensor was air-cleaned every hour, 30 minutes after the velocity measurement. A rain gauge was installed on an open space with wired connection to the sensor node. A 60 W solar panel was set up with two batteries to supply electric power.

At the Wildcat bridge site, a 2.4 GHz Yagi antenna was used on the gateway station while a 2.4 GHz omnidirectional antenna was used on the sensor node. The distance between them was about 18.5 meters. An air-blast cleaning device and a rain

gauge were also installed. A 60 W solar panel was set up with two batteries as the power supply.

At the Silver Creek site, we installed two sensor nodes, one in Silver Creek and the other in Wildcat Creek. The Wildcat Creek sensor node used a 2.4 GHz Yagi antenna pointing to the omnidirectional antenna on the base mote. The distance from the Wildcat Creek to the gateway station was 22 meters. The control and communication module of the Silver Creek sensor node was actually located in the same enclosure as the gateway station, and an original whip antenna on the mote was found sufficient for communication. At the beginning, we used an air compressor to provide air-blast cleaning for each sensor. We later changed the design to clean both sensors using only one air compressor. A rain gauge was installed near the Silver Creek sensor node. A 60 W solar panel was set up with two batteries to provide electric power. Figure 5.27 shows the gateway stations at Fort Riley, Kansas. Figure 5.28 gives the inside view of the gateway station at the Little Kitten site.

Figure 5.27 Gateway stations at Ft. Riley site, KS

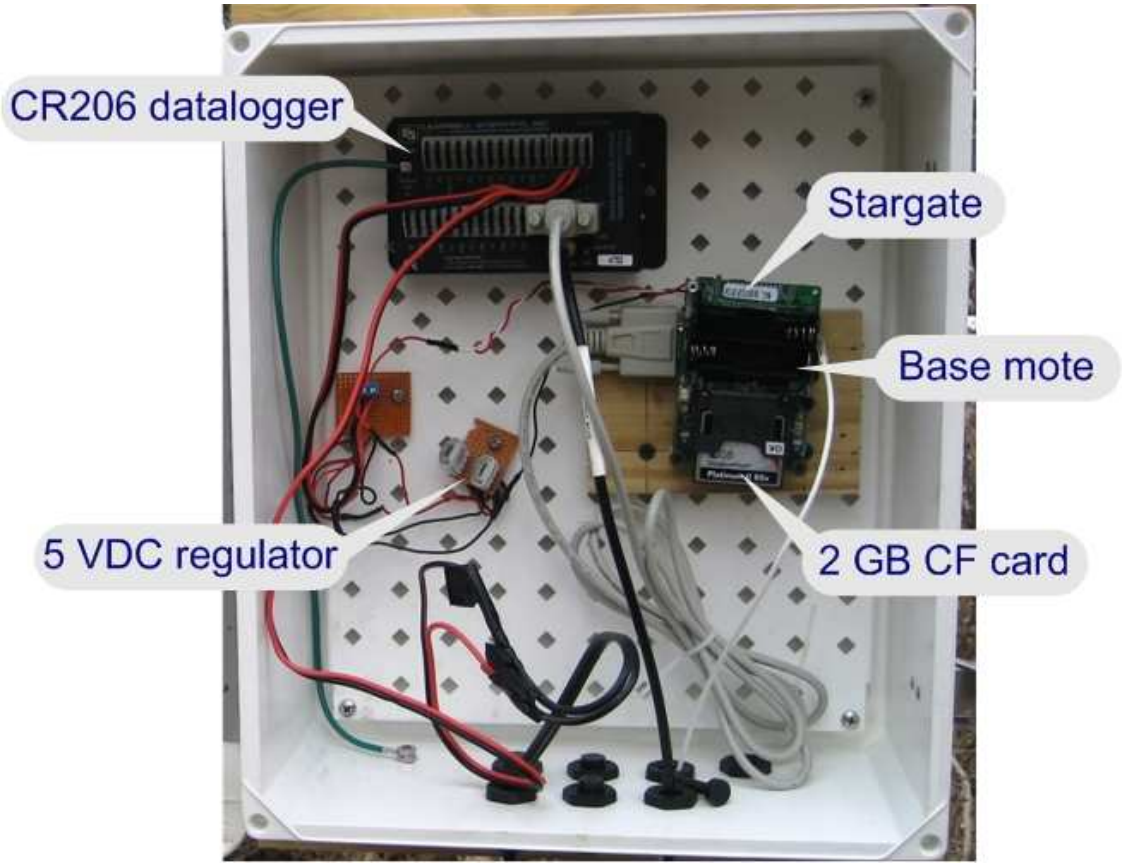


Figure 5.28 Inside view of the gateway station at Little Kitten, Ft. KS.

5.2.6.2.2 Fort Benning site

We installed the WSN at Fort Benning, Georgia, at the end of June, 2009. A central station was located on the Buena Vista Road, 0.86 km from the Upatoi gateway station at longitude 84.748ºW and latitude 32.4452ºN. This location was selected because it had fair wireless coverage by the Alltel Company, which was later merged with the Verizon Wireless Company. Between the Pine Knot gateway station and the central station, we installed a repeater station on the Buena Vista Road, 0.69 km from the central station and 1.01 km from the Pine Knot gateway station, at longitude 84.7407ºW and latitude 32.4455ºN. A map showing these locations is given in Figure 5.29.

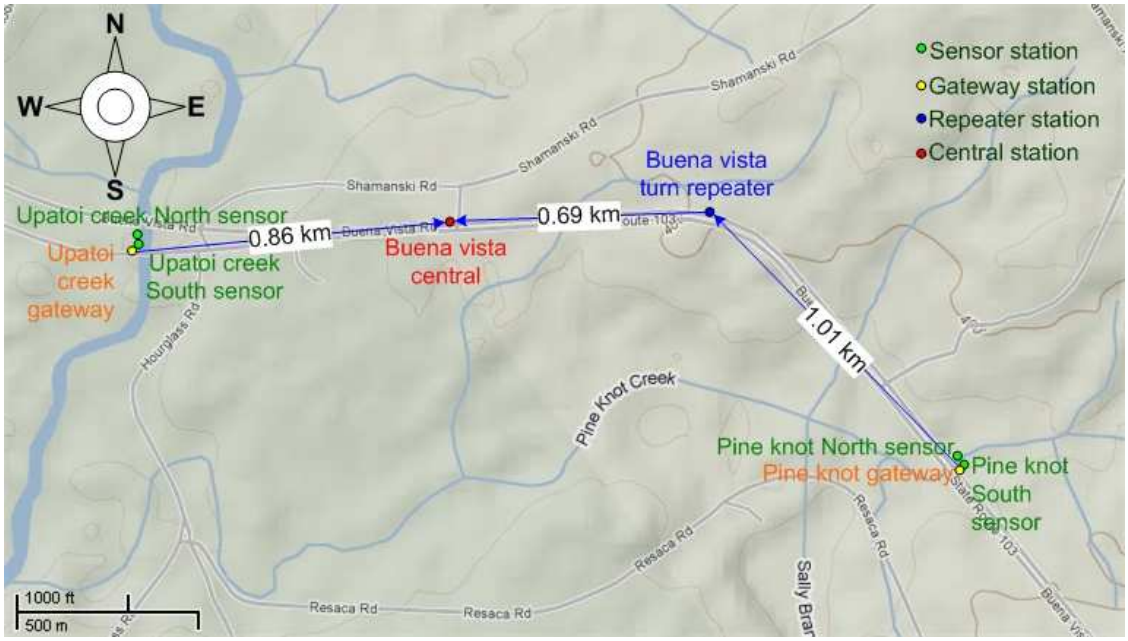


Figure 5.29 Map of the Fort Benning site

The central station had a 900 MHz omnidirectional antenna for communication with both the Upatoi Creek gateway and the repeater station at Buena Vista turn. A fiberglass, dual band cellular antenna (KIT1027, 821-896 MHz/1850-1990 MHz, Laird Technologies, Chesterfield, MO) was also installed for mobile data service. A 20 W solar panel and a battery were used as the power supply (Figure 5.30).

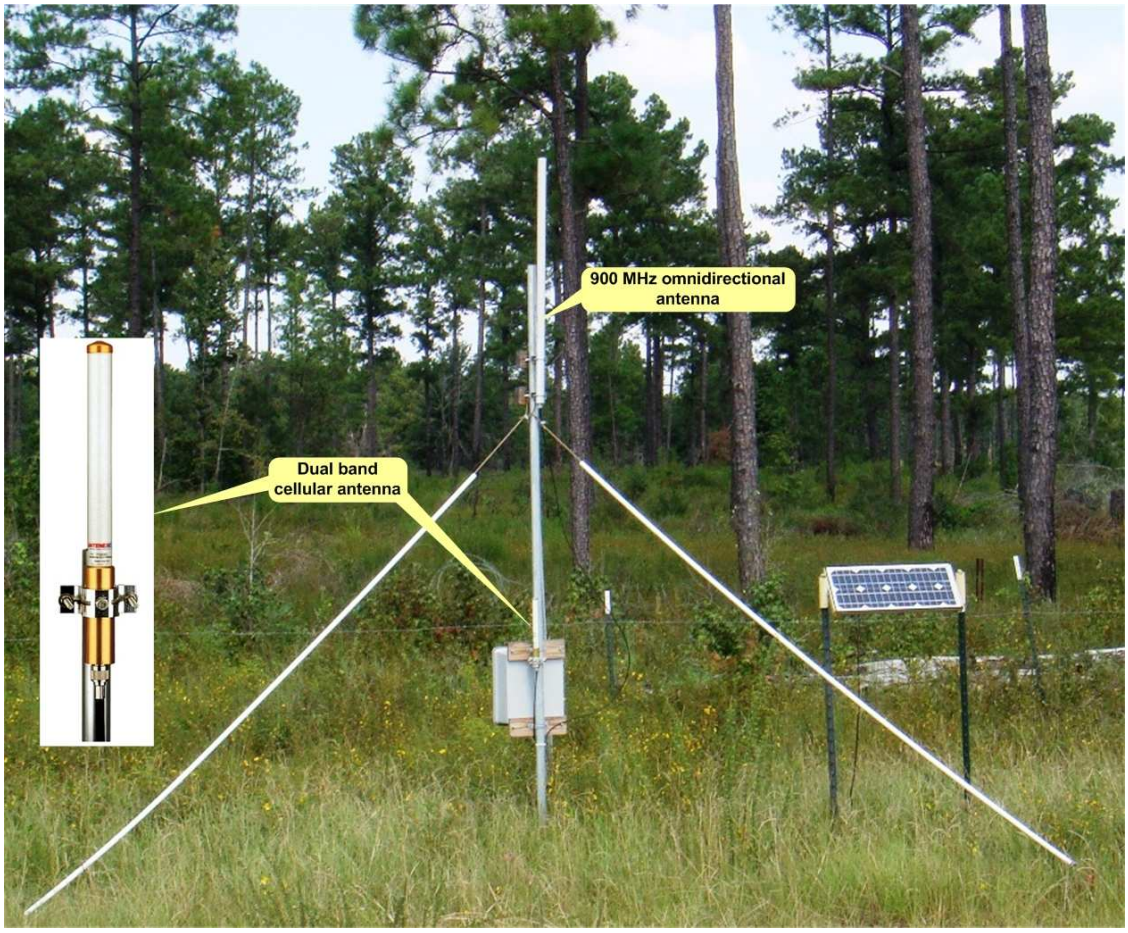


Figure 5.30 Central station at Ft. Benning site, GA

Two local WSNs (LWSNs) were installed, each with two sensor nodes. The first LWSN was located at the Upatoi Creek, under a bridge on the Buena Vista Road, at longitude 84.7572º W and latitude 32.4452º N. Two sensors were installed in the creek; one was more south than the other. Hence, they were named the "Upatoi South" and "Upatoi North" sensor nodes, respectively. The PCB control board, air compressor, and battery for each sensor node were placed in a weather-proof enclosure. Both enclosures were placed on the east bank of the creek. A rain gauge was placed near one of the enclosures. The "Upatoi South" sensor measured both sediment concentration and velocity, whereas the "Upatoi North" sensor only measured sediment. A 5-liter dye canister was used to store blue dye for velocity measurement. 2.4 GHz Yagi antennas on both sensor nodes were directed towards a 2.4 GHz Yagi antenna on the gateway station. The distance between the gateway station and the sensor nodes was about 20 meters. A 900 MHz Yagi antenna was installed on the gateway for MRWN communication. Figure

5.31 shows the gateway station and sensor nodes at the Upatoi Creek site, Fort Benning, GA.

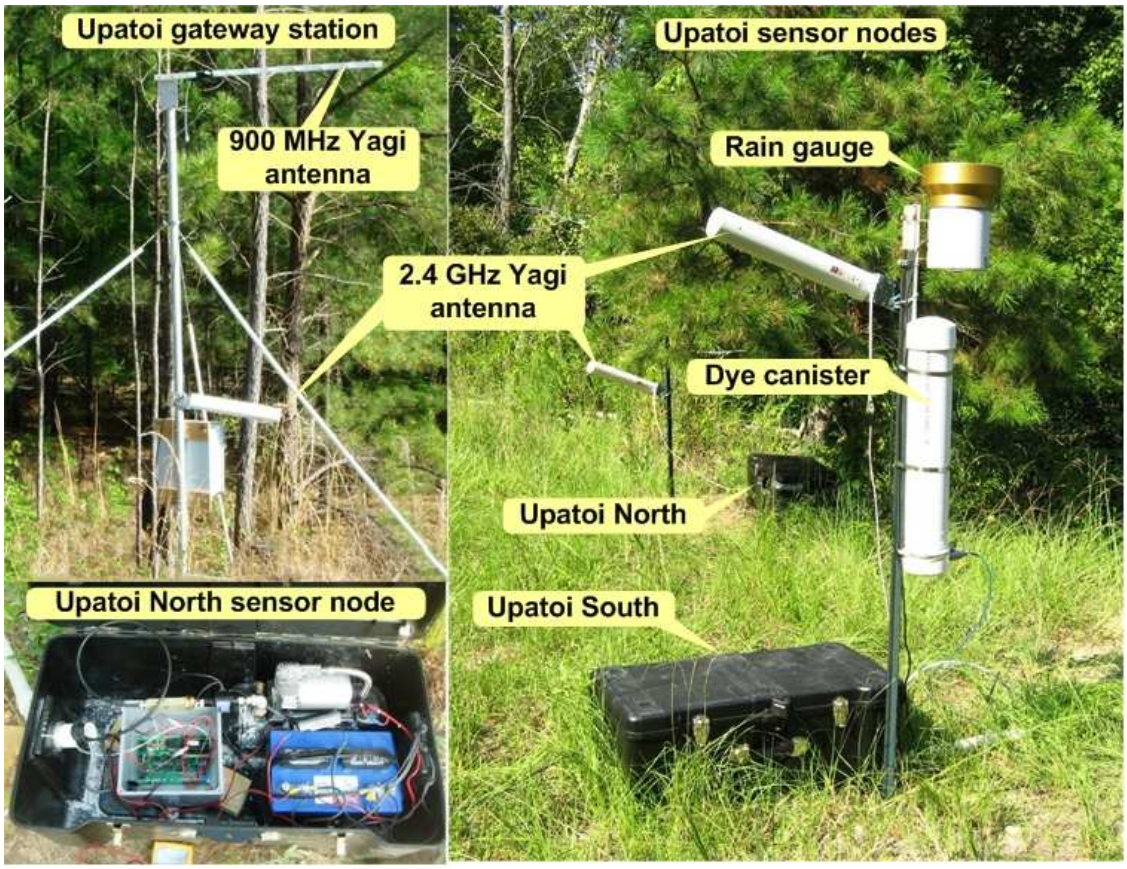


Figure 5.31 Gateway and sensor nodes at Upatoi Creek site, Ft. Benning, GA

The Upatoi South sensor was installed near the east bank of the creek, while the Upatoi North sensor was installed near the center of the creek (Figure 5.32). The sensors were covered with aluminum covers to reduce the influence of ambient light on sensor signals. The shape of the covers for the Upatoi South and Upatoi North was rectangular and diamond, respectively. T-shape fence poles were placed in front of the sensors to prevent damages on the sensor from tree logs and other large debris (Figure 5.33).

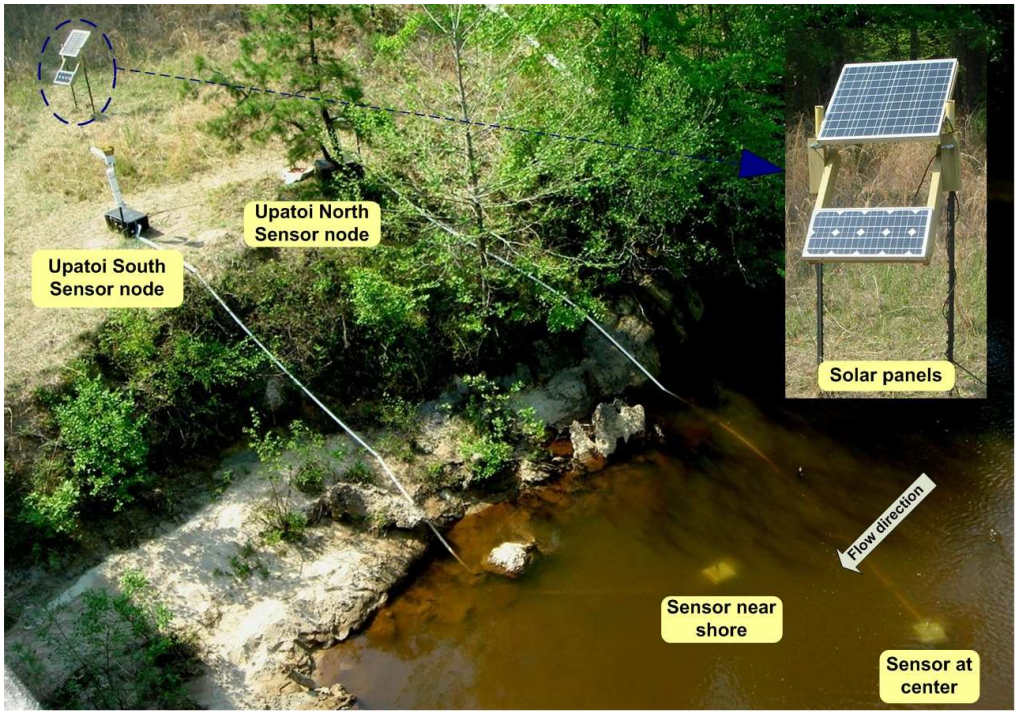


Figure 5.32 Sensor nodes at the Upatoi Creek, Ft. Benning, GA

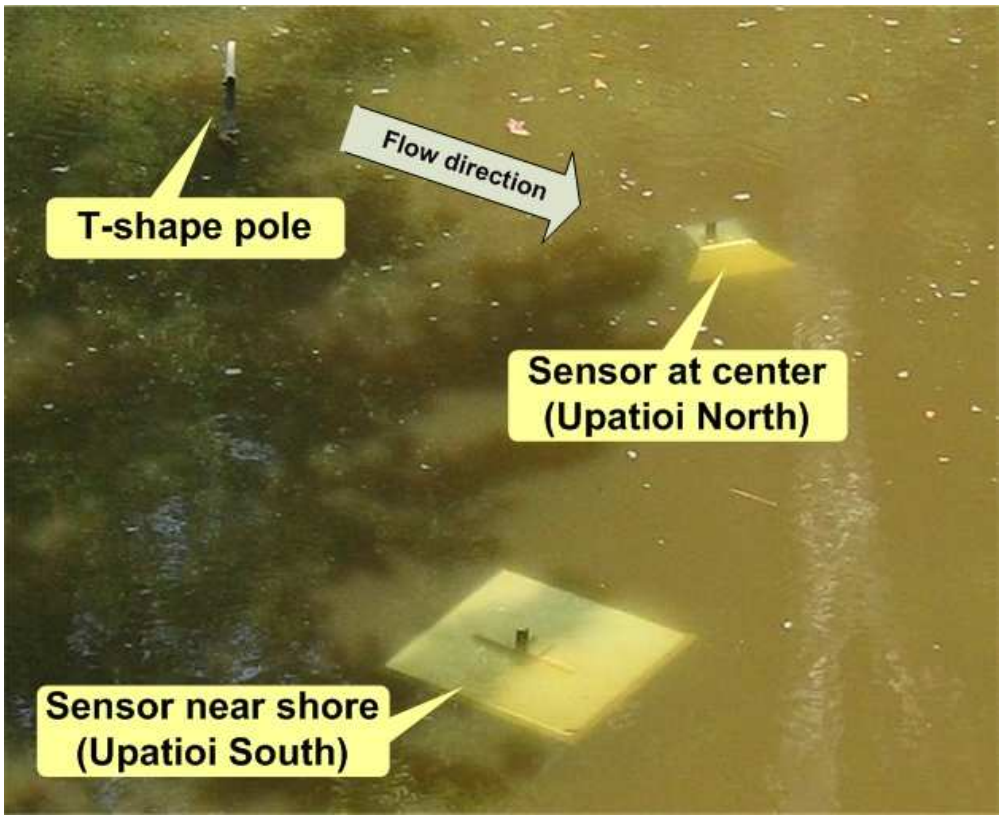


Figure 5.33 Sensor cover for Upatoi Creek site, Ft. Benning, GA

The second LWSN was installed at the Pine Knot Creek, under a bridge on the Buena Vista Road, at longitude 84.733º W and latitude 32.4391ºN. Two sensors were installed on the south and north sides of the creek, respectively. Thus, the sensors were named “Pine Knot South” and “Pine Knot North”, respectively. The gateway station was installed at “Pine Knot South” site. Hence, no antenna was needed for the “Pine Knot South” sensor node. The distance between the gateway station and the North sensor node was about 15 meters; hence a 2.4 GHz Yagi antenna was used at the sensor node. Two solar panels were installed at the Pine Knot gateway station. One was 60 W for the sensory, control, and communication modules; the other was 20 W for the sensor cleaning module. Figure 5.34 shows the gateway station and two sensor nodes at Pine Knot, Fort Benning, GA.

Figure 5.34 Gateway and sensor nodes at Pine Knot site, Ft. Benning, GA

A rectangular shaped cover was used at Pine Knot South and a diamond shape cover was installed at Pine Knot North. The sensor cables were protected inside PVC pipes. Each sensor was protected by a T-shape fence pole at the upstream. The sensors are shown in Figure 5.35.

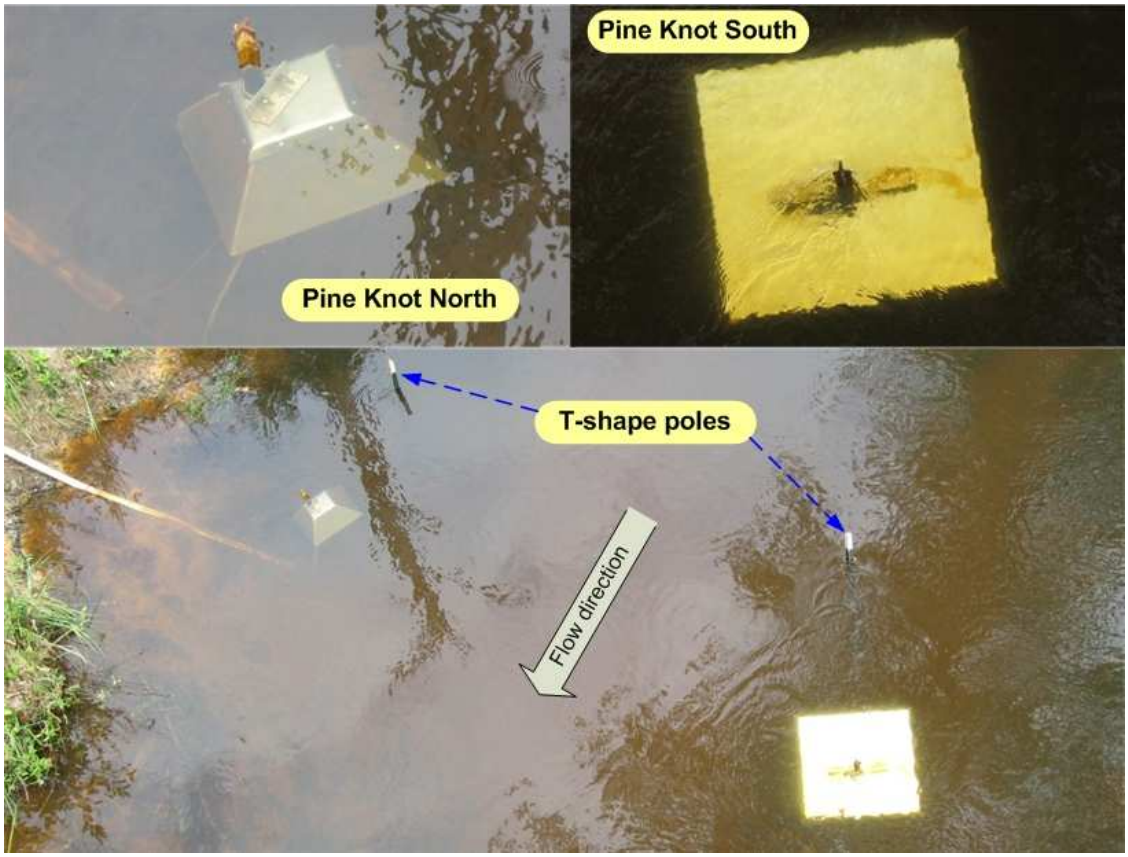


Figure 5.35 Sensor cover at Pine Knot site, Ft. Benning, GA

5.2.6.2.3 Aberdeen proving ground site

The WSN was installed at the Aberdeen proving ground (APG) in October, 2009. The sensors were installed in tidal water. Only sediment concentration was measured because velocity measurement was not needed.

Two locations were selected for the LWSNs. Both were on boat piers, where sensors could be installed in water at proper depth. One LWSN was at the Anita Leight Park on the North, at longitude 76.2745ºW and latitude 39.4507ºN. The central station was placed at the same location. The other LWSN was at the Gunpowder pier on the

South, at longitude 76.3041ºW and latitude 39.3816ºN. Figure 5.36 shows the map for the APG site.

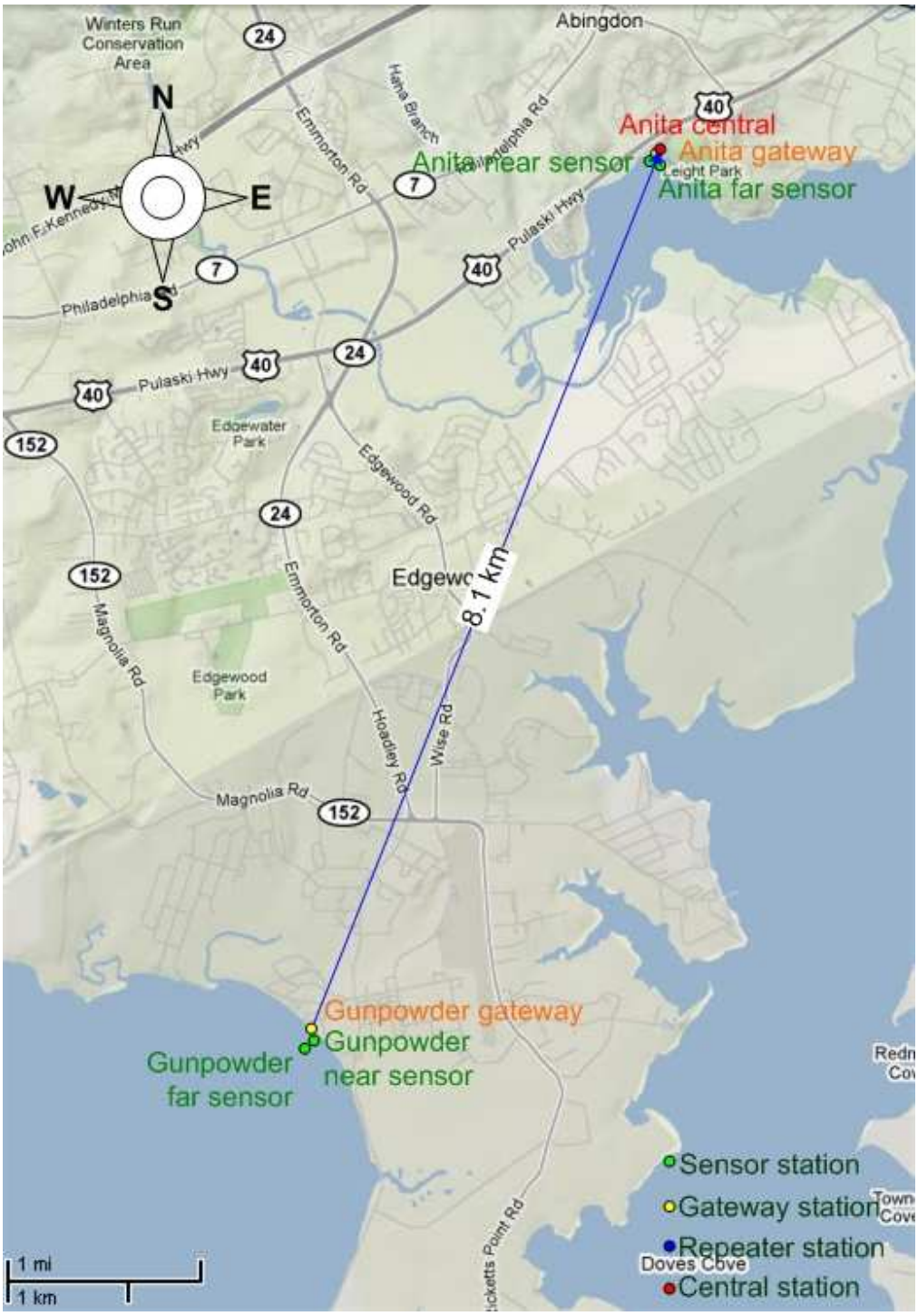


Figure 5.36 Map of the APG site

At each location, two sensors were mounted to the wooden poles of the pier with sunlight covers. Sensors were submerged under water at a depth that kept the sensor in water at low tide. The sensor motes inside its own water-proof enclosures were placed on the piers. Figure 5.37 shows the gateway station and two sensor nodes at the Gunpowder site, APG, MD.

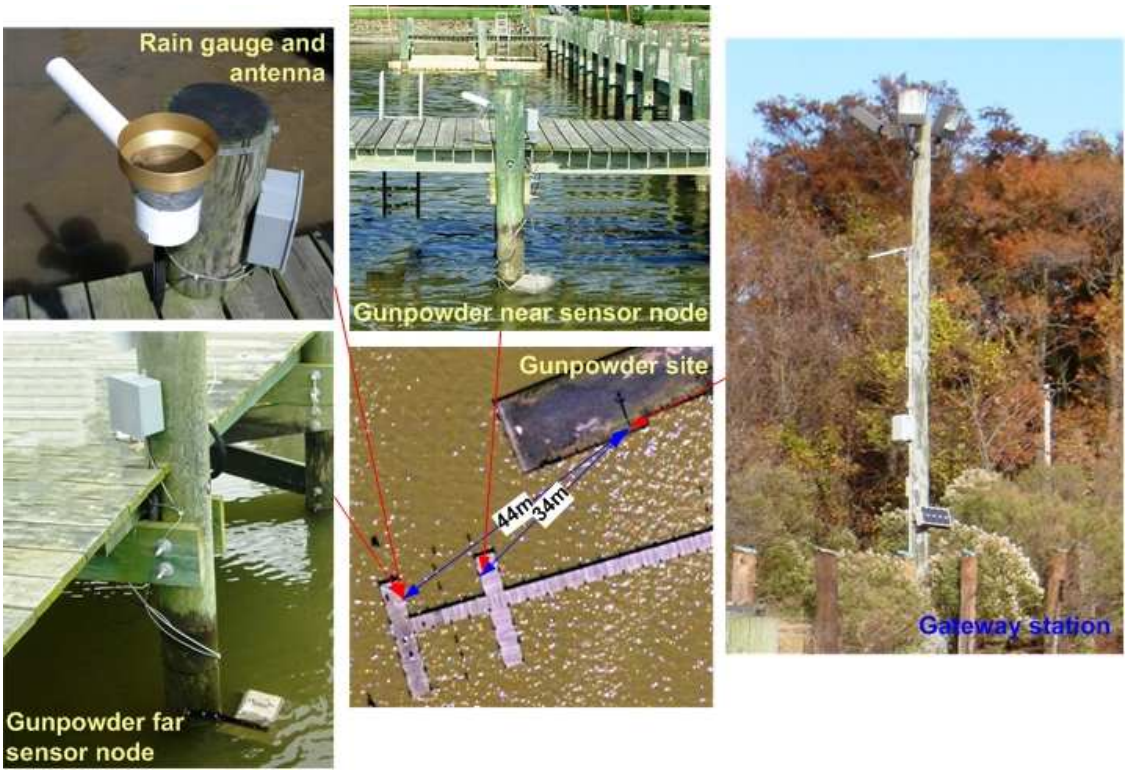


Figure 5.37 Gateway and sensor node at the Gunpowder site, APG, MD

The sensor nodes were named based on their distances to the coast (Figure 5.37). From both "Gunpowder far" and "Gunpowder near" sensor nodes, two 2.4 GHz Yagi antennas were pointing to a 2.4 GHz omnidirectional antenna at the gateway station, at distances of 44 and 34 meters, respectively. A rain gauge was installed at the Gunpowder far sensor node. The sensors were shielded by a diamond shape sunlight cover with a removable roof. This made it easy to manually clean the sensor lenses. Figure 5.38 shows the sunlight cover used in APG, MD.

Figure 5.38 A diamond shape sunlight cover with a removable roof

At the Anita Leight Park site, all components, including the central station, the gateway station and two sensor nodes, were installed within an area of 25 square meters. Hence, external antennas were not necessary for the motes and the datalogger at the gateway station. A 900 MHz Yagi antenna was used for the datalogger at the central station. The two sensor nodes were named based on their distances to the coast as “Anita near” and “Anita far”, respectively. A 60 W solar panel was used for power supply. Figure 5.39 shows the central station, the gateway station and sensor nodes at the Anita Leight site, APG, MD.

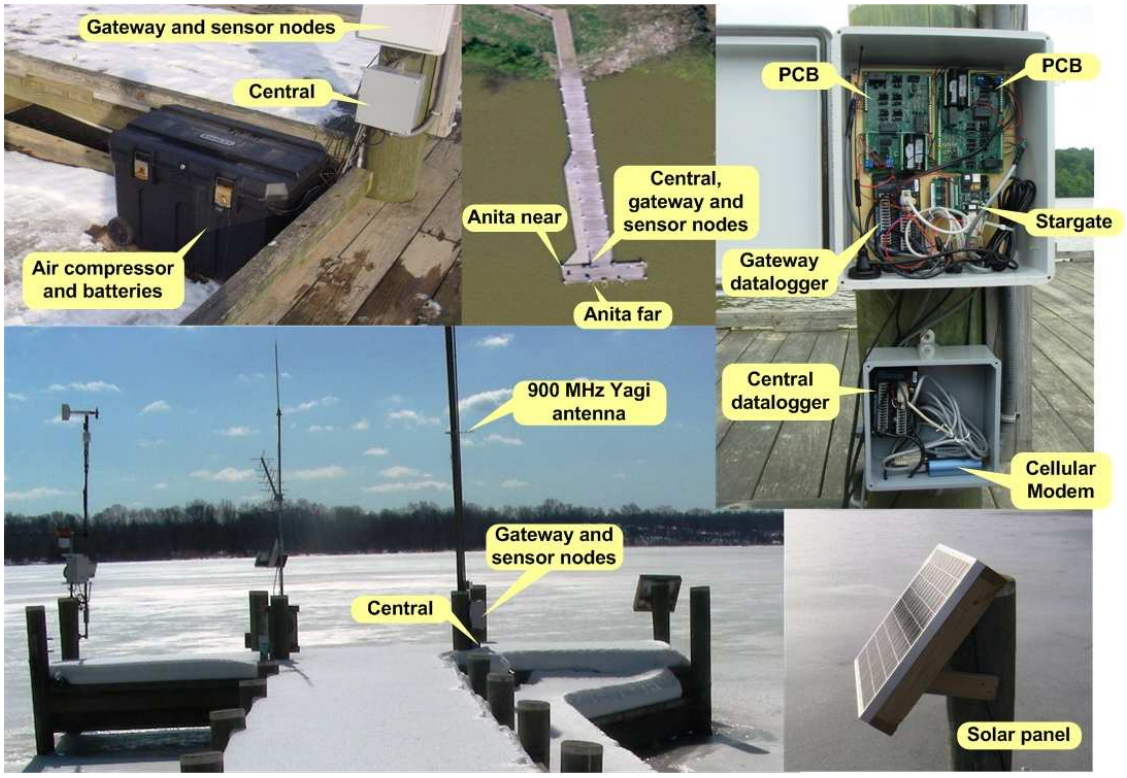


Figure 5.39 Central, gateway and sensor nodes at Anita Leight site, APG, MD

The APG area was much more populated than the experimental sites at Fort Benning and Fort Riley. Location selection for the repeater and central stations was also much more difficult due to the stricter restriction for access within APG. To avoid the need for repeater stations, we raised the heights of the antenna towers at the Gunpowder gateway station and the central station at the Anita Leigh site to about 6 meters and 3 meters, respectively. This allowed direct wireless transmission between the two locations with an effective transmission range of 8 km.

At the APG site, a method of using one air compressor to clean two sensors was tested. Air compressors were the most energy consuming device in the system. Reducing the number of compressors in the system greatly reduced the demand for electric power, thus improving the reliability and extending the life time of the system.

5.3 Results and discussion

Results of the experiments conducted at three military installations are reported in this section. Analysis and discussion are also given.

5.3.1 Packet loss and transmission error

This section reports the packet loss and transmission error rate for each sensor node under normal operation. Transmission breakages of longer than two hours, such as restart of a server, maintenance break at a sensor node, gateway, repeater or central station, breakage from the cellular service, and power outage, were considered abnormal operations. Packet loss and transmission error during abnormal operations were not included in the statistics. In addition, duplicate data transmission intentionally conducted for testing purpose is not included in the reports.

The average packet loss rate and transmission error rate for Little Kitten Creek sensor node were 5.29% and 0.55% respectively. For this sensor node, data were transmitted to the central station via a dedicated repeater station at Cico tank. Figure 5.40 shows the monthly average packet loss rate and transmission error rate for sensor node at Little Kitten Creek, Manhattan, KS.

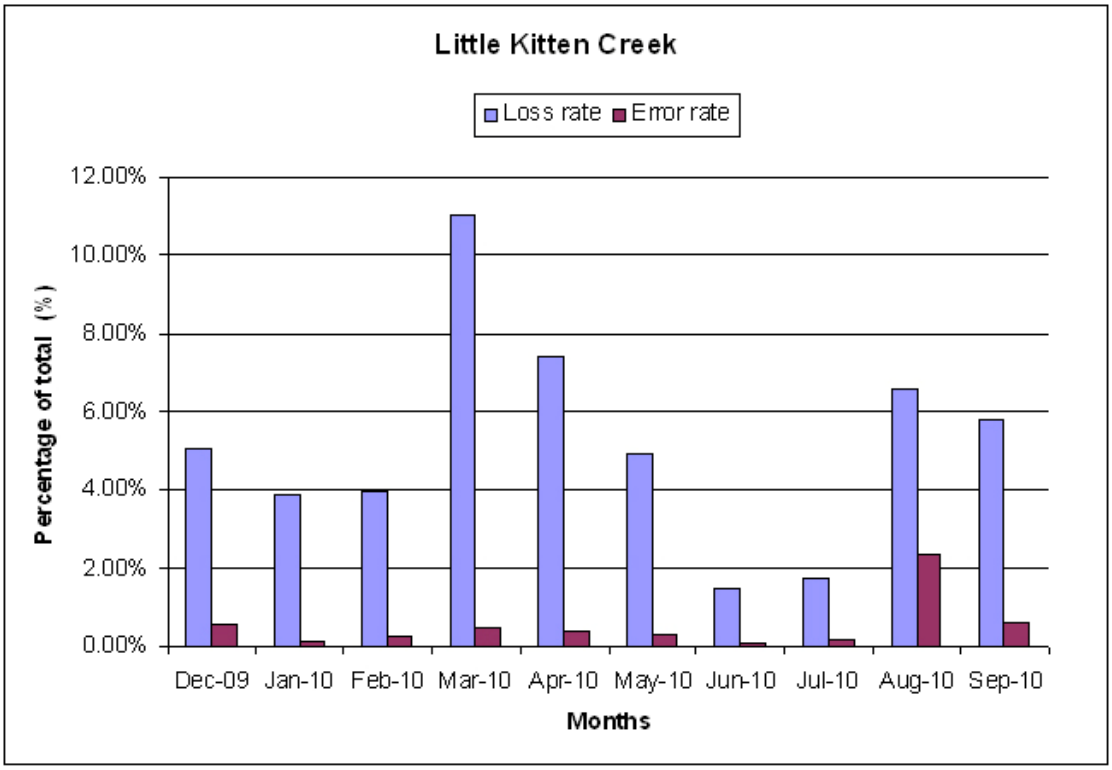


Figure 5.40 Monthly average packet loss rate and transmission error rate for Little Kitten Creek, Manhattan, KS sensor node

Performance of the three-tier WSN in transmitting data from the Little Kitten sensor node and causes for failures within a ten-month period are illustrated in Figure 5.41.

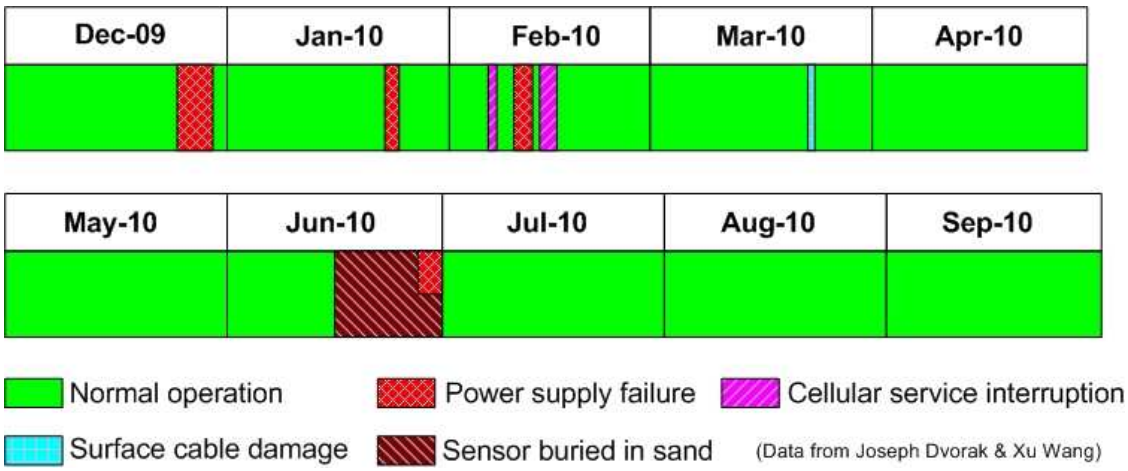


Figure 5.41 Performance in wireless data transmission and causes for failures for Little Kitten sensor node

Because Little Kitten Creek was the closest site to KSU, we used this site for a series of tests to improve the quality of data transmission. However, this site was in an area with dense coverage of trees, bushes, and grasses, which seriously affected mote communication. In order to achieve a low packet loss rate, we removed some of the vegetative cover to create the “line-of- sight” condition for communication within the LWSN. For MRWN communication, a tree near the gateway station blocked the radio wave path between the Little Kitten site and the repeater station at Cico tank. This was probably the main cause for the relatively high packet loss rate and transmission error rate.

The average packet loss rate and transmission error rate for Wildcat Bridge sensor node were 5.78% and 5.03% respectively. For this sensor node, data were transmitted to the central station via a repeater station at Above Keats. Figure 5.42 shows the monthly average packet loss rate and transmission error rate for the sensor node at Wildcat Bridge, Ft. Riley, KS.

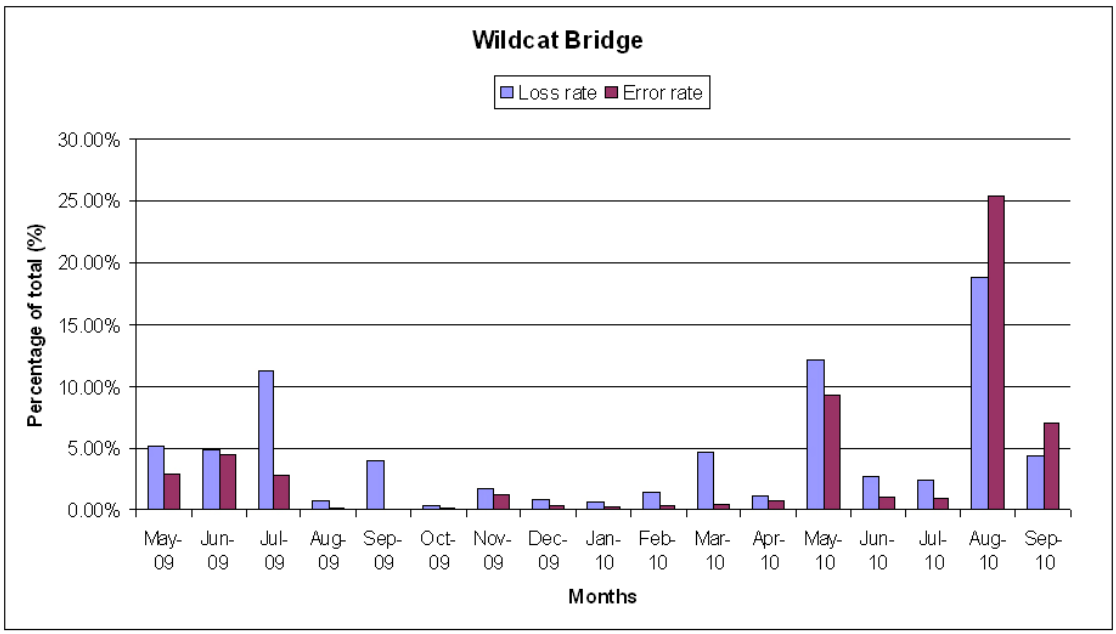


Figure 5.42 Monthly average packet loss rate and transmission error rate for Wildcat Bridge, Ft. Riley, KS sensor node

Performance of the three-tier WSN in transmitting data from the Wildcat bridge sensor node and causes for failures within a ten-month period are illustrated in Figure 5.43.

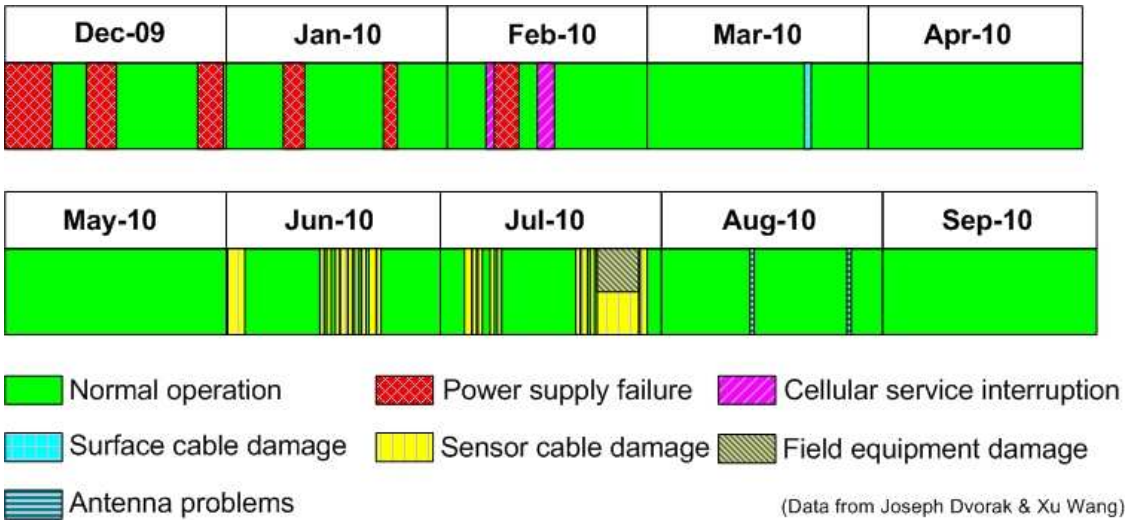


Figure 5.43 Performance in wireless data transmission and causes for failures for Wildcat bridge sensor node

Within a one year period, from May 2009 to Apr. 2010, the Wildcat Bridge site had very low packet loss and transmission error rates. The data transmission quality

depended on the performance of the repeater station at “Above Keats”. From Figures 5.42, 5.44 and 5.46, it can be seen that higher packet loss rates were found for sensors at both the Wildcat Bridge and Silver Creek sites during the same time period. Because these sensor nodes shared the same repeater station, it can be concluded that the function of the repeater station was deteriorated during this period of time. A site inspection found that the antenna tower at the “Above Keats” repeater station fell down.

The average packet loss rate and transmission error rate for Silver Creek sensor node were 19.12% and 13.27% respectively. For this sensor node, data were transmitted to the central station via a repeater station at Above Keats. Figure 5.44 shows the monthly average packet loss rate and transmission error rate for sensor node at Silver Creek, Ft. Riley, KS.

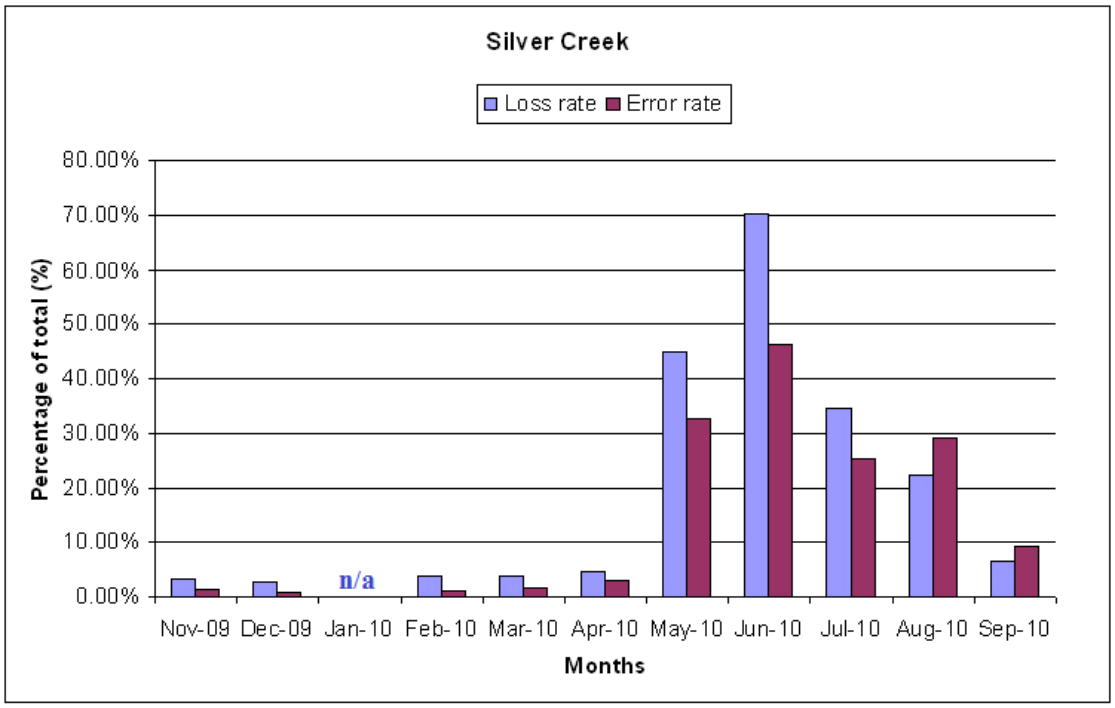


Figure 5.44 Monthly average packet loss rate and transmission error rate for Silver Creek, Ft. Riley, KS sensor node

Performance of the three-tier WSN in transmitting data from the Silver Creek sensor node and causes for failures within a ten-month period are illustrated in Figure 5.45.

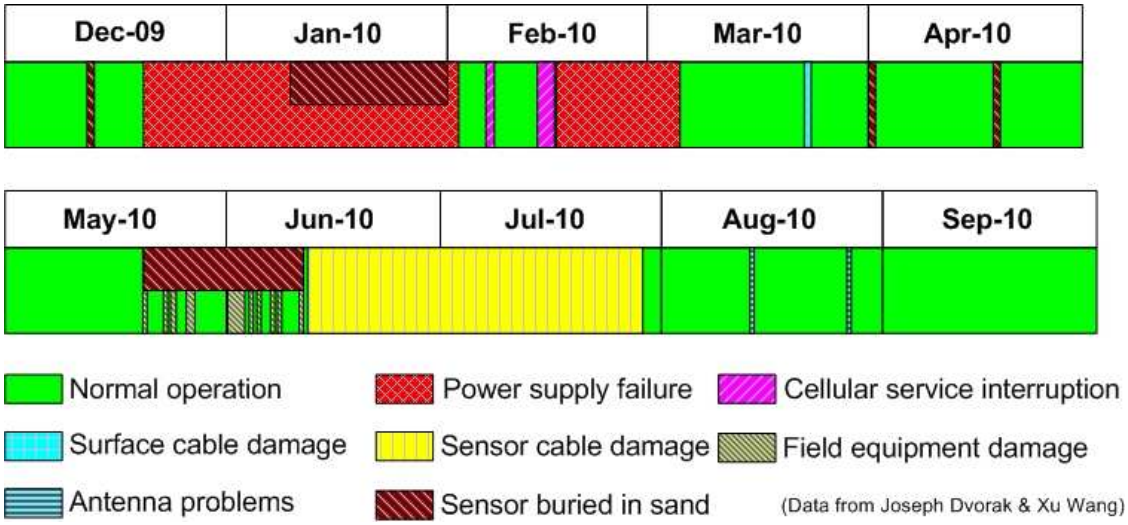


Figure 5.45 Performance in wireless data transmission and causes for failures for Silver Creek sensor node

The Silver Creek sensor node was located in a deep woody area with dense plants and trees. From Nov. 2009 to Apr. 2010, we had low packet loss rates and transmission error rate. However, from the end of Dec. 2009 to the beginning of Feb. 2010, we did not receive any data due to power failure, and very high packet loss rates and transmission error rates have been observed since May 2010. Sensor cable damage and field equipment damage may have been responsible for this.

The average packet loss rate and transmission error rate for Wildcat Creek sensor node were 17.34% and 11.90% respectively. For this sensor node, data were transmitted to the central station via a repeater station at Above Keats. Figure 5.46 shows the monthly average packet loss rate and transmission error rate for sensor node at Wildcat Creek, Ft. Riley, KS.

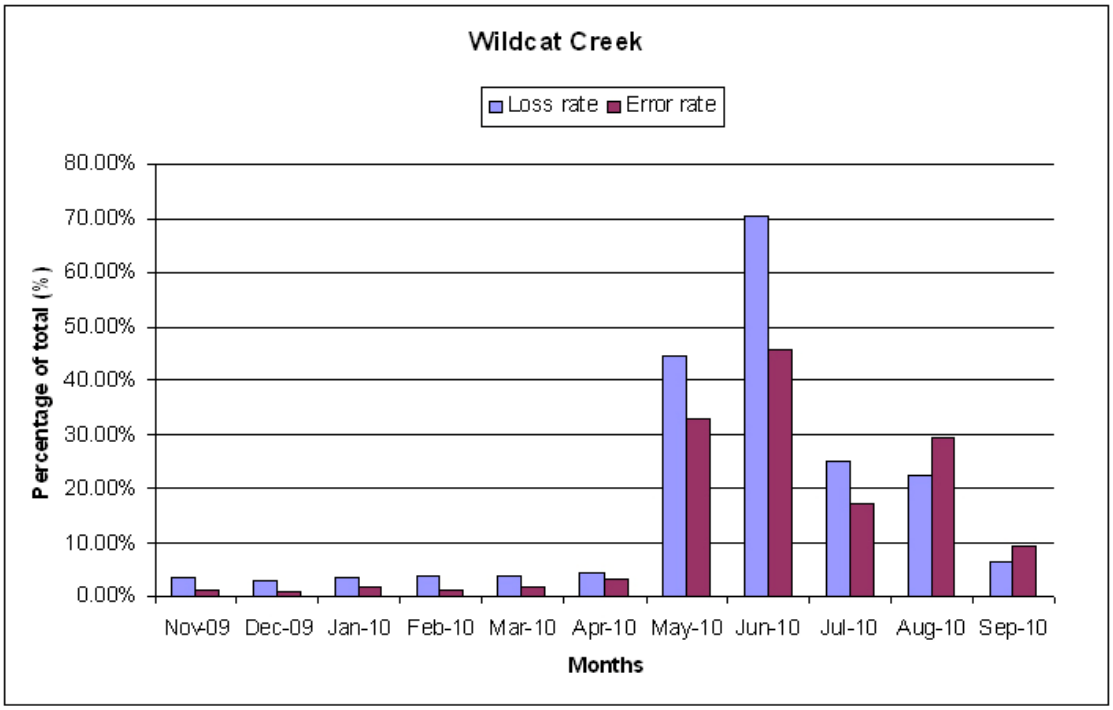


Figure 5.46 Monthly average packet loss rate and transmission error rate for Wildcat Creek, Ft. Riley, KS sensor node

Performance of the three-tier WSN in transmitting data from the Wildcat Creek sensor node and causes for failures within a ten-month period are illustrated in Figure 5.47.

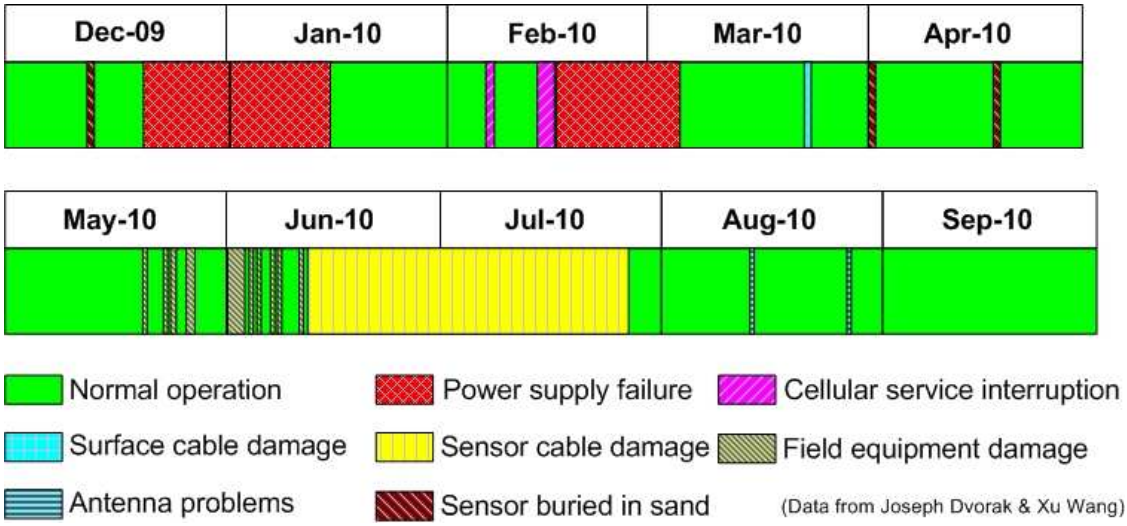


Figure 5.47 Performance in wireless data transmission and causes for failures for Wildcat Creek sensor node

The Wildcat Creek sensor was located very close to the Silver Creek sensor, and both sensors shared the same gateway station and repeater station. The transmission performances for these two sensor nodes were therefore similar.

The average packet loss rate and transmission error rate for Upatoi South sensor node were 0.44% and 0.03% respectively. Figure 5.48 shows the monthly average packet loss rate and transmission error rate for sensor node at Upatoi South, Ft. Benning, GA.

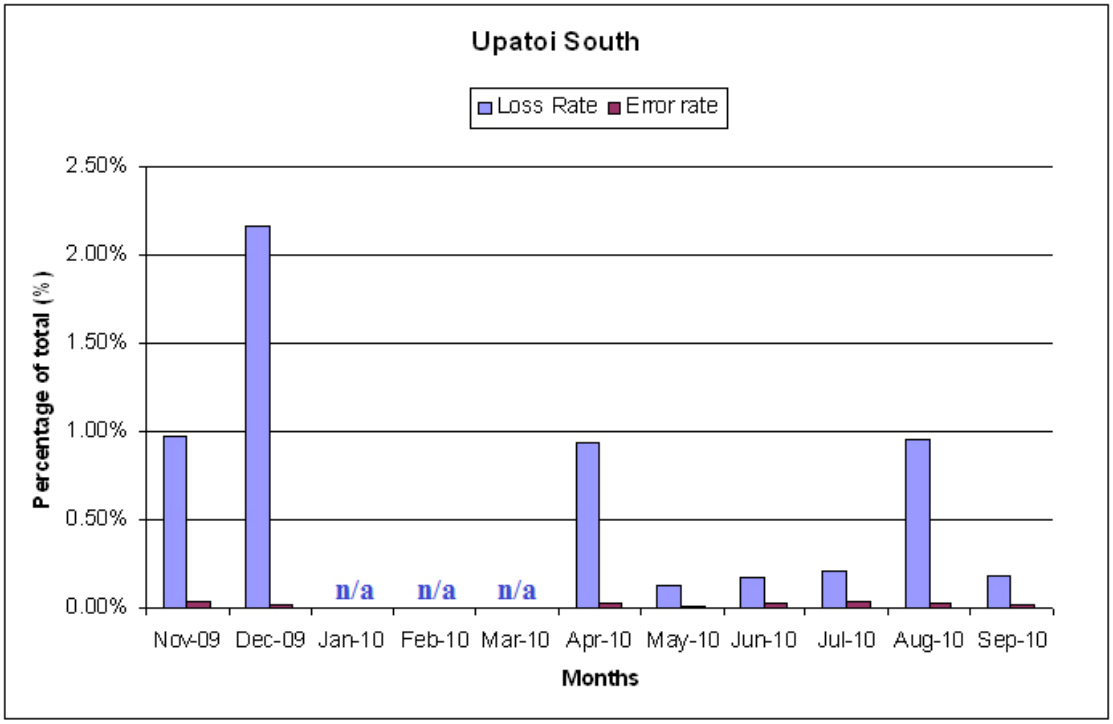


Figure 5.48 Monthly average packet loss rate and transmission error rate for Upatoi South, Ft. Benning, GA sensor node

Performance of the three-tier WSN in transmitting data from the Upatoi South sensor node and causes for failures within an eleven-month period are illustrated in Figure 5.49.

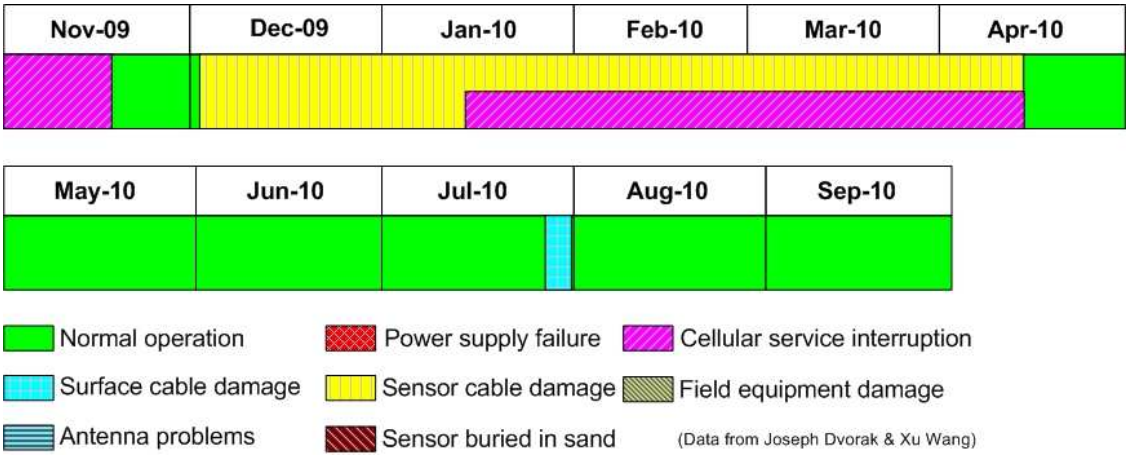


Figure 5.49 Performance in wireless data transmission and causes for failures for Upatoi South sensor node

Both the Upatoi South and North sensor nodes were located in an area with open space. These two sensor nodes were close to each other. The gateway station for Upatoi Creek was located on the bridge with higher elevation than both Upatoi South and North sensor nodes. As a result, the average packet loss rate and transmission error rate were very low. However, sensor cable damage and cellular data service interruption caused data transmission failures from early Dec, 2009 to middle of Apr. 2010.

The average packet loss rate and transmission error rate for Upatoi North sensor node were 2.51% and 0.03% respectively. Figure 5.50 shows the monthly average packet loss rate and transmission error rate for sensor node at Upatoi South, Ft. Benning, GA.

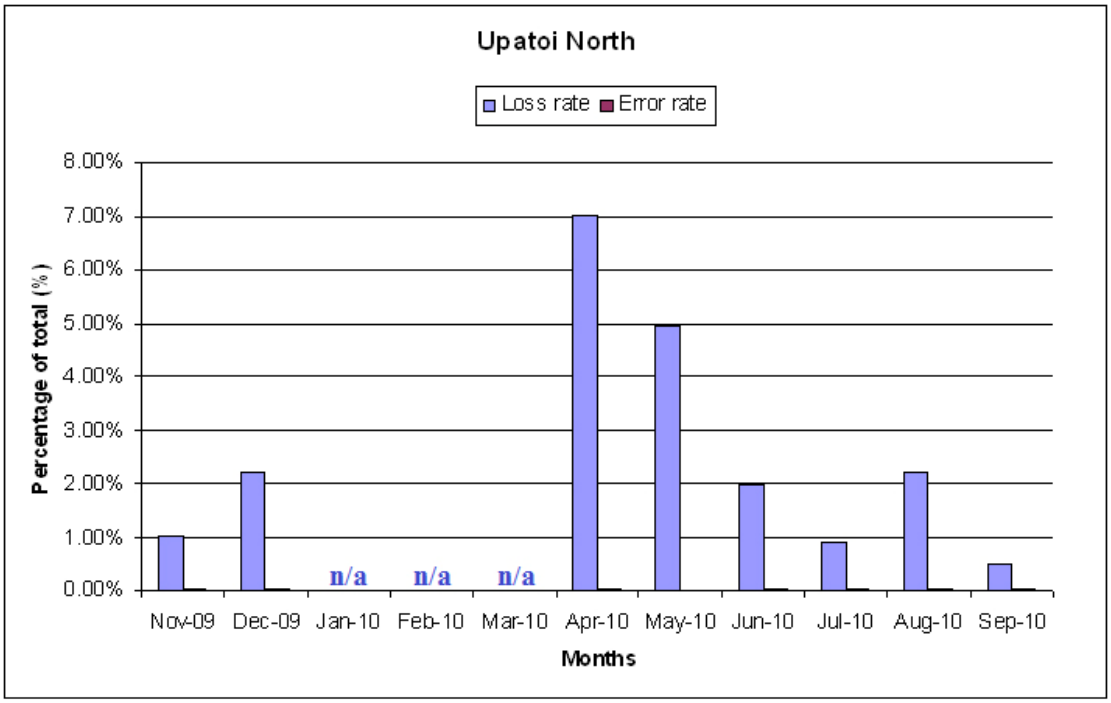


Figure 5.50 Monthly average packet loss rate and transmission error rate for Upatoi North, Ft. Benning, GA sensor node

Performance of the three-tier WSN in transmitting data from the Upatoi North sensor node and causes for failures within an eleven-month period are illustrated in Figure 5.51.

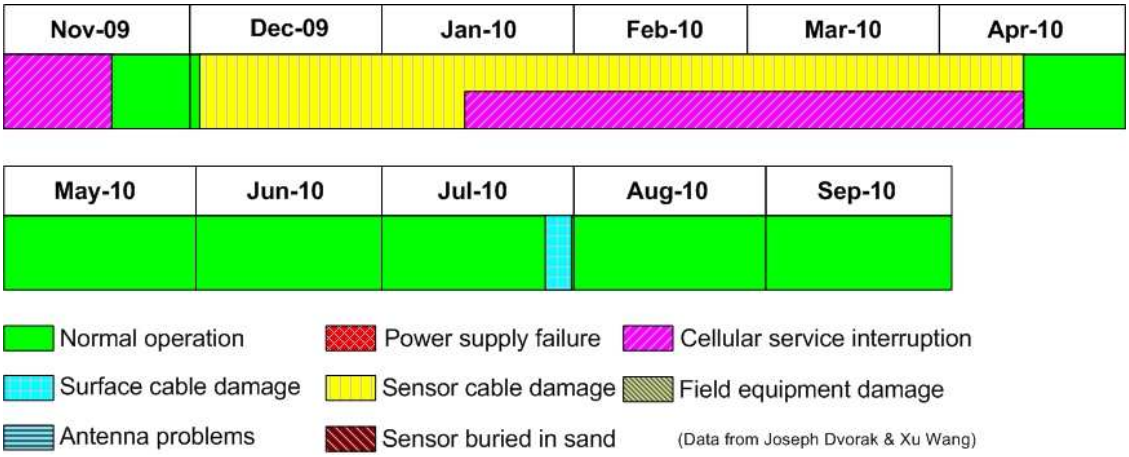


Figure 5.51 Performance in wireless data transmission and causes for failures for Upatoi North sensor node

The average packet loss rate and transmission error rate for Pine Knot South sensor node were 1.78% and 0.07% respectively. For this sensor node, data were

transmitted to the central station via a repeater station at Buena Vista Turn. Figure 5.52 shows the monthly average packet loss rate and transmission error rate for sensor node at Pine Knot South, Ft. Benning, GA.

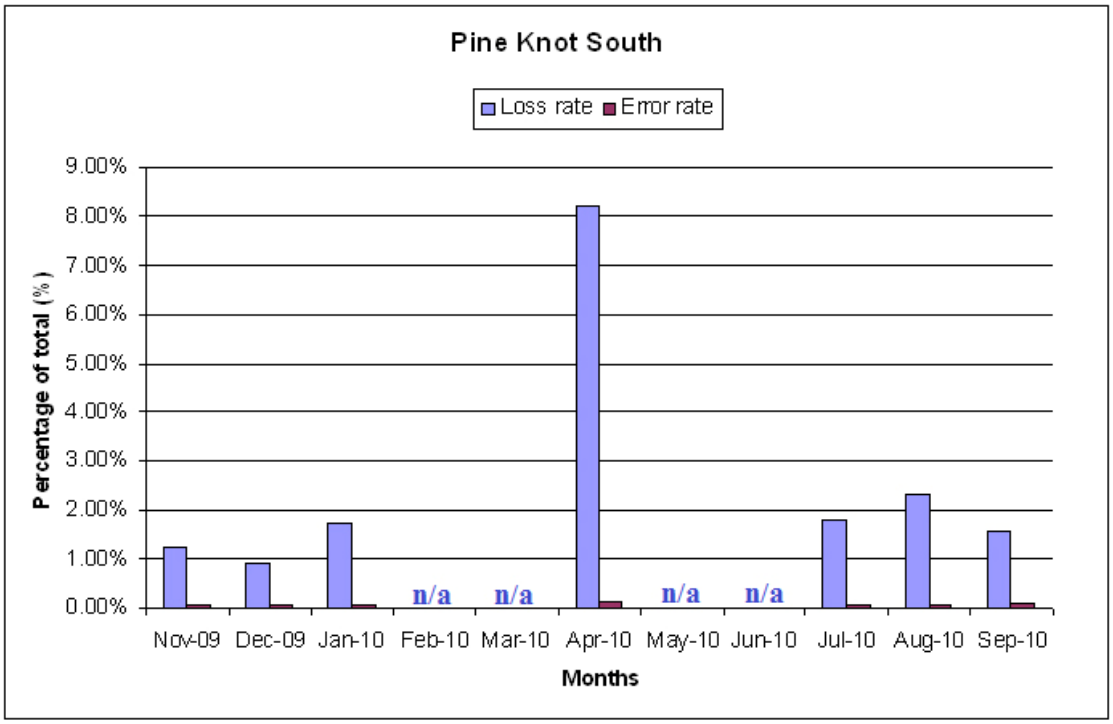


Figure 5.52 Monthly average packet loss rate and transmission error rate for Pine Knot South, Ft. Benning, GA sensor node

Performance of the three-tier WSN in transmitting data from the Pine Knot South sensor node and causes for failures within an eleven-month period are illustrated in Figure 5.53.

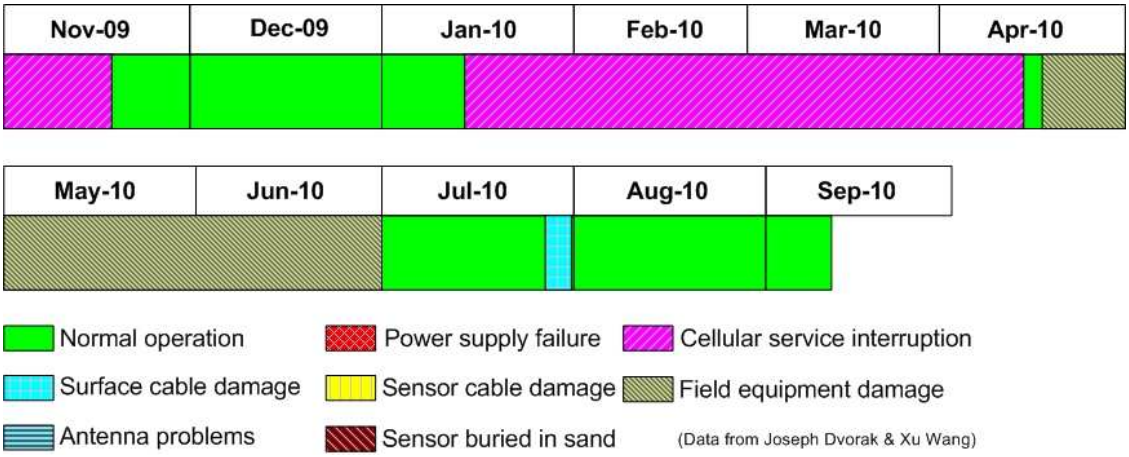


Figure 5.53 Performance in wireless data transmission and causes for failures for Pine Knot South sensor node

Pine Knot South and North sensor nodes were close to each other, and they were near the gateway station. As a result, the data communication quality for both sensor nodes was generally good. However, we completely lost data from Jan. 2010 to Apr. 2010 due to interruption in cellular data service from the carrier (Alltel). Shortly after the data service was resumed in April, an equipment failure that resulted in power loss prevented normal signal transmission until it was repaired in late June, 2010.

The average packet loss rate and transmission error rate for Pine Knot North sensor node were 1.42% and 0.08% respectively. For this sensor node, data were transmitted to the central station via a repeater station at Buena Vista Turn. Figure 5.54 shows the monthly average packet loss rate and transmission error rate for sensor node at Pine Knot North, Ft. Benning, GA.

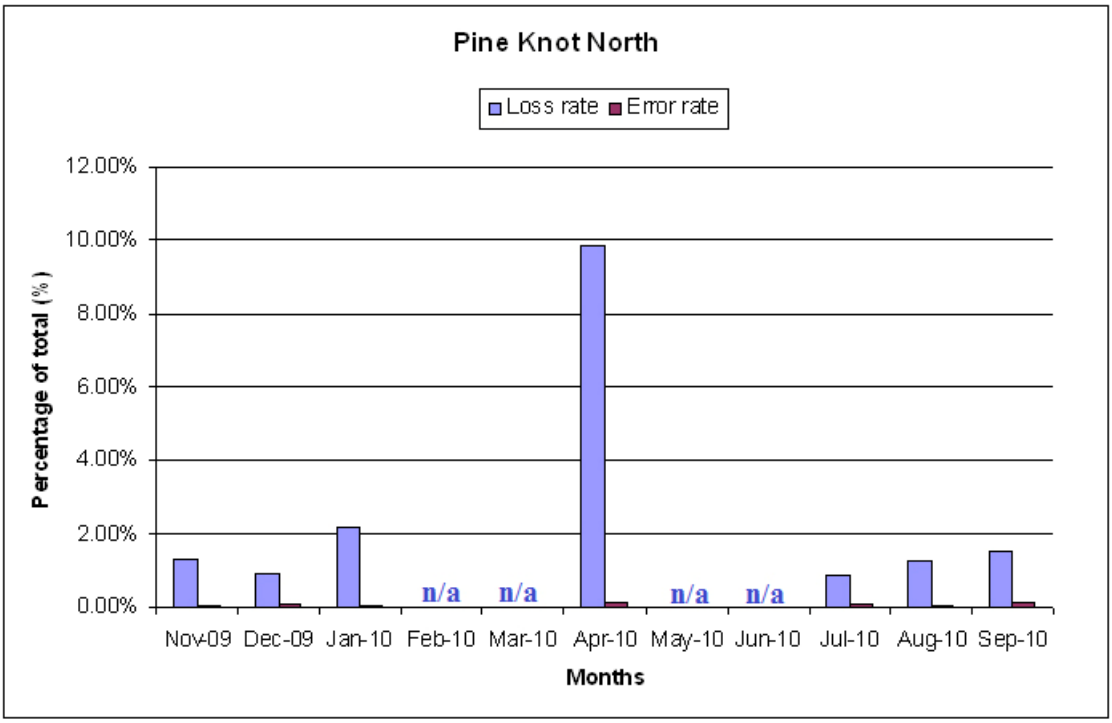


Figure 5.54 Monthly average packet loss rate and transmission error rate for Pine Knot North, Ft. Benning, GA sensor node

Performance of the three-tier WSN in transmitting data from the Pine Knot North sensor node and causes for failures within an eleven-month period are illustrated in Figure 5.55.

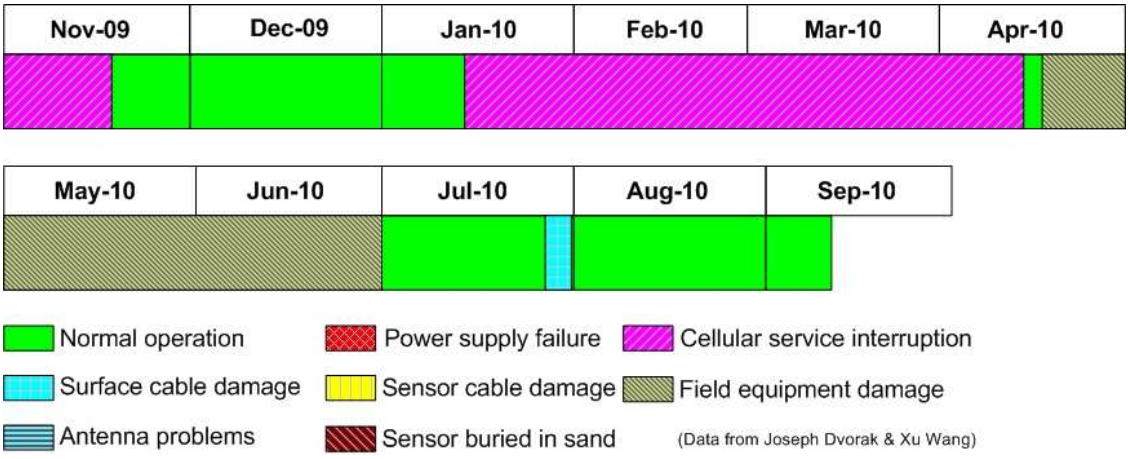


Figure 5.55 Performance in wireless data transmission and causes for failures for Pine Knot North sensor node

The average packet loss rate and transmission error rate for Gunpowder Near sensor node were 30.20% and 8.79% respectively. Figure 5.56 shows the monthly

average packet loss rate and transmission error rate for sensor node at Gunpowder Near, APG, MD.

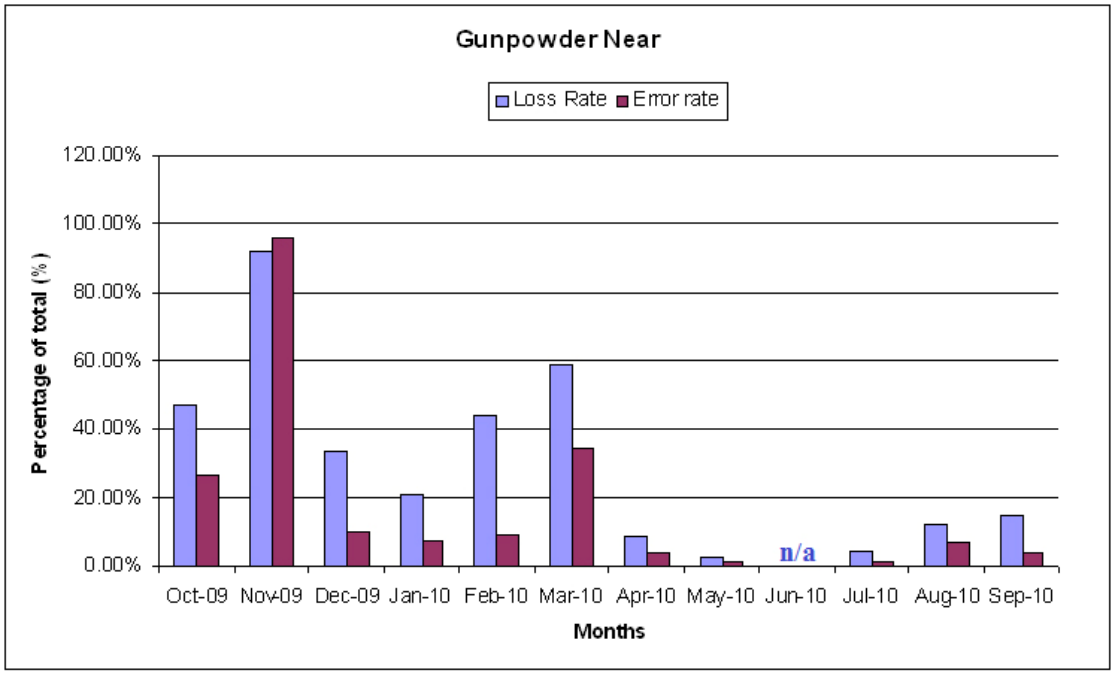


Figure 5.56 Monthly average packet loss rate and transmission error rate for Gunpowder Near, APG, MD sensor node

Performance of the three-tier WSN in transmitting data from the Gunpowder Near sensor node and causes for failures within a twelve-month period are illustrated in Figure 5.57.

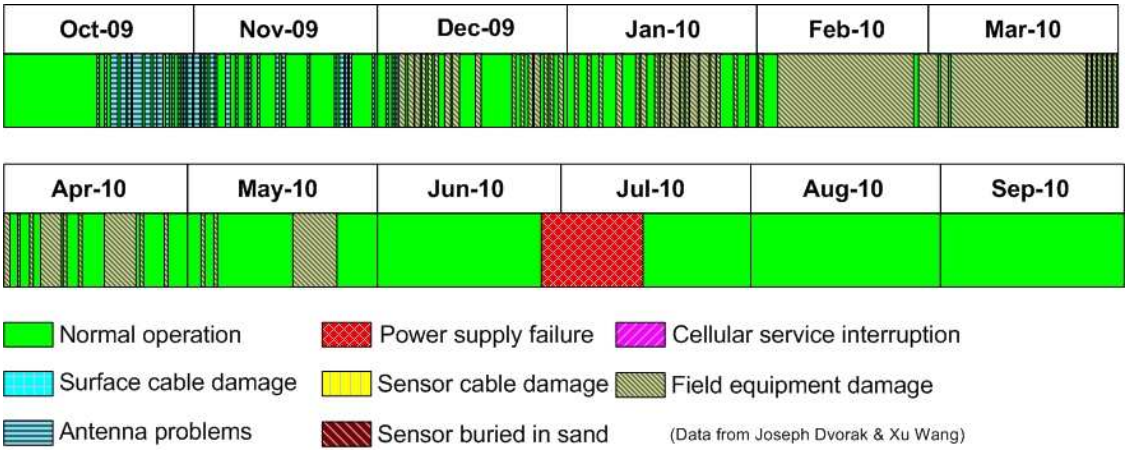


Figure 5.57 Performance in wireless data transmission and causes for failures for Gunpowder Near sensor node

Gunpowder Near and Far sensor nodes were close to each other. They shared the same gateway station. We did not use any repeater station between the Gunpowder gateway and the central station at the Anita Leigh site. In order to transmit data through a distance of 8.1 km, we had to raise the antenna towers at both sites. From Oct. 2009 to Mar. 2010, the packet loss rates and transmission error rates were extremely high. This was because that the clock on the gateway station was not functioning properly yet the data transfer protocol required checking of the time stamps. The problem was solved by temporarily removing the time stamp checking from the program. As a result, the data transmission was greatly improved.

The average packet loss rate and transmission error rate for Gunpowder Far sensor node were 22.61% and 11.53% respectively. Figure 5.58 shows the monthly average packet loss rate and transmission error rate for sensor node at Gunpowder Far, APG, MD.

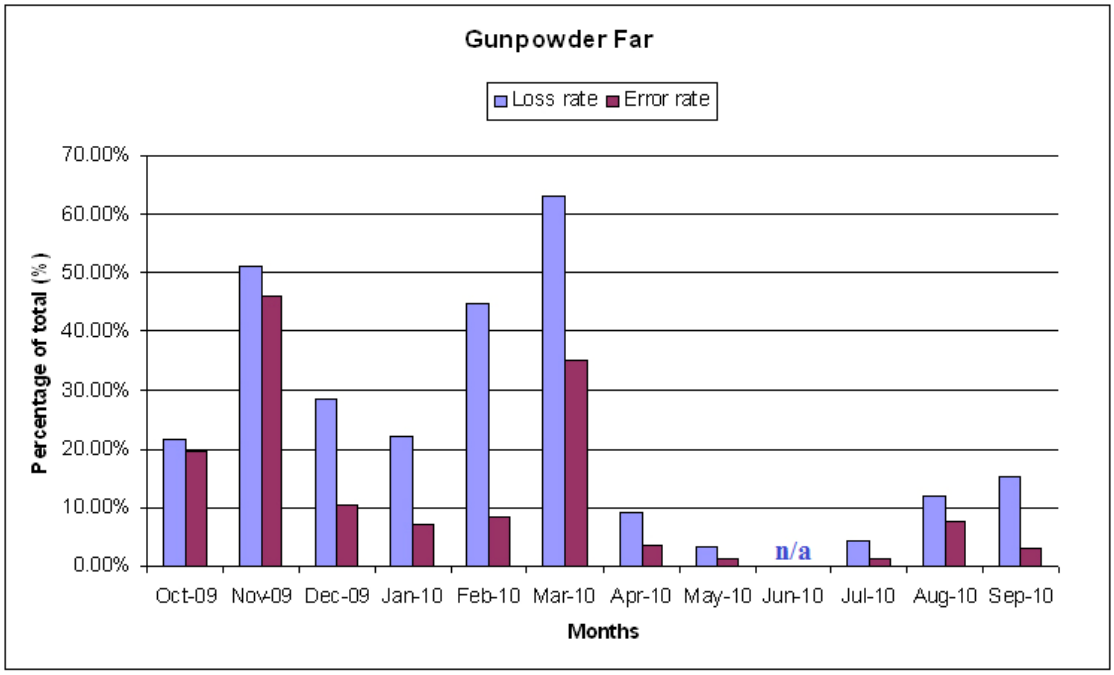


Figure 5.58 Monthly average packet loss rate and transmission error rate for Gunpowder Far, APG, MD sensor node

Performance of the three-tier WSN in transmitting data from the Gunpowder Far sensor node and causes for failures within a twelve-month period are illustrated in Figure 5.59.

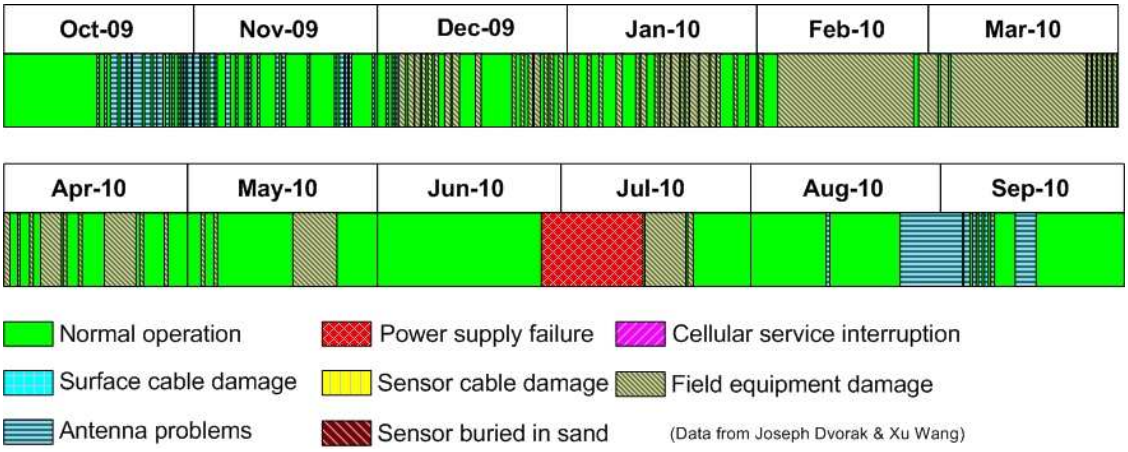


Figure 5.59 Performance in wireless data transmission and causes for failures for Gunpowder Far sensor node

The average packet loss rate and transmission error rate for Anita Near sensor node were 2.33% and 0.79% respectively. Figure 5.60 shows the monthly average packet loss rate and transmission error rate for sensor node at Anita Near, APG, MD.

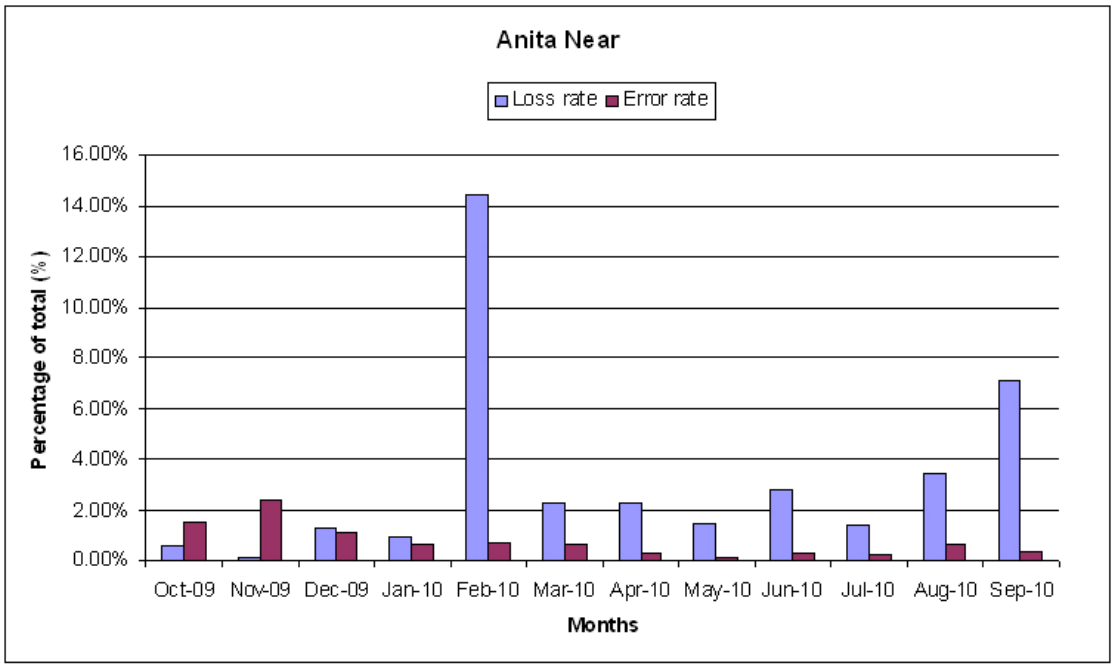


Figure 5.60 Monthly average packet loss rate and transmission error rate for Anita Near, APG, MD sensor node

Performance of the three-tier WSN in transmitting data from the Anita Near sensor node and causes for failures within a twelve-month period are illustrated in Figure 5.61.

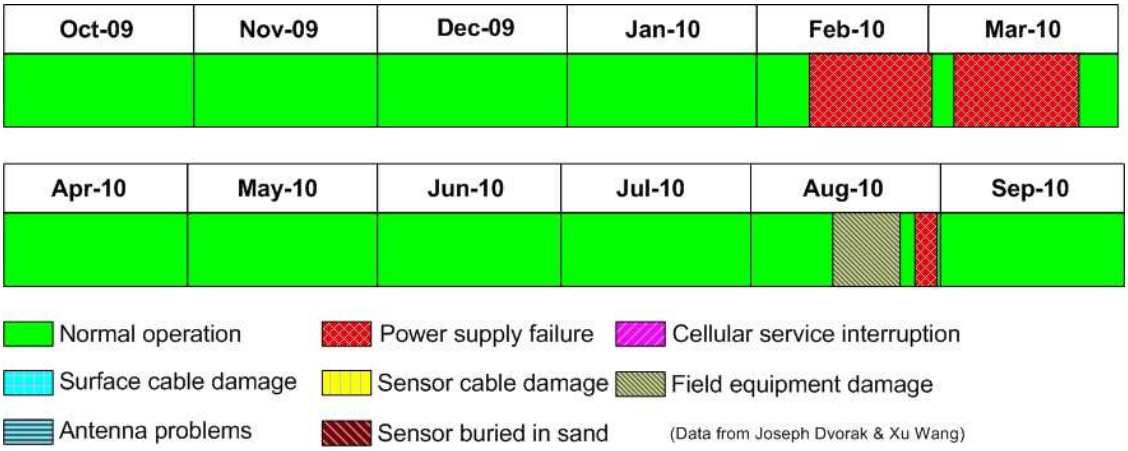


Figure 5.61 Performance in wireless data transmission and causes for failures for Anita Near sensor node

For the Anita Near and Far sensor nodes, the gateway station and central station were located next to each other. As a result, the packet loss rates and transmission error rates were low. There was a transmission breakdown due to a power failure during Feb. 2010, when we obtained only 10 days of data.

The average packet loss rate and transmission error rate for Anita Far sensor node were 4.29% and 0.66% respectively. Figure 5.62 shows the monthly average packet loss rate and transmission error rate for sensor node at Anita Far, APG, MD.

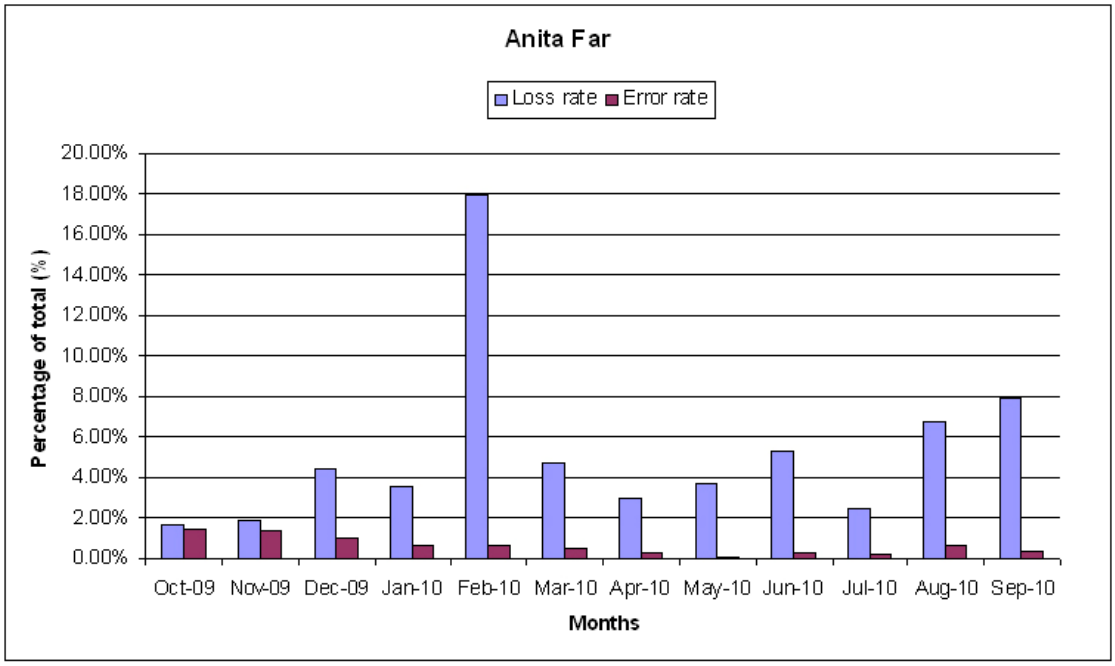


Figure 5.62 Monthly average packet loss rate and transmission error rate for Anita Far, APG, MD sensor node

Performance of the three-tier WSN in transmitting data from the Anita Far sensor node and causes for failures within a twelve-month period are illustrated in Figure 5.63.

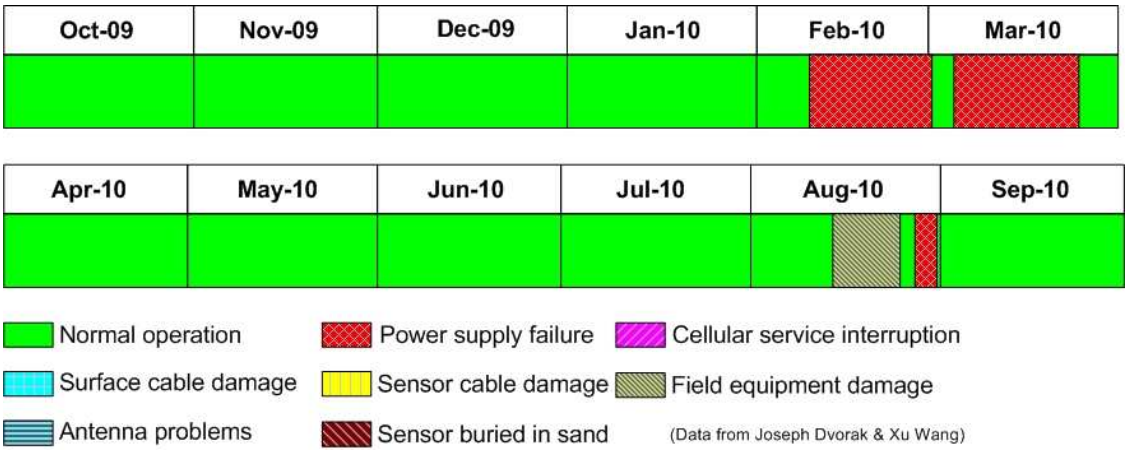


Figure 5.63 Performance in wireless data transmission and causes for failures for Anita Far sensor node

5.3.2 Sediment measurement

The optical sensors responded to changes in sediment concentration. When the sediment concentration increased, signals from the phototransistors placed 45° from the

infrared LED (IR45) and orange LED (ORA45) increased, while the signal from phototransistor placed 180° from the orange LED (ORA180) decreased. Usually, sediment concentration increased during or immediately after rain events, which disturbed soil sediment. Figure 5.64 shows sensor reading changes during three rain events in Little Kitten Creek, Manhattan, Kansas during a 12-day period, from March 26 to April 7 in 2010.

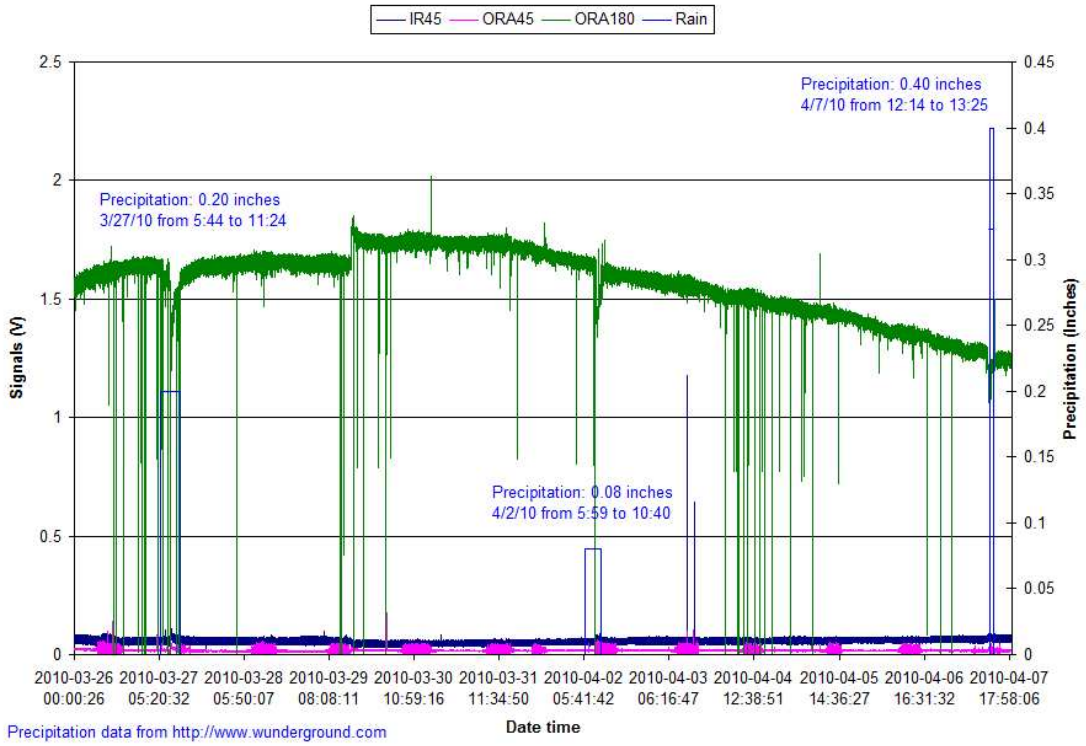


Figure 5.64 Sediment sensor signals and precipitation. Data recorded from March 26 to April 7, 2010, at the Little Kitten Creek site in Manhattan, KS

Within this 12-day period, the historical precipitation data from "Weather Underground" (Weather Underground, 2010) showed three rain events in the area. Accumulated precipitation for each of the rain events was given in Figure 5.64. Corresponding variations in sensor signals can also be observed. After the rain events, the signals gradually returned to their normal levels. The general decreasing trend of the ORA180 signal was probably caused by sensor lens fouling.

Detailed sediment sensor signal and precipitation for these three rain events are shown in Figures 5.65-5.67. A scale factor of 10 was used to the IR45 and ORA45 signals to better observe their variations.

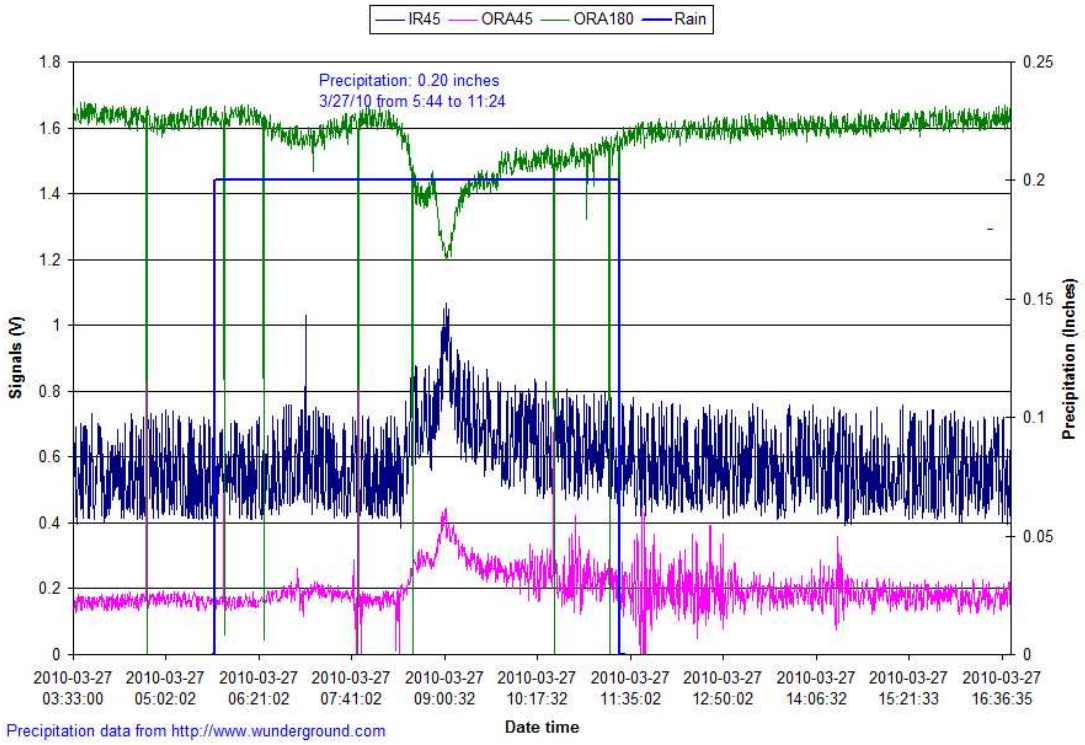


Figure 5.65 Sediment sensor signals and precipitation for the 1st rain event at the Little Kitten Creek site in Manhattan, KS. The IR45 and ORA45 signals were enlarged by a scaling factor of 10.

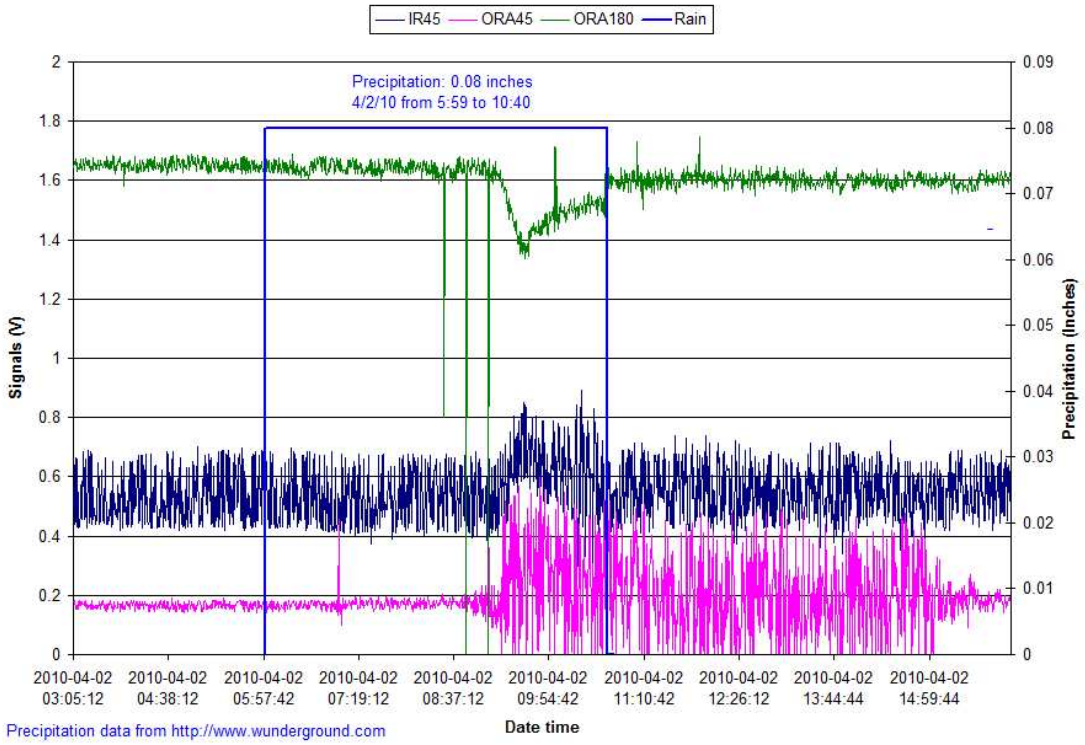


Figure 5.66 Sediment sensor signals and precipitation for the 2nd rain event at the Little Kitten Creek site in Manhattan, KS. The IR45 and ORA45 signals were enlarged by a scaling factor of 10.

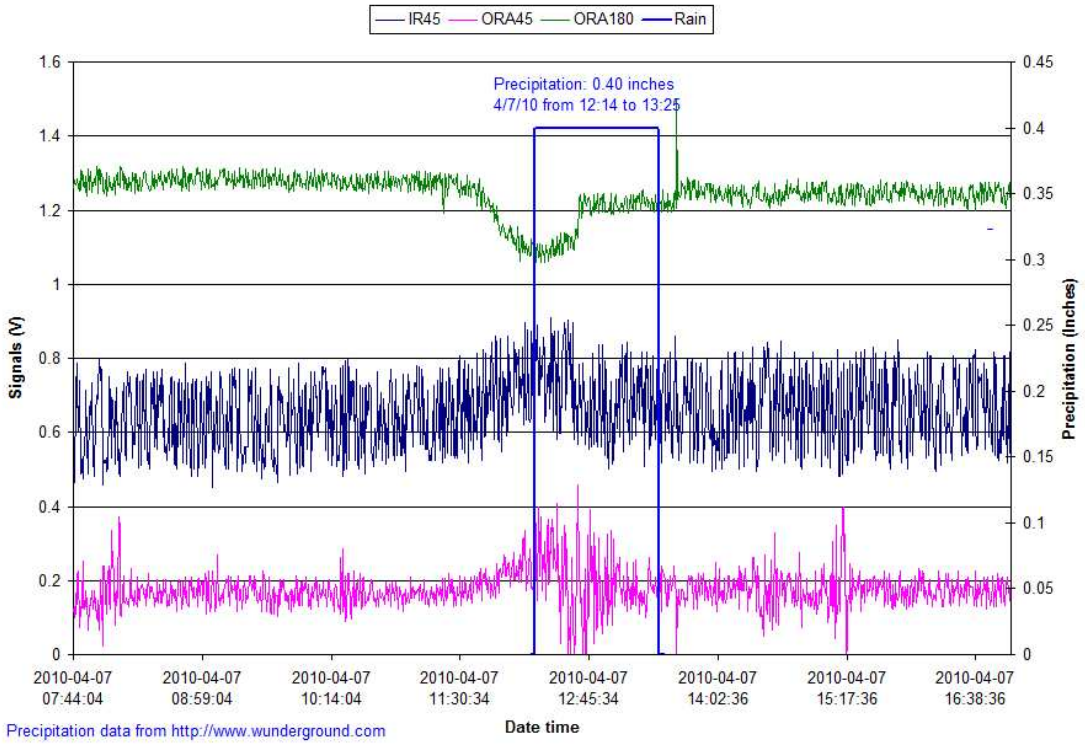


Figure 5.67 Sediment sensor signals and precipitation for 3rd rain event at the Little Kitten Creek site in Manhattan, KS. The IR45 and ORA45 signals were enlarged by a scaling factor of 10.

Figure 5.68 gives another example of sediment measurement at the Wildcat Bridge site at Ft. Riley, Kansas. During the storm, dramatic changes in all three signals (IR45, ORA45 and ORA180) were observed. The historical precipitation data from the Weather Underground showed that a storm occurred a few hours before significant changes were found in the sensor signals. This was probably due to the fact that Wildcat Creek was a larger creek than Little Kitten Creek. Thus, it took a longer time to have disturbed soil sediment accumulated a detectable level. Also notice that, after the storm, it took a longer time for the sensor signals to return to their normal levels.

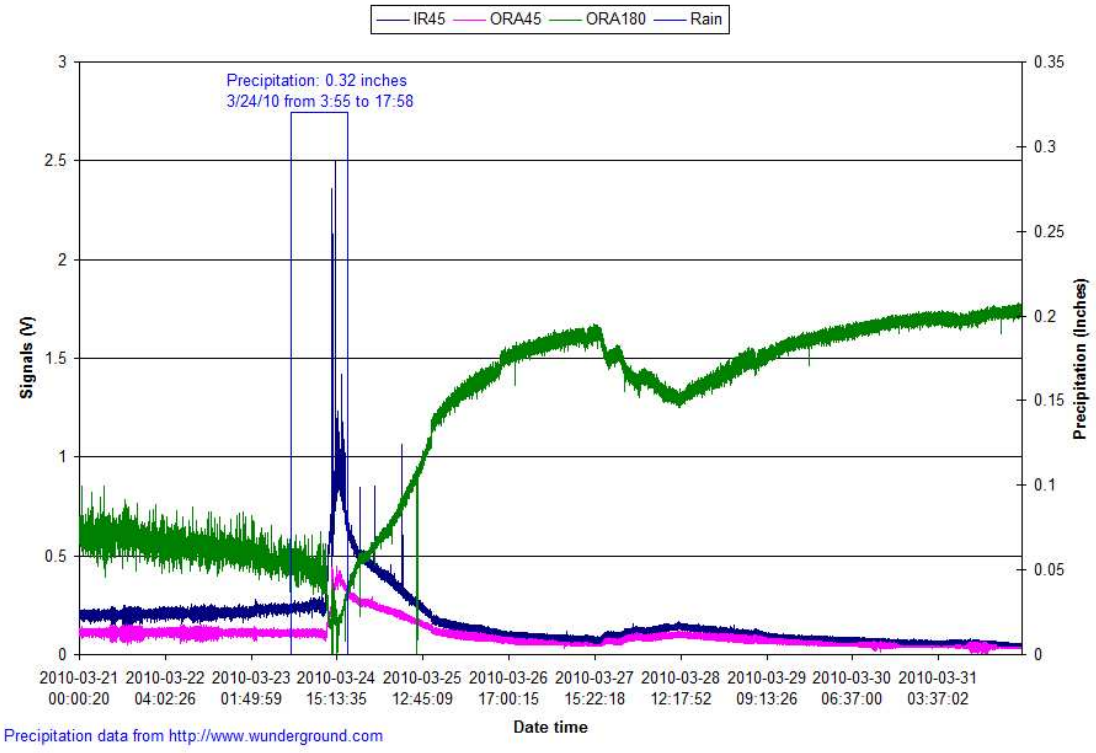


Figure 5.68 Sediment sensor signals measured during a rain event at Wildcat Bridge, Ft. Riley, KS

The sediment data in Figure 5.69 showed two rain events at the Pine Knot South site at Ft. Benning, Georgia. In both events, detectable changes in the signals did not immediately occur when the storm started.

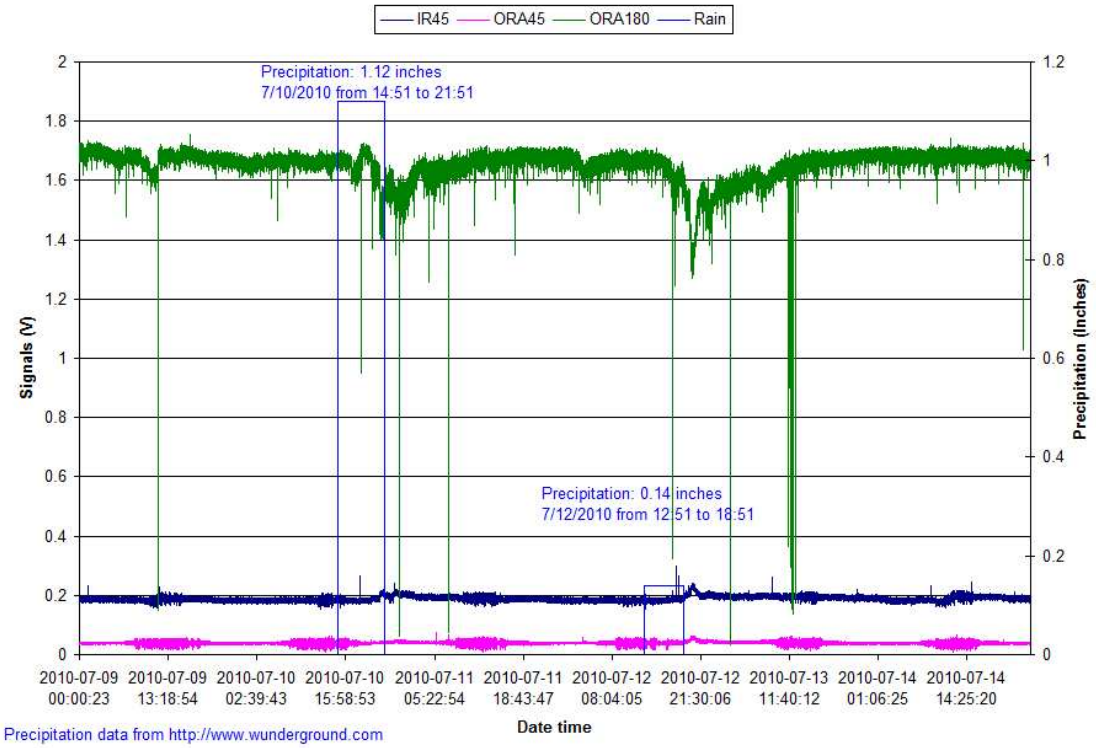


Figure 5.69 SSC signals for Pine Knot South, Ft. Benning, GA

Detailed sediment sensor signal and precipitation for these three rain events are shown in Figures 5.70 and 5.71. Again, a scale factor of 10 was used to the IR45 and ORA45 signals to better observe their variations.

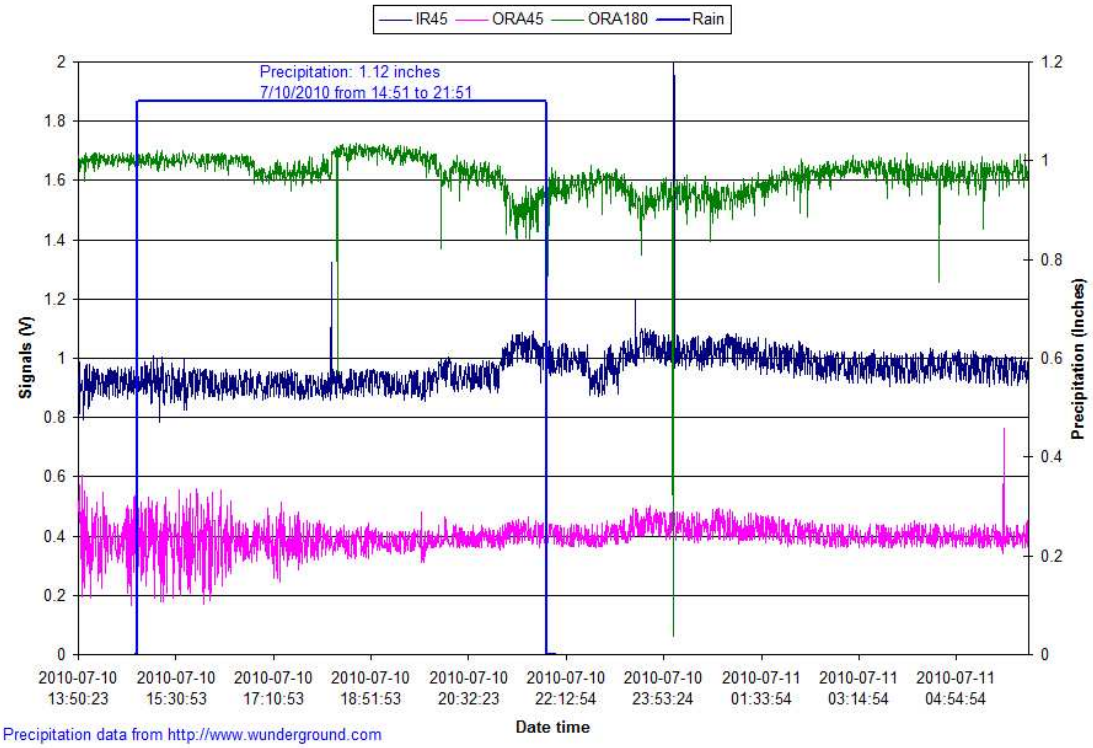


Figure 5.70 Sediment sensor signal and precipitation for the 1st rain event at Pine Knot South, Ft. Benning, GA. The IR45 signal was enlarged by a scaling factor of 5 and ORA45 signal was enlarged by a scaling factor of 10.

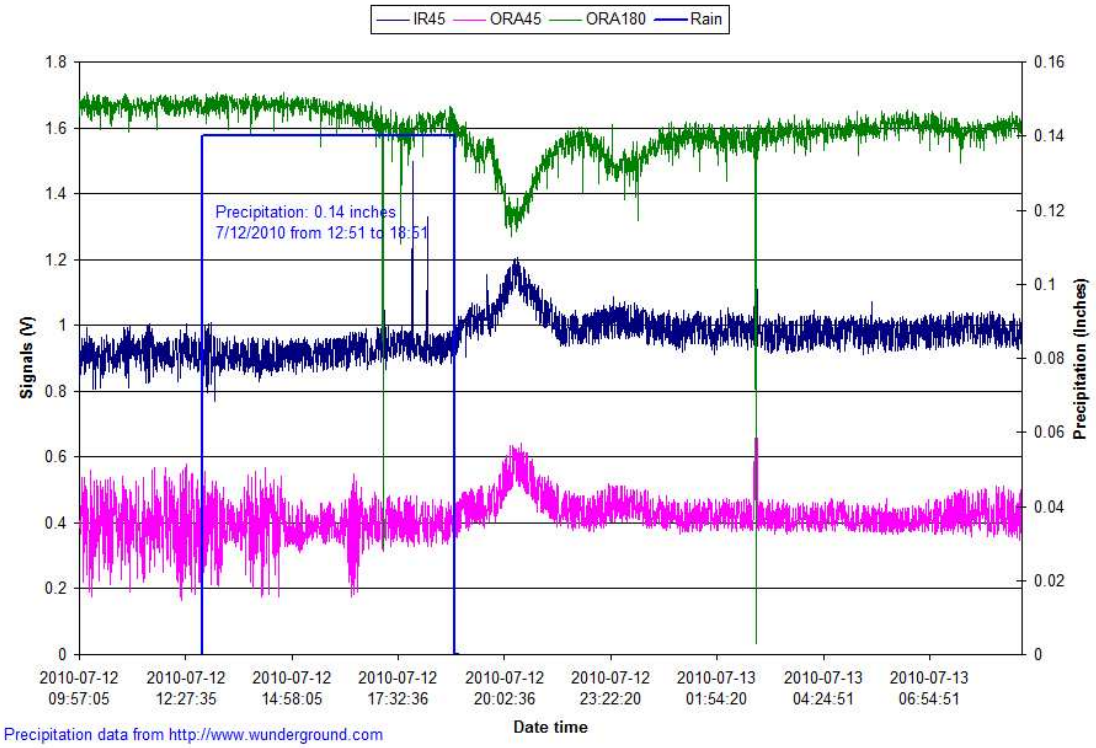


Figure 5.71 Sediment sensor signal and precipitation for the 2nd rain event at Pine Knot South, Ft. Benning, GA. The IR45 signal was enlarged by a scaling factor of 5 and ORA45 signal was enlarged by a scaling factor of 10.

5.3.3 Water temperature measurement

The temperature data were used for temperature compensation for SSC calculation (Zhang, 2009). Figure 5.72 shows temperature measured at the Little Kitten Creek site in Manhattan, KS. The daily high-low temperatures recorded by "Weather Underground" (Weather Underground, 2010) for the same region were also shown in the figure. Comparison between the two sets of data indicated that the range of daily water temperature variation was smaller than that of daily air temperature variation. However, the main trends in water and air temperature variations were similar.

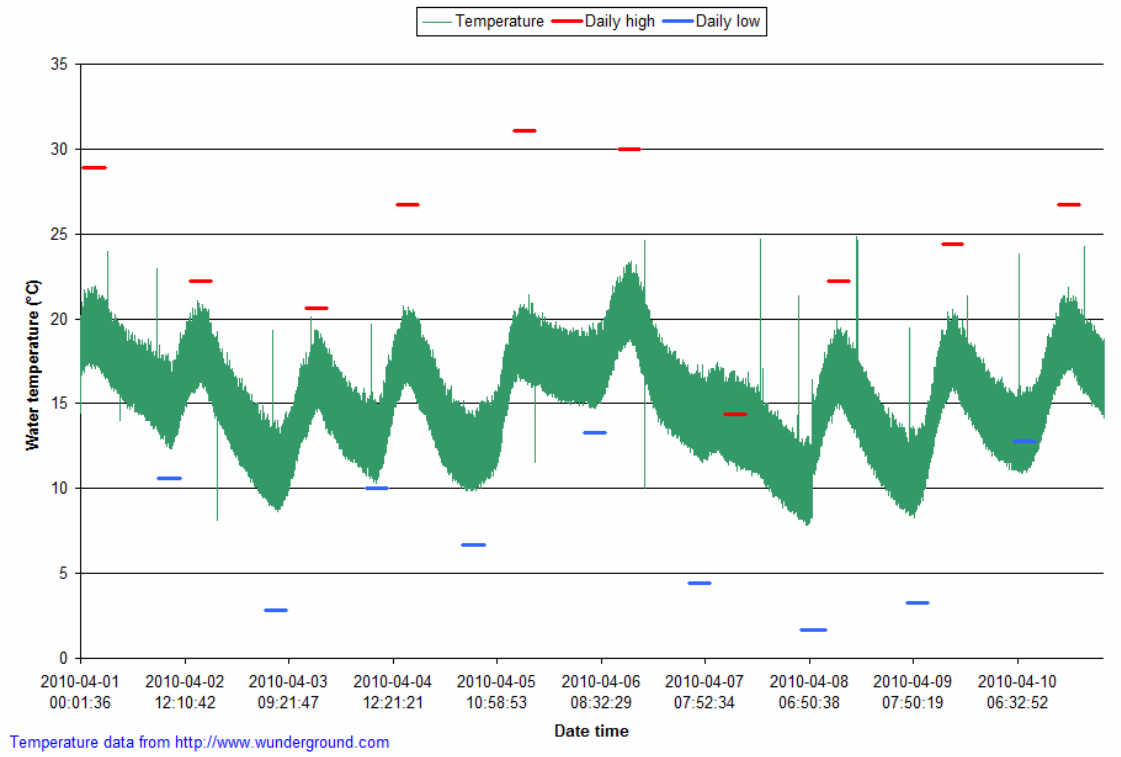


Figure 5.72 Water temperature measured at the Little Kitten Creek site in Manhattan, KS

5.3.4 Precipitation measurement

Figure 5.73 shows the precipitation data recorded on June 2, 2009 at Wildcat Bridge, Ft. Riley, KS. As we described early, the rain gauge data contained a cumulative counter, which represented the total number of tipping pulses generated by the rain gauge. Each tipping pulse represented 0.01 inches of rainfall. Tracing changes of this counter within a period of time, we could obtain the precipitation data for this period. In Figure 5.73, precipitation measured by the rain gauge was compared with the record from the "Weather Underground" web site (Weather Underground, 2010). The precipitation measured by the rain gauge was drawn in inches for the accumulated tipping pulses at times they were recorded, whereas the Weather Underground data was drawn as rectangles, with the width and height representing the time period and total precipitation, respectively. From the figure it can bee seen that the "Weather Underground" web site and the rain gauge recorded two rain events within similar time periods. For the first rain event, the precipitations reported by the two sources were 0.92 and 1.2 inches,

respectively. The precipitations reported for the second event were 0.31 and 0.2 inches, respectively. The small differences between the two sources were probably due to the difference in locations where the measurements were made.

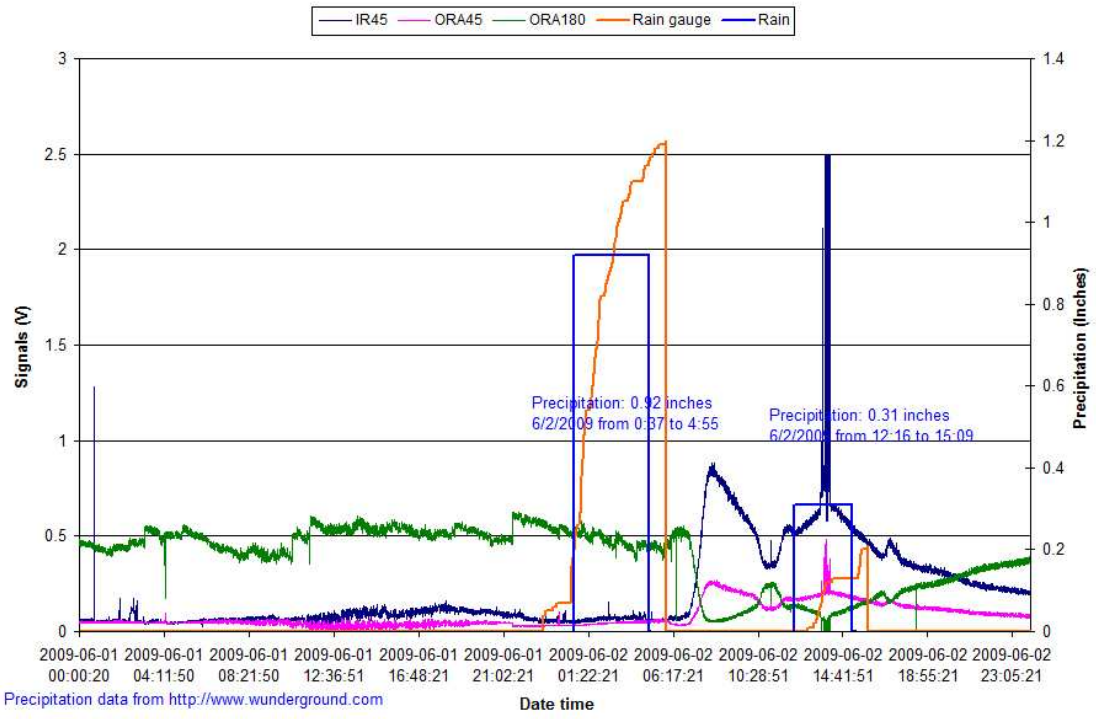


Figure 5.73 Precipitation data from Wildcat Bridge, Ft. Riley, KS

5.3.5 Air-blast cleaning

The air-blast cleaning valve was turned on for 10 seconds every hour. Figure 5.74, shows the sediment signals measured within a 24 hour period. It was obvious that, every time the lenses was cleaned, the ORA180 signal had a sudden increase, whereas the IR45 and ORA45 signals displayed a reduction at the same time, which can be displayed with a larger scale (Figure 5.75). According to Zhang (2009), fouling on the sensor lenses usually caused decreases in transmitted light (ORA180) and increases in backscattered light (IR45 and ORA45). After the lenses were cleaned, the signals returned to their normal levels.

Compared with other components in the system, the air compressor consumed a much greater amount of electric power. Figure 5.74 shows the measured battery voltage during the same period the sediment was measured. Remarkable voltage drops occurred

every two hours and, in most cases, the time the battery voltage dropped coincided with the time the ORA180 signal increased, indicating that the air pressure accumulated in the air tank could only support two 10-second air-blast cleaning operations. After two cleaning operations, the air compressor had to be turned on to refill the air tank.

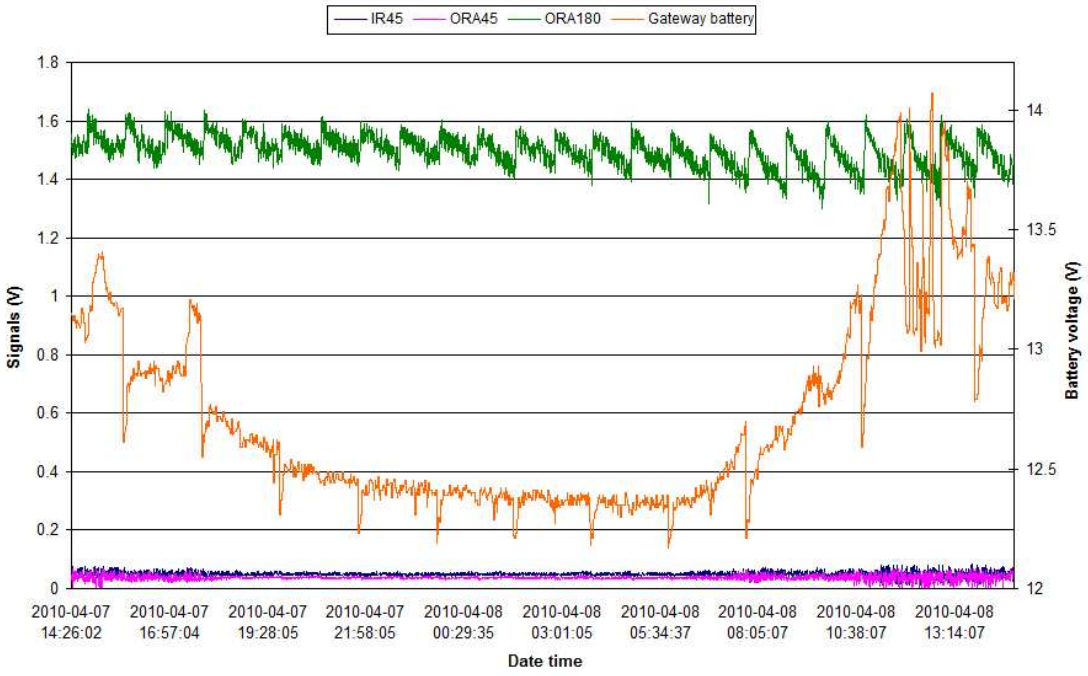


Figure 5.74 Air-blast clean effects at Wildcat Bridge, Ft. Riley, KS

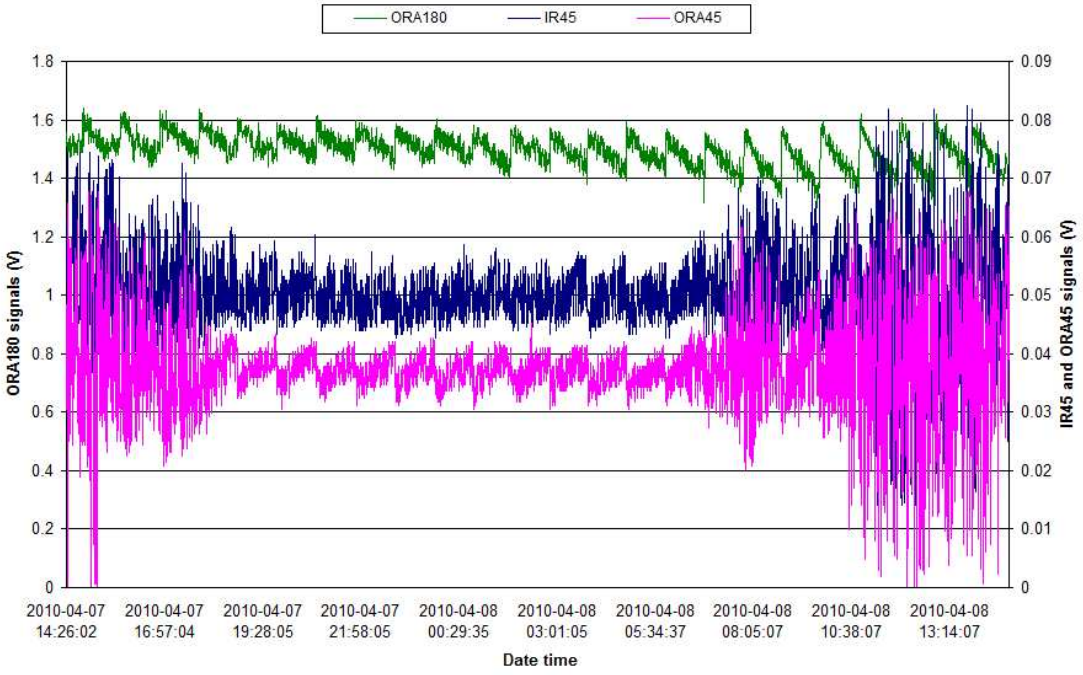


Figure 5.75 Sediment signals with air-blast clean effects at Wildcat bridge, Ft. Riley, KS

5.3.6 Velocity measurement

Sensors for simultaneous measurement of sediment and velocity were deployed at both the Upatoi and Pine Knot sites at Fort Benning and the Little Kitten site at Fort Riley. For these sensors, both velocity raw data and the calculated flow velocity were transmitted to the database server. The velocity raw data contained two sets of data - the upstream data named “ORA180 1” and the downstream data named “ORA180 2”. For both sets of data, 512 samples were taken at a sampling frequency of 279 Hz. Figure 5.76 displays an example of the raw data. The obvious voltage drops on both the upstream and downstream signals were caused by the release of blue dye. The time lag between the downstream and upstream signals was calculated through a cross-correlation analysis. Figure 5.77 shows the calculated cross-correlation coefficient.

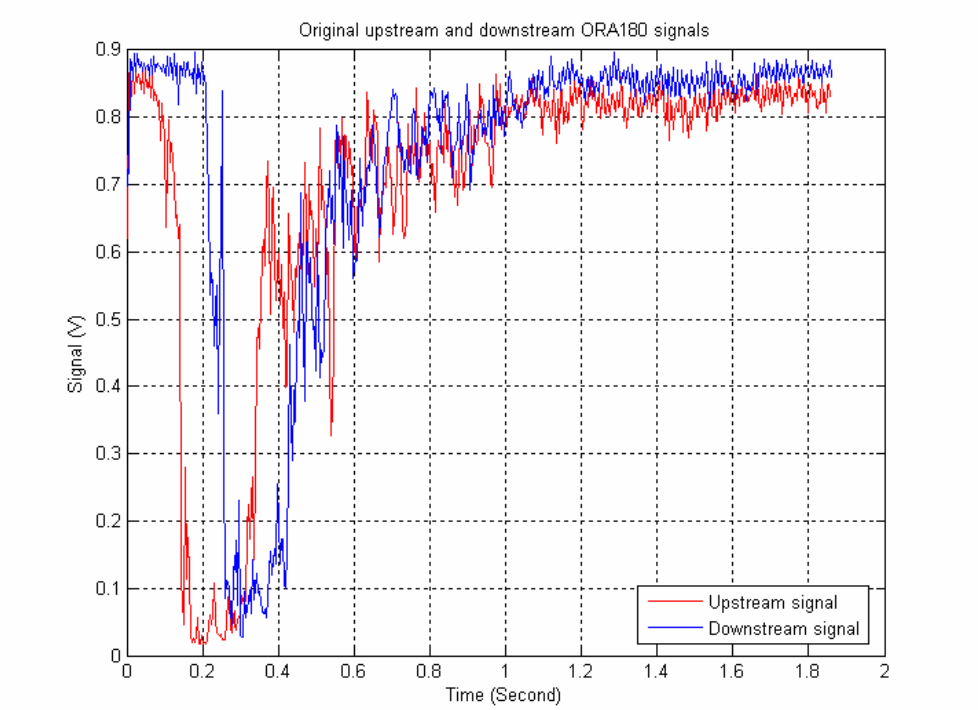


Figure 5.76 Velocity data taken from Upatoi South, Ft. Benning, GA

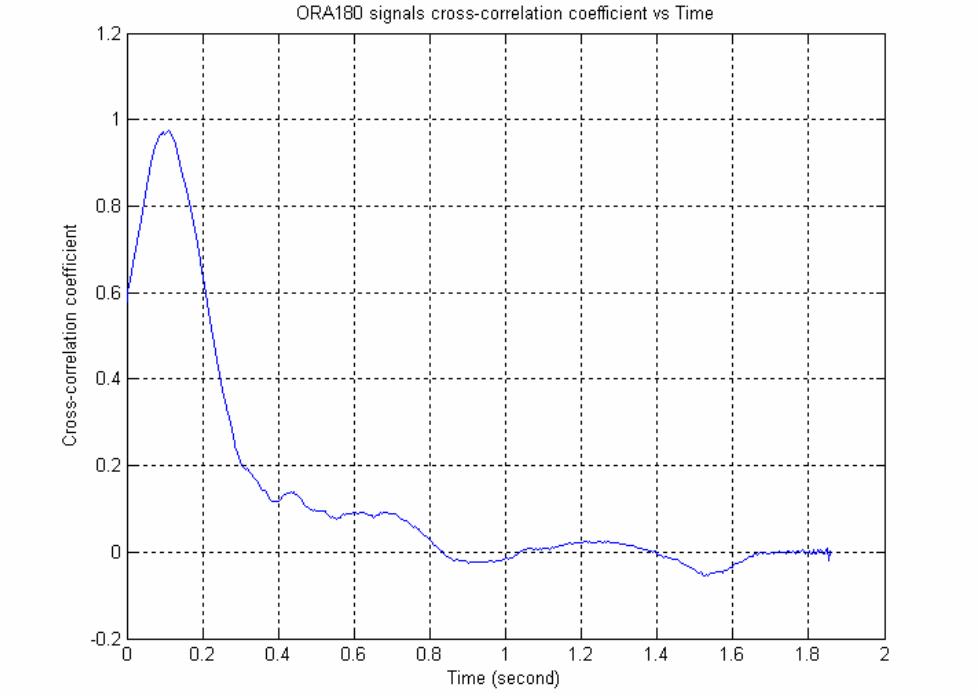


Figure 5.77 Calculated cross-correlation coefficient

5.4 System cost

Material costs for the major components of the three-tier WSN system, including sensor node, gateway station, repeater station, and central station, are listed in the following tables.

Table 5.9 Material cost for a sensor node that measures only sediment concentration

Components	Price
Sensory module	$571.00
Sensor cleaning module	$167.00
Control and communication module	$446.00
Power supply module	$567.00
Total	$1,751.00

Table 5.10 Material cost for a sensor node that measures both sediment and velocity

Components	Price
Sensory module	$828.00
Sensor cleaning module	$167.00
Control and communication module	$446.00
Power supply module	$567.00
Total	$2,008.00

Table 5.11 Material cost for a gateway station

Components	Price
Stargate module	$747.00
Radio communication module	$900.00
Power supply module	$395.00
Total	$2,042.00

Table 5.12 Material cost for a repeater station

Components	Price
Radio communication module	$900.00
Power supply module	$335.00
Total	$1,235.00

Table 5.13 Material cost for a central station

Components	Price
Cellular module	$575.00
Radio communication module	$900.00
Power supply module	$335.00
Total	$1,810.00

Table 5.14 Material cost for the three-tier WSN installed at the three military installations with twelve sensor nodes

Components	Price
Ft Riley, KS site	$17,667.00
Ft. Benning, GA site	$14,647.00
APG, MD site	$12,898.00
Total	$45,212.00

The costs listed in these tables do not include labor and traveling cost for system development, deployment, and maintenance.

5.5 Potential and limitations of the system

Field tests of the three-tier wireless sensor network demonstrated that the system was capable of remote, real-time environmental monitoring. With proper site selection, accurate antenna positioning, and proper maintenance, the system achieved low packet loss rates and low transmission error rates. The sensors were capable of measuring sediment concentration and flow velocity simultaneously. The temperature and precipitation measurements were effective. Frequent lens cleaning using the air-blast cleaning system helped reduce signal distortion due to lens fouling.

This system, with proper modifications, could be used for other field monitoring applications, such as precision agriculture, watershed monitoring, and forest fire prevention. The system was especially useful for remote monitoring in areas with insufficient cellular coverage. Potentials of the developed technology are great.

There were a few limitations for the system. A major limitation was the need for frequent field maintenance by experienced personnel. During the past two years, we encountered many difficulties at the Ft. Benning and APG sites mainly because we had no experienced personnel at the sites to conduct troubleshooting and repair. Although the system could issue alert messages for low power supply, it was unable to detect other problems automatically. Currently, the only solution for most system failures was to send technicians to the field for inspection and repair.

Another problem was a number of uncontrollable/unpredictable factors that affected the operation of the system. Examples were cellular carrier service stoppage, server compute IP address change, severe storm and associated flood, snow cover on solar panels, animal damage on wires and cables, and human vandalism and sabotage. Most of these factors were random events. Although we received daily reports from the server on data transmission quality and battery voltages, we have not implemented solutions that would proactively predict and prevent most of the system failures.

CHAPTER 6 - CONCLUSIONS

The conclusions derived from our studies on both the two-tier and three tier wireless sensor networks are summarized as follows:

6.1 Two-tier wireless sensor network

In the two-tier WSN study, we developed a remote, real-time, wireless sediment monitoring system and addressed several challenges associated with the system.

1. For the higher tier communication, we used a commercial GPRS mobile service to provide long-range wireless data transfer in an area where wired or wireless Internet access was not available. The use of a range extender antenna enhanced GPRS signal power and reduced abrupt FTP disconnections.

2. For the lower-tier communication, wireless transmission was achieved in a radio-hostile environment by using properly selected omnidirectional and Yagi directional antennas with properly selected antenna tower heights.

3. Reliable power supplies were provided in the testing area with solar panels. By using a switching voltage regulator in hardware and a short sampling duration in software, the battery life was extended.

4. Software for both the remote system and the indoor server computer was developed. Algorithms for auto-restart after power outage developed for both the server and the Stargate computer improved the reliability of the system.

5. The sensors successfully measured in-stream sediment concentrations and the two-tier WSN successfully transmitted the measurement data to an indoor server computer, which made the data available on the Internet. The system provided the end-users with convenient access to the real-time data without installing additional software.

6. The cost for the system was reduced by using appropriate antennas to achieve long-range wireless transmission. The total cost for such a system with two sensors was US $3,000.

6.2 Three-tier wireless sensor network

In the study of three-tier WSN, we further developed the architecture of the communication system and applied it to environmental monitoring at a much larger scale. The following conclusions can be drawn from this study:

1. The three-tier WSN architecture was successfully implemented at three military installations - Fort Riley, Kansas, Fort Benning, Georgia, and Aberdeen Proving Ground (APG), Maryland.

2. In addition to in-stream sediment concentration, other environment-related variables, such as water temperature, precipitation, and stream flow velocity, were also monitored. All data were transmitted to a server computer in real time through the three-tier WSN.

3. For the lower tier communication, sensor nodes successfully controlled data acquisition, rain gauge measurement, velocity measurement, and air-blast lens cleaning. Data from the sensor nodes were wirelessly transferred to the gateway stations using wireless motes. The longest transmission distance was about 100 meters.

4. Wireless dataloggers/radio transceivers were used for the middle-tier communications. These dataloggers were installed in the gateway, repeater, and central stations. Careful site selection insured good network performances. The longest transmission ranges at APG and Fort Riley were 8.1 and 5.9 km, respectively.

5. Signal splitters were used in two repeater stations and a central station at the Fort Riley site, where data transmissions in multiple directions were involved. The combined use the splitters and Yagi directional antennas enhanced signal strength, reduced the need for additional repeaters and, hence, reduced the overall cost.

6. For the higher-tier data communication, commercial cellular data services of GPRS for T-Mobile GSM system, EVDO for Verizon CDMA system, and EDGE for AT&T GSM system, were used at the Fort Riley, Fort Benning, and APG sites, respectively. The server computer at Kansas State University (KSU) received data from all three installations successfully.

7. Solar panels were selected for all experimental sites based on electrical power demands. Deep-cycle, marine batteries were used as power storages for the solar panels and power supplies for the system. This energy harvesting mechanism met the need of the system for reasonably long life span.

8. System reliability and performance were enhanced by enabling the brown-out detection function on the motes, reducing the Stargate's data backup file size when it became necessary, and adding virtual memory on the Stargate.

9. Complete software for the three-tier WSN system was developed. Functions of the software included data acquisition and processing, relay control, time synchronization, data communication, and data storage. The software automatically handled the system's starting or restarting process, which reduced the maintenance need.

10. Time synchronization between two sensor motes and the mote at the gateway station guaranteed that no sediment or velocity measurement was conducted during the air cleaning operation. Time synchronization between the Stargate and the datalogger improved the accuracy of time information generated by the Stargate.

11. Long-term field experiments conducted in three military installations indicated that low packet loss rate and error rate were achieved for most sensor nodes. However, maintaining low packet loss and transmission error rate for a long period was difficult. Causes for failures and interrupts included carrier service interrupts, weather damage, animal damage, and human vandalism and sabotage.

12. Real-time data received from three military installations were saved into a MySQL database server. These data can be accessed via Internet.

13. The material cost for the three-tier WSN installed at three military installations with totally 12 sensors was less than $46,000.

CHAPTER 7 - REFERENCES

Agilent Technologies. 2011. Agilent N9340B Handheld Spectrum Analyzer (HSA) Technical Overview.

Atmel Corporation. 2006. Atmel 8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash.

Barrenetxea, G., F. Ingelrest, G. Schaefer and M. Vetterli. 2008. Wireless sensor networks for environmental monitoring: The sensorscope experience. In *2008 IEEE International Zurich Seminar on Communications (IZS)*, 98-101.

Boutell, T. 2007. GD library. Available at: http://www.libgd.org/Main_Page. Accessed March/15 2010.

Brodsky, I. 1990. Wireless Data Networks and the Mobile Workforce. *Telecommunications* 24(12): 31-35.

Brown, D. 1985. A physical meteor-burst propagation model and some significant results for communication system design. *IEEE Journal on Selected Areas in Communications* 3(5): 745-755.

Campbell Scientific. 2010a. CR206: Datalogger with 915 MHz Spread-Spectrum Radio. Available at: http://www.campbellsci.com/cr206. Accessed Feb/20 2010.

Campbell Scientific. 2010b. Te525 tipping bucket rain gauge.

Carullo, A., S. Corbellini, M. Parvis and A. Vallan. 2009. A Wireless Sensor Network for Cold-Chain Monitoring. *IEEE Transactions on Instrumentation and Measurement* 58(5): 1405-1411.

Chan, H. A. 2007. Comparing wireless data network standards. *AFRICON 2007*,1-15. Windhoek, Namibia.

Cisco. 2010. Chapter 13: Point to Point Protocol (PPP). In *Internetworking Technology Handbook*, 1-4. ed. Anonymous , .

Commuri, S., V. Tadigotla and M. Atiquzzaman. 2008. Reconfigurable Hardware Based Dynamic Data Aggregation in Wireless Sensor Networks. *International Journal of Distributed Sensor Networks* 4(2): 194-212.

Corke, P., P. Valencia, P. Sikka, T. Wark and L. Overs. 2007. Long-duration solar-powered wireless sensor networks. In *Proceedings of the 4th workshop on Embedded networked sensors*, 33-37. Cork, Ireland: ACM.

Country Studies US. 2008. Manhattan weather. Available at: http://countrystudies.us/united-states/weather/kansas/manhattan.htm. Accessed May/20 2010.

Crossbow. 2008. Stargate and mote. Available at: http://www.xbow.com. Accessed Feb/20 2008.

Crossbow. 2007. MTS/MDA Sensor Board Users Manual.

Crossbow. 2006. Stargate Developer's Guide.

Crossbow. 2005. Getting Started Guide. 30 San Jose, CA, USA.

Crossbow. 2003. MPR - Mote Processor Radio Board MIB - Mote Interface / Programming Board User's Manual.

Cumberland, B. C., J. S. Valacich and L. M. Jessup. 2004. Understanding meteor burst communications technologies. 47(1): 89-92.

Darr, M. J. and L. Zhao. 2008. A Model for Predicting Signal Transmission Performance of Wireless Sensors in Poultry Layer Facilities. *Transactions of the ASABE* 51(5): 1817-1822.

Demirkol, I., C. Ersoy and F. Alagoz. 2006. MAC protocols for wireless sensor networks: a survey. *IEEE Communications Magazine* 44(4): 115-121.

Eklund, C., R. B. Marks, K. L. Stanwood and S. Wang. 2002. IEEE standard 802.16: a technical overview of the WirelessMAN™ air interface for broadband wireless access. *IEEE communications magazine* 40(6): 98-107.

ESTCP. 2010. The Environmental Security Technology Certification Program. Available at: http://www.estcp.org/. Accessed 2/15 2010.

FCC. 2011. Federal Communications Commission - Rules and Regulations. FCC. Available at: http://wireless.fcc.gov/index.htm?job=rules_and_regulations. Accessed March 18 2011.

Gemtek. 2008. UWB technology. Available at: http://www.gemtek.com.tw/. Accessed Feb/18 2008.

Handley, P. A., D. P. Fraser and T. A. E. Ltd. 1992. Meteor-burst communications for rural environments. In *AFRICON'92 Proceedings., 3rd AFRICON Conference*, 378-381.

Hartung, C., R. Han, C. Seielstad and S. Holbrook. 2006. FireWxNet: a multi-tiered portable wireless system for monitoring weather conditions in wildland fire environments. In *Proceedings of the 4th international conference on Mobile systems, applications and services*, 41. Uppsala, Sweden: ACM.

He, T., B. M. Blum, J. A. Stankovic and T. Abdelzaher. 2004. AIDA: Adaptive application-independent data aggregation in wireless sensor networks. *ACM Transactions on Embedded Computing Systems (TECS)* 3(2): 426-457.

Healy, B. C., W. H. Shaw and J. R. Litko. 1989. A modeling perspective for meteor burst communication. In *Proceedings of the 21st conference on Winter simulation*, 1022-1031. ACM.

Johann, G., W. Daniel and A. Sajjad. 2008. Power Aware Simulation Framework for Wireless Sensor Networks and Nodes. *EURASIP Journal on Embedded Systems* 2008(3): 1-16.

Kim, Y., R. G. Evans, W. Iversen and F. J. Pierce. 2006. Instrumentation and control for wireless sensor network for automated irrigation. *ASABE paper 061105*. Portland, Oregon.

Krishnamachari, B., D. Estrin and S. Wicker. 2002. The impact of data aggregation in wireless sensor networks. In *Proceedings of the 22nd International Conference on Distributed Computing Systems*, 575-578.

Lewis, F. L. 2004. Wireless Sensor Networks. In *Smart Environments: Technologies, Protocols, and Applications*, ed. Cook, D.J. and Das, S.K., New York: John Wiley.

Lin, M., Y. Wu and I. Wassell. 2008. Wireless sensor network: Water distribution monitoring system. In *IEEE Radio and Wireless Symposium, Orlando, Florida*, 775-778.

Liu, H., Z. Meng and S. Cui. 2007. A wireless sensor network prototype for environmental monitoring in greenhouses. In *Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007. International Conference on*, 2344-2347. Shanghai, China: .

Mainwaring, A., D. Culler, J. Polastre, R. Szewczyk and J. Anderson. 2002. Wireless sensor networks for habitat monitoring. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, 88-97. ACM New York, NY, USA.

Mao, S., Y. T. Hou and M. Wu. 2008. Exploiting edge capability for wireless sensor networking. *IEEE Wireless Communications* 15(4): 67-73.

Marín, I., E. Arceredillo, A. Zuloaga and J. Arias. 2005. Wireless Sensor Networks: A Survey on Ultra-Low Power-Aware Design. In *Proceedings of world academy of science, engineering and technology*, 44-49. Citeseer.

Markov, A. 2006. WarFTP V1.82 Tutorial. Available at: http://www.warftp.org/guides/warftpd_1.82_tutorial.pdf. Accessed March/21 2010.

Mathioudakis, I., N. M. White and N. R. Harris. 2007. Wireless Sensor Networks: Applications Utilizing Satellite Links. In *IEEE 18th International Symposium on Personal, Indoor and Mobile Radio Communications, 2007. PIMRC 2007*, 1-5. Citeseer.

MaxStream. 2008. RF Modem. Available at: http://www.maxstream.net. Accessed Feb 20 2008.

Mukhopadhyay, S., C. Schurgers, D. Panigrahi and S. Dey. 2009. Model-Based Techniques for Data Reliability in Wireless Sensor Networks. *IEEE Transactions on Mobile Computing* 8(4): 528-543.

Niyato, D., E. Hossain, M. M. Rashid and V. K. Bhargava. 2007. Wireless sensor networks with energy harvesting technologies: a game-theoretic approach to optimal energy management. *IEEE Wireless Communications* 14(4): 90-96.

Nortel. 2007. Nortel WiMAX 802.16e Portfolio. Available at: http://telstarint.com/Nortel%20Wimax_files/Nortel%20mobile.pdf. Accessed Mar/18 2008.

Oetting, J., B. Allen, I. Hamilton and M. D. Bethesda. 1980. An analysis of meteor burst communications for military applications. *Communications, IEEE Transactions on [legacy, pre-1988]* 28(9 Part 1): 1591-1601.

Osechas, O., J. Thiele, J. Bitsch and K. Wehrle. 2008. Ratpack: Wearable Sensor Networks for Animal Observation. In *Conference proceedings: Annual International Conference of the IEEE Engineering in Medicine and Biology Society. IEEE Engineering in Medicine and Biology Society. Conference*, 538-541. Vancouver, Canada: Citeseer.

Raghunathan, V., S. Ganeriwal and M. Srivastava. 2006. Emerging techniques for long lived wireless sensor networks. *IEEE Communications Magazine* 44(4): 108-114.

Rappaport, T. S. 2007. *Wireless communications principles and practice.* second ed. New Jersey: Prentice Hall, Inc.,.

Red Hat, I. 2010. Cygwin User's Guide.

Ren, F., C. Lin and F. Liu. 2008. Self-correcting time synchronization using reference broadcast in wireless sensor network. *IEEE Wireless Communications* 15(4): 79-85.

Roberts, S. 1991. *Solar electricity: a practical guide to designing and installing small photovoltaic systems.* New York: Prentice Hall.

Sanchez, E. R., F. Gandino, B. Montrucchio and M. Rebaudengo. 2007. Increasing Effective Radiated Power in Wireless Sensor Networks with Channel Coding Techniques. In *Electromagnetics in Advanced Applications, 2007. ICEAA 2007. International Conference on*, 403-406. Turin, Italy: Citeseer.

SatWest. 2008. Satellite network systems. Available at: http://www.satwest.com/. Accessed Feb/20 2008.

SERDP. 2010. The Strategic Environmental Research and Development Program Homepage. Available at: www.serdp.org. Accessed 2/15 2010.

Sharpe, M. 2003. It's a bug's life: biosensors for environmental monitoring. *Journal of environmental monitoring : JEM* 5(6): 109N-113N. England.

Sierra Wireless. 2007. Raven X 1x/EV-DO for Alltel User Guide.

Stoianov, I., L. Nachman, S. Madden and T. Tokmouline. 2007. PIPENETa wireless sensor network for pipeline monitoring. In *Proceedings of the 6th international conference on Information processing in sensor networks*, 264-273. Cambridge, Massachusetts: ACM.

Stoll, Q. M. 2004. Design of a real-time, optical sediment concentration sensor. PhD diss. Manhattan, Kansas: Kansas State University, Department of Biological and Agricultural Engineering.

Turkka, J. and M. Renfors. 2008. Path Loss Measurements for a Non-Line-of-Sight Mobile-to-Mobile Environment. In *ITS Telecommunications, 2008. ITST 2008. 8th International Conference on*, 274-278.

USGS. 2010. Water Science Glossary of Terms. Available at: http://ga.water.usgs.gov/edu/dictionary.html#S. Accessed 2/15 2010.

Wang, N., N. Zhang and M. Wang. 2006. Wireless sensors in agriculture and food industry—Recent development and future perspective. *Computers and Electronics in Agriculture* 50(1): 1-14.

Wang, X. D. and H. V. Poor. 2003. *Wireless communication systems: advanced techniques for signal reception.* Prentice Hall.

Weather Underground. 2010. Weather Station History. Available at: http://www.wunderground.com. Accessed Aug/16 2010.

Werner-Allen, G., J. Johnson, M. Ruiz, J. Lees and M. Welsh. 2005. Monitoring volcanic eruptions with a wireless sensor network. In *Proc. Second European Workshop on Wireless Sensor Networks (EWSN'05)*, 108-120. Citeseer.

Wikipedia. 2011. Wireless sensor network. Available at: http://en.wikipedia.org/wiki/Wireless_sensor_network. Accessed 04/27 2011.

Wikipedia. 2010a. File Transfer Protocol. Available at: http://en.wikipedia.org/wiki/File_Transfer_Protocol. Accessed Jan/20 2010.

Wikipedia. 2010b. frequency-hopping spread spectrum. Available at: http://en.wikipedia.org/wiki/Frequency-hopping_spread_spectrum. Accessed Jun/16 2010.

Xu, K., Q. Wang, H. Hassanein and G. Takahara. 2005. Optimal wireless sensor networks (WSNs) deployment: minimum cost with lifetime constraint. In *IEEE International Conference on Wireless And Mobile Computing, Networking And Communications, 2005.(WiMob'2005)*, 454-461.

Xu, N., S. Rangwala, K. K. Chintalapudi, D. Ganesan, A. Broad, R. Govindan and D. Estrin. 2004. A wireless sensor network for structural monitoring. In *Proceedings*

of the 2nd international conference on Embedded networked sensor systems, 13-24. ACM New York, NY, USA.

Yang, L. and G. B. Giannakis. 2004. Ultra-wideband communications: an idea whose time has come. *IEEE Signal Processing Magazine* 21(6): 26-54.

Yi, S., Y. Pei and S. Kalyanaraman. 2003. On the capacity improvement of ad hoc wireless networks using directional antennas. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, 108-116. ACM New York, NY, USA.

Younis, M. and K. Akkaya. 2008. Strategies and techniques for node placement in wireless sensor networks: A survey. *Ad Hoc Networks* 6(4): 621-655.

Zhang, Y. 2009. An optical sensor for in-stream monitoring of suspended sediment concentration. PhD Dissertation, Kansas State University, Department of Biological and Agricultural Engineering.

Zhang, Y., N. Zhang, G. Grimm, C. Johnson, D. Oarrd and J. Steichen. 2007. Long-term field test of an optical sediment-concentration sensor at low-water stream crossings (LWSC). *ASABE paper* 072137. St. Joseph, Mich.

Zhang, Z. 2004. Investigation of wireless sensor networks for precision agriculture. *ASABE Paper* 41154. Ottawa, Ontario, Canada.

Zhou, Y., X. Yang, X. Guo, M. Zhou and L. Wang. 2007. A Design of Greenhouse Monitoring & Control System Based on ZigBee Wireless Sensor Network. In *International Conference on Wireless Communications, Networking and Mobile Computing, 2007.* 2563-2567. Shanghai: WiCom 2007.

Appendix A- Mote program for sensor control and data transmitting used in one or two-tier WSN

File Name: XsensorMDA300M.nc

```
/* XSensorMDA300M                                                    tab:4
 *  IMPORTANT: READ BEFORE DOWNLOADING, COPYING, INSTALLING OR USING. By
 *  downloading, copying, installing or using the software you agree to
 *  this license.  If you do not agree to this license, do not download,
 *  install, copy or use the software.
 *
 *  Intel Open Source License
 *
 *  Copyright (c) 2002 Intel Corporation
 *  All rights reserved.
 *  Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:
 *
 * Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. Neither the name of the Intel Corporation nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.
 */
/**************************************************************************
 *    - Tests the MDA300 general prototyping card
 *      (see Crossbow MTS Series User Manual)
 *    -  Read and control all MDA300 signals:
 *     ADC0, ADC1, ADC2, ADC3,...ADC11 inputs, DIO 0-5,
 *     counter, battery, humidity, temp
```

```
*-----------------------------------------------------------------------
* Output results through mica2 uart and radio.
* Use xlisten.exe program to view data from either port:
*  uart: mount mica2 on mib510 with MDA300
*        (must be connected or now data is read)
*        connect serial cable to PC
*        run xlisten.exe at 57600 baud
*  radio: run mica2 with MDA300,
*         run another mica2 with TOSBASE
*         run xlisten.exe at 56K baud
* LED: the led will be green if the MDA300 is connected to the mica2 and
*      the program is running (and sending out packets).  Otherwise it is red.
*-----------------------------------------------------------------------
* Data packet structure:
* PACKET #1
* ----------------
*  msg->data[0] : sensor id, MDA300 = 0x81
*  msg->data[1] : packet number = 1
*  msg->data[2] : node id
*  msg->data[3] : reserved
*  msg->data[4,5] : analog adc data Ch.0
*  msg->data[6,7] : analog adc data Ch.1
*  msg->data[8,9] : analog adc data Ch.2
*  msg->data[10,11] : analog adc data Ch.3
*  msg->data[12,13] : analog adc data Ch.4
*  msg->data[14,15] : analog adc data Ch.5
*  msg->data[16,17] : analog adc data Ch.6
* C:\Program Files\UCB\cygwin\opt\tinyos-
1.x\contrib\xbow\tos\sensorboards\mda300\SamplerC.nc
 *****************************************************************/
// include sensorboard.h definitions from tos/mda300 directory
includes sensorboard;
module XSensorMDA300M
{
```

```
    provides interface StdControl;
     uses {
          interface Leds;
          //Sampler Communication
          interface StdControl as SamplerControl;
          interface Sample;
          //UART communication
          interface StdControl as UARTControl;
          interface BareSendMsg as UARTSend;
          interface ReceiveMsg as UARTReceive;
          //RF communication
          interface StdControl as CommControl;
          interface BareSendMsg as SendMsg;
          interface ReceiveMsg as ReceiveMsg;
          //Timer
          interface Timer as SamplingTimer;
          interface Timer as SleepTimer1;
          interface Timer as SleepTimer2;
          interface Timer as SleepTimer3;
          interface Timer as SleepTimer4;
          interface Timer as SleepTimer5;
          interface Timer as SleepTimer6;
          interface Timer as SleepTimer7;
          interface Timer as SleepTimer8;
          interface Timer as SleepTimer9;
          interface Timer as SleepTimer10;
          interface Timer as SleepTimer11;
    }
}
implementation
{
//#define ANALOG_SAMPLING_TIME    30520    //sampling interval = 30 seconds
#define ANALOG_SAMPLING_TIME   10000    //sampling interval = 10 second, not in used.
#define DIGITAL_SAMPLING_TIME  100      //0.1 second
```

```
#define MISC_SAMPLING_TIME     0
//#define SCALE_SAMPLING_TIME    2750   //every 30 seconds with any delay in datalogger
#define SCALE_SAMPLING_TIME    214 //(for 0.1 second DIGITAL_SAMPLING_TIME)
1992 (for 0.01second DIGITAL_SAMPLING_TIME) 30 seconds
#define READ_DELAY 48  //delay time from turn on and turn off light  48 for 96 ms
#define ANALOG_SEND_FLAG  1
#define DIGITAL_SEND_FLAG 1
#define MISC_SEND_FLAG    1
#define ERR_SEND_FLAG     1
#define PACKET1_FULL          0x10EF   //channel 0 to 3 plus battery
#define MSG_LEN  31   // excludes TOS header, but includes xbow header
   enum {
         PENDING = 0,
         NO_MSG = 1
   };
   enum {
         MDA300_PACKET1 = 1,
         MDA300_PACKET2 = 2,
         MDA300_PACKET3 = 3,
         MDA300_PACKET4 = 4,
         MDA300_ERR_PACKET = 0xf8
   };
   enum {
         SENSOR_ID = 0,
         PACKET_ID,
         NODE_ID,
         RESERVED,
         DATA_START
   } XPacketDataEnum;
   /* Messages Buffers */
   TOS_Msg packet[5];
   TOS_Msg uart_send_buffer, radio_send_buffer;
   TOS_MsgPtr uart_msg_ptr, radio_msg_ptr;
   TOS_Msg errMsg_uart, errMsg_radio;
```

```
    uint16_t errMsg_status;
    uint8_t next_packet;
    bool    sending_packet;
    uint16_t msg_status[5], pkt_full[5];
    char test;
        uint8_t data_t;
        uint16_t delay_time;
    int8_t record[25];
/*************************************************************************
 * Initialize the component. Initialize Leds
 *************************************************************************/
    command result_t StdControl.init() {
        call Leds.init();
        atomic {
          errMsg_status=0;
          uart_msg_ptr = &uart_send_buffer;
          radio_msg_ptr = &radio_send_buffer;
          next_packet = 1;
              data_t = 0xc0;
              delay_time = 0x0;
          sending_packet = FALSE;
        }
        msg_status[1] = 0;
        pkt_full[1] = PACKET1_FULL;
    call UARTControl.init();
    call SamplerControl.init();
    call CommControl.init();
    return SUCCESS;
    }
/** Sends a plain text error string using the text_msg board type. */
    task void send_uart_err_msg(){
        uint8_t i;
        char *errMsg = "mda300 not found";
        errMsg_status = 1 && ERR_SEND_FLAG;
```

```
    if (!errMsg_status) return;
    errMsg_uart.data[SENSOR_ID] = SENSOR_BOARD_ID;
    errMsg_uart.data[PACKET_ID] = MDA300_ERR_PACKET;
    errMsg_uart.data[NODE_ID] = TOS_LOCAL_ADDRESS;
    errMsg_uart.addr = TOS_UART_ADDR;
    errMsg_uart.type = 0;
    errMsg_uart.length = MSG_LEN; //TOSH_DATA_LENGTH;
    errMsg_uart.group = TOS_AM_GROUP;
    i = 0;
    // Copy error string
    while ((*errMsg) && (i <= MSG_LEN-1)) {
      errMsg_uart.data[DATA_START + i] = errMsg[i];
      i++;
    }

    // Copy over uart packet to radio packet (identical)
    for (i = 0; i <= MSG_LEN-1; i++) errMsg_radio.data[i] = errMsg_uart.data[i];
    call UARTSend.send(&errMsg_uart);
}
    //Read data only once
    task void ReadOnce0()
    {
        if (call Sample.getSampleOnce(0) == FAIL)
        {
            post ReadOnce0();
        }
    }
    task void ReadOnce1()
    {
        if (call Sample.getSampleOnce(1) == FAIL)
        {
            post ReadOnce1();
        }
    }
```

```
        task void ReadOnce2()
        {
                if (call Sample.getSampleOnce(2) == FAIL)
                {
                        post ReadOnce2();
                }
        }
        task void ReadOnce3()
        {
                if (call Sample.getSampleOnce(3) == FAIL)
                {
                        post ReadOnce3();
                }
        }
/*      task void ReadOnce4()
        {
                if (call Sample.getSampleOnce(4) == FAIL)
                {
                        post ReadOnce4();
                }
        }
        task void ReadOnce5()
        {
                if (call Sample.getSampleOnce(5) == FAIL)
                {
                        post ReadOnce5();
                }
        }
*/
/*****************************************************************
 * Start the component. Start the clock. Setup timer and sampling
 *****************************************************************/
   command result_t StdControl.start() {
        call UARTControl.start();
```

```
        call SamplerControl.start();
        call CommControl.start();
        //if(call PlugPlay())
        if(TRUE)
        {
           call SamplingTimer.start(TIMER_REPEAT, 30720);    //sampling every 30s
        //call SamplingTimer.start(TIMER_REPEAT, 10240);    //sampling every 10s
           //channel parameteres are irrelevent
     record[16] = call Sample.getSample(0,
BATTERY,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
           //start sampling  channels. Channels 7-10 with averaging since they are more
percise.channels 3-6 make active excitation
           record[0] = call
Sample.getSample(0,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT );
           record[1] = call
Sample.getSample(1,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT );
           record[2] = call
Sample.getSample(2,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
           record[3] = call
Sample.getSample(3,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT );
           //record[4] = call
Sample.getSample(4,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
           //record[5] = call
Sample.getSample(5,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
           //record[6] = call
Sample.getSample(6,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
           //record[7] = call
Sample.getSample(7,ANALOG,ANALOG_SAMPLING_TIME,SAMPLER_DEFAULT);
                call Leds.greenOn();
        }
        else {
           post send_uart_err_msg();
           call Leds.redOn();
        }
```

```
        return SUCCESS;
    }
/*****************************************************************
 * Stop the component.
 *****************************************************************/
    command result_t StdControl.stop() {
        call SamplerControl.stop();
        return SUCCESS;
    }
/*****************************************************************
 * Task to uart as message
 *****************************************************************/
    task void send_uart_msg(){
        uint8_t i;

        atomic sending_packet = TRUE;
        uart_msg_ptr->addr = TOS_UART_ADDR;
        uart_msg_ptr->type = 0;
        uart_msg_ptr->length = MSG_LEN;
        uart_msg_ptr->group = TOS_AM_GROUP;
        uart_msg_ptr->data[SENSOR_ID] = SENSOR_BOARD_ID;
        uart_msg_ptr->data[PACKET_ID] = next_packet;
        uart_msg_ptr->data[NODE_ID] = TOS_LOCAL_ADDRESS;
        for (i = 4; i <= MSG_LEN-1; i++)
           uart_msg_ptr->data[i] = packet[next_packet].data[i];
        call UARTSend.send(uart_msg_ptr);
    }
/*****************************************************************
 * Task to transmit radio message
 * NOTE that data payload was already copied from the corresponding UART packet
 *****************************************************************/
    task void send_radio_msg()
        {
           uint8_t i;
```

```
            radio_msg_ptr->addr = TOS_BCAST_ADDR;
            radio_msg_ptr->type = 0;
            radio_msg_ptr->length = MSG_LEN; //TOSH_DATA_LENGTH;
            radio_msg_ptr->group = TOS_AM_GROUP;
            radio_msg_ptr->data[SENSOR_ID] = SENSOR_BOARD_ID;
            radio_msg_ptr->data[PACKET_ID] = next_packet;
            radio_msg_ptr->data[NODE_ID] = TOS_LOCAL_ADDRESS;
            for (i = 4; i <= MSG_LEN-1; i++)
                    radio_msg_ptr->data[i] = packet[next_packet].data[i];
      call SendMsg.send(radio_msg_ptr);
         }
/*************************************************************************
 * Uart msg xmitted.
 * Transmit same msg over radio
 *************************************************************************/
   event result_t UARTSend.sendDone(TOS_MsgPtr msg, result_t success) {
                    uart_msg_ptr = msg;
                    //call Leds.yellowOn();
                    post send_radio_msg();
          return SUCCESS;
   }
/*************************************************************************
 * Radio msg xmitted.
 *************************************************************************/
   event result_t SendMsg.sendDone(TOS_MsgPtr msg, result_t success) {
                    radio_msg_ptr = msg;
                    call Leds.yellowToggle();
      // mark that this packet has been sent
                    atomic {
                            sending_packet = FALSE;
                    }
                    return SUCCESS;
   }
/*************************************************************************
```

```
 * Radio msg rcvd.
 * This app doesn't respond to any incoming radio msg
 * Just return
 *****************************************************************************/
   event TOS_MsgPtr ReceiveMsg.receive(TOS_MsgPtr data)  {
                return data;
   }
/*****************************************************************************
 * Uart msg rcvd.
 * This app doesn't respond to any incoming uart msg
 * Just return
 *****************************************************************************/
   event TOS_MsgPtr UARTReceive.receive(TOS_MsgPtr data) {
                return data;
   }
/**
 * Handle a single dataReady event for all MDA300 data types.
 * @author    Leah Fera, Martin Turon
 * @version   2004/3/17      leahfera    Intial revision
 * @n         2004/4/1       mturon      Improved state machine
 */
   event result_t Sample.dataReady(uint8_t channel,uint8_t channelType,uint16_t data)
         {
            switch (channelType) {
                case ANALOG:
                   switch (channel) {
                      // MSG 1 : first part of analog channels (0-6)
                      case 0:
                            if (data_t == 0x01){
                                  packet[1].data[DATA_START+14]=data & 0xff;
                                  packet[1].data[DATA_START+15]=(data >> 8) & 0xff;
                                  atomic {msg_status[1] |=0x0001;}
                            }
                            if (data_t == 0x03){
```

```
                    //packet[1].data[DATA_START+16]=data & 0xff;
                    //packet[1].data[DATA_START+17]=(data >> 8) & 0xff;
                    //atomic {msg_status[1] |=0x0010;}
            }
    break;
case 1:
            if (data_t == 0x05){
                    packet[1].data[DATA_START+2]=data & 0xff;
                    packet[1].data[DATA_START+3]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0002;}
            }
            if (data_t == 0x07){
                    packet[1].data[DATA_START+4]=data & 0xff;
                    packet[1].data[DATA_START+5]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0020;}
            }
    break;
case 2:
            if (data_t == 0x09){
                    packet[1].data[DATA_START+6]=data & 0xff;
                    packet[1].data[DATA_START+7]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0004;}
            }
            if (data_t == 0x0b){
                    packet[1].data[DATA_START+8]=data & 0xff;
                    packet[1].data[DATA_START+9]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0040;}
            }
    break;

case 3:
            if (data_t == 0x09){
                    packet[1].data[DATA_START+10]=data & 0xff;
                    packet[1].data[DATA_START+11]=(data >> 8) & 0xff;
```

```
                    atomic {msg_status[1] |=0x0008;}
                }
                if (data_t == 0x0b){
                    packet[1].data[DATA_START+12]=data & 0xff;
                    packet[1].data[DATA_START+13]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0080;}
                }
            break;
/*          case 4:
                if (data_t == 0x03){
                    packet[1].data[DATA_START+14]=data & 0xff;
                    packet[1].data[DATA_START+15]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0001;}
                }
                if (data_t == 0x01){
                    //packet[1].data[DATA_START+16]=data & 0xff;
                   //packet[1].data[DATA_START+17]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0010;}
                }
                break;
            case 5:
                if (data_t == 0x03){
                    packet[1].data[DATA_START+14]=data & 0xff;
                    packet[1].data[DATA_START+15]=(data >> 8) & 0xff;
                    atomic {msg_status[1] |=0x0001;}
                }
                if (data_t == 0x01){
                    //packet[1].data[DATA_START+16]=data & 0xff;
                   //packet[1].data[DATA_START+17]=(data >> 8) & 0xff;
                    //atomic {msg_status[1] |=0x0010;}
                }
                break;
            case 6:
                packet[1].data[DATA_START+12]=data & 0xff;
```

```
                        packet[1].data[DATA_START+13]=(data >> 8) & 0xff;
                        atomic {msg_status[1]|=0x0080;}
                        break;
*/
                    default:
                        break;
                } // case ANALOG (channel)
                break;
            case BATTERY:
                        packet[1].data[DATA_START+0]=data & 0xff;
                        packet[1].data[DATA_START+1]=(data >> 8) & 0xff;
                        atomic {msg_status[1]|=0x1000;}
                        break;
            default:
                break;
        } // switch (channelType)
        atomic {
                    if (!sending_packet){
                        if (msg_status[1] == PACKET1_FULL) {
                            msg_status[1] = 0;
                            post send_uart_msg();
                        }
                    }
        }
        return SUCCESS;
    }
/*****************************************************************
 * Timer Fired
 *****************************************************************/
  event result_t SamplingTimer.fired() {
            data_t=0x0;     //Turn on BG
            call Sample.set_digital_output(3, SET_LOW, 0x0); //turn off all lights
            call SleepTimer1.start(TIMER_ONE_SHOT,READ_DELAY);
            return SUCCESS;
```

```
}
event result_t SleepTimer1.fired() {
        data_t=0x01;    //Read BG On
        post ReadOnce0();
        call SleepTimer2.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
    }
    event result_t SleepTimer2.fired() {
        data_t=0x02;    //Turn off lights
        call SleepTimer3.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
    }
    event result_t SleepTimer3.fired() {
        data_t=0x03; //Read BG Off
        call SleepTimer4.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
    }
    event result_t SleepTimer4.fired() {
        data_t=0x04;    //Turn on IR
        post ReadOnce1();       //This is not the measurement that we need.
        call Sample.set_digital_output(3, SET_LOW, 0x04); //turn on IR
        call SleepTimer5.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
    }
    event result_t SleepTimer5.fired() {
        data_t=0x05;    //Read IR On
        post ReadOnce1();
        call SleepTimer6.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
    }
    event result_t SleepTimer6.fired() {
        data_t=0x06;    //Turn off all lights
        post ReadOnce1(); //This is not the measurement that we need.
        call Sample.set_digital_output(3, SET_LOW, 0x0); //turn off all lights
```

```
        call SleepTimer7.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
}
event result_t SleepTimer7.fired() {
        data_t=0x07;            //Read IR Off
        post ReadOnce1();
        call SleepTimer8.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
}
event result_t SleepTimer8.fired() {
        data_t=0x08;    //Turn on ORG
        post ReadOnce2(); //This is not the measurement that we need.
        post ReadOnce3(); //This is not the measurement that we need.
        call Sample.set_digital_output(3, SET_LOW, 0x08);  //turn On ORG
        call SleepTimer9.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
}
event result_t SleepTimer9.fired() {
        data_t=0x09;            //Read ORG On
        post ReadOnce2();
        post ReadOnce3();
        call SleepTimer10.start(TIMER_ONE_SHOT,READ_DELAY);

        return SUCCESS;
}
event result_t SleepTimer10.fired() {
        data_t=0x0a;    //Turn off all lights
        post ReadOnce2(); //This is not the measurement that we need.
        post ReadOnce3(); //This is not the measurement that we need.
        call Sample.set_digital_output(3, SET_LOW, 0x0); //turn off all lights
        call SleepTimer11.start(TIMER_ONE_SHOT,READ_DELAY);
        return SUCCESS;
}
event result_t SleepTimer11.fired() {
```

```
                data_t=0x0b;            //Read ORA Off
                post ReadOnce2();
                post ReadOnce3();
                return SUCCESS;
        }
}
```

Appendix B - Stargate program for data collection and transmission using AirCard

File Name: PPP.sh

```
echo Starting PPP ......
mknod /dev/ppp c 108 0
/sbin/modprobe -v ppp
pppd call ac750 &
echo Wait for 30 seconds......
sleep 30
```

File Name: cfcard.rc

```
#!/bin/sh
#cfcard.rc
#recover date from saved time and adjust it by time server via internet connection.
case $1 in
  start)
    #recover date from last shutdown time
          date --set="`cat /mnt/cf1/mydata/SavedDate`"
          date --set='+3 minutes'
          #ntpdate tock.usno.navy.mil
          ntpdate tick.ucla.edu
          ;;
  stop)
          echo "goodbye"
          ;;
esac
exit 0
```

File Name: stargate-watchdog

```
#!/bin/sh
#
export PATH=$PATH:/usr/sbin
case "$1" in
```

```
    start)
        echo -n "Starting watchdog: stargate-watchdog"
            start-stop-daemon --start --quiet -m --pidfile /var/run/stargate-watchdog.pid --
background --exec /usr/sbin/stargate-watchdog.sh
            echo "."
            ;;
    stop)
        echo -n "Stopping watchdog: stargate-watchdog"
            start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/stargate-watchdog.pid
            /usr/sbin/stargate-watchdog-stop.sh
            echo "."
        ;;
      restart)
            echo -n "Restarting watchdog: stargate-watchdog"
            start-stop-daemon --stop --quiet --oknodo --pidfile /var/run/stargate-watchdog.pid
            /usr/sbin/stargate-watchdog-stop.sh
            sleep 2
            start-stop-daemon --start --quiet -m --pidfile /var/run/stargate-watchdog.pid --
background --exec /usr/sbin/stargate-watchdog.sh
            echo "."
            ;;
      *)
        echo "Usage: /etc/init.d/stargate-watchdog {start|stop|restart}"
        exit 1
esac
exit 0
```

File Name: xfer.rc

```
#!/bin/sh
#xfer.rc
CFROOT=/mnt/cf1
DATAROOT=${CFROOT}/mydata
ETCROOT=/etc/init.d
XFERROOT=/etc/xfer
LOCAL_FILE=${DATAROOT}/data_for_ftp
```

```
LOCAL_FILE_OR=${DATAROOT}/xlisten_data
K_FILE_XL=${CFROOT}/kill_xlisten
K_FILE_FT=${CFROOT}/kill_ftp_test
K_FILE_RC=${CFROOT}/kill_xfer.rc
K_FILE_tmp=${CFROOT}/kill_tmp2
case $1 in
  start)
        ${CFROOT}/Xlisten.sh
        ${CFROOT}/do-clean-ftp.sh
        ${CFROOT}/killID.sh&
        #copy ftp file LOCAL_FILE from original file LOCAL_FILE_OR
        #cp -r $LOCAL_FILE_OR $LOCAL_FILE
        ${XFERROOT}/ftp_test_02.sh&
        ${CFROOT}/Readxlisten.sh&
        ;;
  stop)
        #kill xlisten-arm
        ps -o "%u %p %c" |grep "[[:space:]]xlisten-arm$" > $K_FILE_XL 2>&1
        grep "root" $K_FILE_XL &>/dev/null
        if [ $? -eq 0 ] ; then
                sed -e 's/xlisten-arm//g' $K_FILE_XL > $K_FILE_tmp
                sed -e 's/root/kill/g' $K_FILE_tmp > $K_FILE_XL
                chmod +rx $K_FILE_XL #give rights
                source $K_FILE_XL #run kill file
                cat $K_FILE_XL #for testing only
                >$K_FILE_XL #clean kill file
                echo kill xlisten-arm done
        else
                echo do not find xlisten-arm
        fi
        #kill ftp_test.sh
        ps -o "%u %p %c" |grep "[[:space:]]ftp_test.sh$" > $K_FILE_FT 2>&1
        grep "root" $K_FILE_FT &>/dev/null
        if [ $? -eq 0 ] ; then
```

```
                sed -e 's/ftp_test.sh//g' $K_FILE_FT > $K_FILE_tmp
                sed -e 's/root/kill/g' $K_FILE_tmp > $K_FILE_FT
                chmod +rx $K_FILE_FT #give rights
                source $K_FILE_XL #run kill file
                cat $K_FILE_FT #for testing only
                >$K_FILE_FT #clean kill file
                echo kill ftp_test.sh done
        else
                echo do not find xlisten-arm
        fi
        ;;
esac
exit 0
```

File Name: Xlisten.sh

```
#!/bin/bash
#Xlisten.sh
#start xlisten to get data from mote
CFROOT=/mnt/cf1
DATAROOT=/mnt/cf1/mydata
LOG_FILE=$DATAROOT/event.log
DATA_FILE=$DATAROOT/xlisten_data
K_FILE_tmp=$CFROOT/kill_tmp2

LD_LIBRARY_PATH=/usr/local/armpgsql/lib nice -n -20 /mnt/cf1/bin/xlisten-arm -r -q -t -1 >>
$DATA_FILE &
ps -o "%u %p %c" |grep "[[:space:]]xlisten-arm$" > $K_FILE_tmp 2>&1
grep "root" $K_FILE_tmp &>/dev/null
if [ $? -eq 0 ] ; then
        echo xlisten-arm start at `date +%D' '%T` >> $LOG_FILE
        echo xlisten-arm start at `date +%D' '%T`
else
        echo xlisten-arm fail to start at `date +%D' '%T` >> $LOG_FILE
        echo xlisten-arm fail to start at `date +%D' '%T`
fi
```

```
exit 0
```

File Name: killID.sh

```
#!/bin/sh
#killID.sh
#This script will kill a hung ftp process (hung for more than 2 minutes) so that the new ftp can start.
CFROOT=/mnt/cf1
DATAROOT=/mnt/cf1/mydata
LOG_FILE=$DATAROOT/event.log
K_FILE_01=$CFROOT/kill_01
K_FILE_02=$CFROOT/kill_02
K_FILE_tmp=$CFROOT/kill_tmp
while true
do
ps -A -o "%u %p %c" |grep "[[:space:]]ftp$" > $K_FILE_01 2>/dev/null
grep "root" $K_FILE_01 &>/dev/null
        if [ $? -eq 0 ] ; then
                sed -e 's/ftp//g' $K_FILE_01 > $K_FILE_tmp
                sed -e 's/root/kill/g' $K_FILE_tmp > $K_FILE_01
                sleep 120 #sleep 2 minutes
                ps -A -o "%u %p %c" |grep "[[:space:]]ftp$" > $K_FILE_02 2>/dev/null
                grep "root" $K_FILE_02 &>/dev/null
                if [ $? -eq 0 ] ; then
                        sed -e 's/ftp//g' $K_FILE_02 > $K_FILE_tmp
                        sed -e 's/root/kill/g' $K_FILE_tmp > $K_FILE_02
                        diff $K_FILE_01 $K_FILE_02 &>/dev/null
                        if [ $? -eq 0 ]; then
                                chmod +rx $K_FILE_01
                                source $K_FILE_01
                                cat $K_FILE_01 #for testing only
                                cat $K_FILE_02 #for testing only
                                >$K_FILE_01
                                >$K_FILE_02
                                echo kill dead ftp at `date +%D' '%T` >> $LOG_FILE
```

```
                                echo kill dead ftp at `date +%D' '%T`
                        fi
                fi
        fi
sleep 60
done
exit 0
```

File Name: Readxlisten.sh

```
#!/bin/sh
#Readxlisten.sh
#This script will show the newest incoming data on the stdout so that a Labview program can
read it from RS232 port.
DATAROOT=/mnt/cf1/mydata
LOCAL_FILE_OR=$DATAROOT/xlisten_data
LineNumber_New=$DATAROOT/NewlineNumber_rd
LineNumber_Old=$DATAROOT/OldlineNumber_rd
while true
do
        let tailrow_rd=0
        while [ $tailrow_rd -lt 2 ]; do
                sleep 1
                oldrow_rd=`cat /mnt/cf1/mydata/OldlineNumber_rd`
                wc -l $LOCAL_FILE_OR > $LineNumber_New
                newrow_rd=`sed -e's/\/mnt\/cf1\/mydata\/xlisten_data//g'
/mnt/cf1/mydata/NewlineNumber_rd`
                tailrow_rd=$(($newrow_rd-$oldrow_rd))
        done
        tail -n $tailrow_rd $LOCAL_FILE_OR > /dev/tts/0
        oldrow_rd=$(($tailrow_rd+$oldrow_rd))
        echo $oldrow_rd > $LineNumber_Old
done
```

File Name: ftp_test_02.sh

```
#!/bin/sh
#ftp_test_02.sh
```

```
DATAROOT=/mnt/cf1/mydata
XFERROOT=/etc/xfer
FTP_PROG=$XFERROOT/do-file-xfer-02.sh
LOCAL_FILE_OR=$DATAROOT/xlisten_data
LineNumber_New=$DATAROOT/NewlineNumber
while true
do
        let tailrow=0
        while [ $tailrow -lt 24 ]; do
                sleep 1
                oldrow=`cat /mnt/cf1/mydata/OldlineNumber`
                wc -l $LOCAL_FILE_OR > $LineNumber_New
        newrow=`sed -e's/\/mnt\/cf1\/mydata\/xlisten_data//g' /mnt/cf1/mydata/NewlineNumber`
                tailrow=$(($newrow-$oldrow))
        done
        #echo Please wait.....
        $FTP_PROG
        if [ $? -ne 0 ] ; then
                echo XFER FAILED
        fi
done
```

File Name: do-file-xfer-02.sh

```
#!/bin/sh
#do-file-xfer-02.sh
DATAROOT=/mnt/cf1/mydata
LOG_FILE=$DATAROOT/event.log
SCRIPTS_DIR=/etc/xfer
ETC=$SCRIPTS_DIR/etc
REMOTE=129.130.80.73
FTP_OUTPUT=/tmp/ftp.out
REMOTE_TEMP_NAME=tmp$$
LOCAL_FILE=$DATAROOT/data_for_ftp
LOCAL_FILE_OR=$DATAROOT/xlisten_data
LineNumber_New=$DATAROOT/NewlineNumber
```

```
LineNumber_Old=$DATAROOT/OldlineNumber
Iteration_File=$DATAROOT/IterationNumber
Date_File=$DATAROOT/SavedDate
REMOTE_DIR=Stargate
REMOTE_FILE=data_for_ftp.txt
#LOGIN_OK="230 Login successful"
#FILE_SENT_OK="226 File receive OK"
LOGIN_OK="^230"
FILE_SENT_OK="^226"
do_ftp () {
        ftp  -n -v $REMOTE <<EOF > $FTP_OUTPUT 2>&1
        user $USERNAME $PASSWORD
        binary
        cd $REMOTE_DIR
        put $LOCAL_FILE $REMOTE_FILE
        quit
EOF
}
do_ftp_bye () {
        ftp  -n -v $REMOTE <<EOF > $FTP_OUTPUT 2>&1
        user $USERNAME $PASSWORD
        bye
EOF
}
# Check if login was OK
check_login_ok_1 () {
   grep "$LOGIN_OK" $FTP_OUTPUT
   if [ $? -ne 0 ] ; then
        echo The First Time Login Failed at `date +%D' '%T`
                echo The First Time Login Failed at `date +%D' '%T` >> $LOG_FILE
                iteration=`cat /mnt/cf1/mydata/IterationNumber`
                sleep $iteration
                do_ftp_bye
                check_login_ok_2
```

```
   else
                echo LOGIN WAS OK
                echo "15"> $Iteration_File
   fi
}
check_login_ok_2 () {
   grep "$LOGIN_OK" $FTP_OUTPUT
   if [ $? -ne 0 ] ; then
        echo The Second Time Login Failed at `date +%D' '%T`
                echo The Second Time Login Failed at `date +%D' '%T` >> $LOG_FILE
                sleep $iteration
                do_ftp_bye
                check_login_ok_3
   else
                echo LOGIN WAS OK
                echo "15"> $Iteration_File
   fi
}
check_login_ok_3 () {
   grep "$LOGIN_OK" $FTP_OUTPUT
   if [ $? -ne 0 ] ; then
        echo The Third Time Login Failed at `date +%D' '%T`
                echo The Third Time Login Failed at `date +%D' '%T` >> $LOG_FILE
                #increase iteration number if the jam is hard to solve at this moment
                let "iteration = iteration * 2 "
                if [$iteration -gt 3600]; then
                        let "iteration = 3600 "
                fi
                echo $iteration > $Iteration_File
                #save current date time
                echo `date` > $Date_File
                echo Restart at `date +%D' '%T`
                echo Restart at `date +%D' '%T` >> $LOG_FILE
                shutdown -r now
```

```
            exit 2
    else
                echo LOGIN WAS OK
                echo "15"> $Iteration_File
    fi
}
check_xfer_ok () {
    grep "$FILE_SENT_OK" $FTP_OUTPUT
    if [ $? -ne 0 ] ; then
        echo File-Transfer Failed at `date +%D' '%T`
                echo File-Transfer Failed at `date +%D' '%T` >> $LOG_FILE
                do_ftp_bye
            exit 2
    else
                echo File Sent OK at `date +%D' '%T`
                #save old row number
                oldrow=$(($tailrow+$oldrow))
                echo $oldrow > $LineNumber_Old
        fi
}
copy_file(){
        #number check
        oldrow=`cat /mnt/cf1/mydata/OldlineNumber`
        wc -l $LOCAL_FILE_OR > $LineNumber_New
        newrow=`sed -e's/\/mnt\/cf1\/mydata\/xlisten_data//g' /mnt/cf1/mydata/NewlineNumber`
        tailrow=$(($newrow-$oldrow))
        #limit the maximum number of lines for one ftp transfer (517k data).
        if [$tailrow -gt 2500] ; then
                let "tailrow = 2500"
        fi
        #copy file
        #tail -n $(($tailrow+4)) $LOCAL_FILE_OR > $LOCAL_FILE
        tail -n $tailrow $LOCAL_FILE_OR > $LOCAL_FILE
}
```

```
# The main program
if [ -f $ETC/USERNAME ] ; then
   USERNAME=`cat $ETC/USERNAME`
else
   echo $PROG "No User-Name File"
   exit 2
fi
if [ -f $ETC/PASSWORD ] ; then
   PASSWORD=`cat $ETC/PASSWORD`
else
   echo $PROG "No Password File"
   exit 2
fi
# Do the transfer and check the results
copy_file
do_ftp
check_login_ok_1
check_xfer_ok
#echo ALL OK!
```

File Name: check_init.sh

```
#!/bin/sh
#check_init.sh
#This script check if following four processes are running, if any one of them is not working,
reboot the Stargate. The processes are:
# xlisten-arm,  stargate-watchd, ftp_test_02.sh and Readxlisten.sh
CFROOT=/mnt/cf1
DATAROOT=/mnt/cf1/mydata
LOG_FILE=$DATAROOT/event.log
CHECK_FILE=$DATAROOT/checkfile
Date_File=$DATAROOT/SavedDate
sleep 120 # give 2 minutes initial time for these four processes.
ps -A -o "%u %p %c" |grep "[[:space:]]xlisten-arm$" > $CHECK_FILE 2>/dev/null
grep "xlisten-arm" $CHECK_FILE &>/dev/null
if [ $? -ne 0 ] ; then
```

```
        #reboot stargate
        echo `date` > $Date_File        #save current date time
        echo Restart at `date +%D' '%T`
        echo Restart at `date +%D' '%T` >> $LOG_FILE
        shutdown -r now
fi
ps -A -o "%u %p %c" |grep "[[:space:]]stargate-watchd$" > $CHECK_FILE 2>/dev/null
grep "stargate-watchd" $CHECK_FILE &>/dev/null
if [ $? -ne 0 ] ; then
        #reboot stargate
        echo `date` > $Date_File        #save current date time
        echo Restart at `date +%D' '%T`
        echo Restart at `date +%D' '%T` >> $LOG_FILE
        shutdown -r now
fi
ps -A -o "%u %p %c" |grep "[[:space:]]ftp_test_02.sh$" > $CHECK_FILE 2>/dev/null
grep "ftp_test_02.sh" $CHECK_FILE &>/dev/null
if [ $? -ne 0 ] ; then
        #reboot stargate
        echo `date` > $Date_File        #save current date time
        echo Restart at `date +%D' '%T`
        echo Restart at `date +%D' '%T` >> $LOG_FILE
        shutdown -r now
fi
ps -A -o "%u %p %c" |grep "[[:space:]]Readxlisten.sh$" > $CHECK_FILE 2>/dev/null
grep "Readxlisten.sh" $CHECK_FILE &>/dev/null
if [ $? -ne 0 ] ; then
        #reboot stargate
        echo `date` > $Date_File        #save current date time
        echo Restart at `date +%D' '%T`
        echo Restart at `date +%D' '%T` >> $LOG_FILE
        shutdown -r now
fi
exit 0
```

Appendix C - Program on host for AirCard transmission

File Name: backbone.java

```
// backbone.java
// by Wei Han
// Nov 16, 2007
import java.io.*;
import java.sql.*;
class backbone
{
	static String DateFile = "c:/Inetpub/wwwroot/MissionKC/data/latest_datetime.txt";
	static String RecordFile = "c:/Inetpub/wwwroot/MissionKC/data/data_for_record.txt";
	static String AppendFile = "c:/Inetpub/wwwroot/MissionKC/data/data_for_append.txt";
	static String CountFile = "c:/Inetpub/wwwroot/MissionKC/data/counter.txt";
	static String gdFile = "c:/Inetpub/wwwroot/MissionKC/c_code/bkj_gd.exe";
	static String rendFile = "c:/Inetpub/wwwroot/MissionKC/c_code/bkj_render.exe";
	static String str,line,lineDate;
	public static void main(String[] args) throws InterruptedException
	{
		try{
			System.out.println("Backbone server start at "+DateUtils.now()+"\n");
			////get the latest datetime from database and write it to file
latest_datetime.txt
			try {
				Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
				String connString = "jdbc:odbc:DRIVER=Microsoft Access
Driver (*.mdb);DBQ=c:/Inetpub/wwwroot/MissionKC/database/Stargate.mdb;PWD=mypass";
				Connection con =
DriverManager.getConnection(connString,"Admin","bae840-05");
				// try and create a java.sql.Statement so we can run queries
				Statement s = con.createStatement();
				s.execute("exec sp_newdate");
				ResultSet rs = s.getResultSet();
```

```
                    rs.next();
                    str = rs.getString(1);
                    rs.close();
                    s.close();
                    con.close();
                    //wrtie new date to file
                  try{
                              // Create file
                              FileWriter fstream = new FileWriter(DateFile);
                    BufferedWriter out = new BufferedWriter(fstream);
                              out.write(str);
                              //Close the output stream
                              out.close();
                              fstream.close();
                  }catch (Exception e){//Catch exception if any
                    System.err.println("Error: " + e.getMessage());
                  }
                  }
                  catch (Exception e) {
                    System.out.println("Error: " + e);
                  }

                  ////start to run a C program -- bkj_render.exe, this is an endless loop program, you need to kill it if you want to stop the progam
                  try
                  {
                    Runtime rt = Runtime.getRuntime();
                    Process p = rt.exec(rendFile);
                  }
                  catch(Exception e)
                  {
                    System.out.println("Error: " + e);
                  }
                  while(true){//endless while loop
```

```
                try{
                        //read from counter.txt file
            try {
                                FileReader input = new FileReader(CountFile);
                        BufferedReader bufRead = new BufferedReader(input);
               line = bufRead.readLine();
               bufRead.close();
               input.close();
                        }catch (IOException e){
                  e.printStackTrace();
               }
                        if(line.equals("11")){
                                try {//open database
                        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                String connString = "jdbc:odbc:DRIVER=Microsoft Access Driver
(*.mdb);DBQ=c:/Inetpub/wwwroot/MissionKC/database/Stargate.mdb;PWD=mypass";
        Connection con = DriverManager.getConnection(connString,"Admin","bae840-05");
                        // try and create a java.sql.Statement so we can run queries
                                        Statement s = con.createStatement();
                        try {//open file to read new data
                        FileReader input = new FileReader(AppendFile);
                        BufferedReader bufRead = new BufferedReader(input);
                           // Read lines
                           line = bufRead.readLine();
                           while (line != null){
                                //validate raw data based on its format & length
                                if (line.startsWith("[") && line.length()>=21){
                                                        lineDate=line;
                                                line = bufRead.readLine();
                                                        if (line.length()>=82){
                        s.execute("Insert into tblOriginal (rawdata) Values('" + lineDate + "')");
                        s.execute("Insert into tblOriginal (rawdata) Values('" + line + "')");
                                                                }
                                                        }
```

```
                    line = bufRead.readLine();
                    }
   bufRead.close();
                    input.close();
                }catch (IOException e){
        e.printStackTrace();
      }
//escape 7d5d
s.execute ("exec sp_escape7d");
s.execute ("exec sp_replace7d");
s.execute ("exec sp_checkTotal");
ResultSet rs = s.getResultSet();
rs.next();
                str = rs.getString(1);
                s.execute ("exec sp_reset");
                while(!str.equals("0")){
s.execute ("exec sp_escape7d01");
s.execute ("exec sp_replace7d01");
s.execute ("exec sp_checkTotal");
rs = s.getResultSet();
rs.next();
str = rs.getString(1);
s.execute ("exec sp_reset");
                }
                //escape 7e5d
                s.execute ("exec sp_escape7e");
s.execute ("exec sp_replace7e");
s.execute ("exec sp_checkTotal");
rs = s.getResultSet();
                rs.next();
                str = rs.getString(1);
                s.execute ("exec sp_reset");
                while(!str.equals("0")){
s.execute ("exec sp_escape7e");
```

```
s.execute ("exec sp_replace7e");
s.execute ("exec sp_checkTotal");
rs = s.getResultSet();
rs.next();
                    str = rs.getString(1);
s.execute ("exec sp_reset");
                }
//data process
s.execute ("exec sp_DateData");
s.execute ("exec sp_final");
                    //get the latest date in database
                    s.execute("exec sp_newdate");
                    rs = s.getResultSet();
                    rs.next();
                    str = rs.getString(1);
                    //rs.close();
                    //get one day early data (negtive 1
means the past one day)
                    s.execute("SELECT * from tblResults
Where strDatetime between DateAdd('d', -1, '" + str + "') and #" + str + "# order by strDatetime
DESC"); // select the data from the table
                    rs = s.getResultSet(); // get any
ResultSet that came from our query
                    if (rs != null) // if rs == null, then there
is no ResultSet to view
                    {
                       try{
                            // Create file
FileWriter fstream = new FileWriter(RecordFile);
BufferedWriter out = new BufferedWriter(fstream);
                    while (rs.next()){
                out.write(rs.getString(1)+" "+rs.getString(2)+"
"+rs.getString(3)+" "+rs.getString(4)+" "+rs.getString(5)+" "+rs.getString(6)+"
"+rs.getString(7)+" "+rs.getString(8)+" "+rs.getString(9)+" "+rs.getString(10)+"
```

```
"+rs.getString(11)+" "+rs.getString(12)+" "+rs.getString(13)+" "+rs.getString(14)+"
"+rs.getString(15)+" "+rs.getString(16)+"\n");
                                                    }
                                                //Close the output stream
                                                out.close();
                                                fstream.close();
                                            }catch (Exception e){//Catch exception
if any
                                        System.err.println("Error: " + e.getMessage());
                                            }
                                            }
                                            rs.close();
                                            s.close(); // close the Statement to let the
database know we're done with it
                                            con.close(); // close the Connection to
let the database know we're done with it
                                        }
                                        catch (Exception e) {
                                            System.out.println("Error: " + e);
                                        }

                                        try
                                        {//run c program to draw pictures -- bkj_gd.exe
                                            Runtime rt = Runtime.getRuntime();
                                            Process p = rt.exec(gdFile);
                                        }
                                        catch(Exception e)
                                        {
                                            System.out.println("Error: " + e);
                                        }

                                        //set counter.txt file = 00
                                    try{
                                        FileWriter fstream = new FileWriter(CountFile);
```

```
                                        BufferedWriter out = new BufferedWriter(fstream);
                                                out.write("00");
                                                out.close();
                                                fstream.close();
                                        }catch (Exception e){//Catch exception if any
                                                System.err.println("Error: " + e.getMessage());
                                        }
                                        }//end of if line == 11
                                        //Pause for 60 seconds
                                        System.out.println(DateUtils.now()+"\n");
                                        Thread.sleep(60000);
                                }catch(Exception e){
                                System.err.println("While loop Error: " + e.getMessage());
                                }finally{
                                        Thread.sleep(10000);
                                        continue;
                                }
                        }//end of endless loop
                }catch (Exception e){//Catch exception if any
                                        System.err.println("Main Error: " + e.getMessage());
                }
    }
}
```

File Name: bkj_render.c

```
/* bkj_render.c
   by Wei Han
   Sep 10, 2007
   This program is used to read xlisten raw data, make a copy of it, convert it to engineering unit
and save it to a txt file (render.txt).
   It also interactive with a java program to manipulate raw data.
   gcc -o  bkj_render  bkj_render.c
    bkj_render.exe
*/
```

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <windows.h> //for Sleep function
#include "bkj.h"
float EUVoltage(int); //function to convert Engineering Unit
int CompareDate(int,int,int,int,int,int,int,int,int,int,int,int); //function to compare date time
main(int argc, char **argv){
        char *fname_ftp = "c:/ftproot/Stargate/data_for_ftp.txt";
        char *fname_read = "c:/Inetpub/wwwroot/MissionKC/data/data_for_read.txt";
        char *fname_append = "c:/Inetpub/wwwroot/MissionKC/data/data_for_append.txt";
        char *fname_rended = "c:/Inetpub/wwwroot/MissionKC/data/rended_data.txt";
        char *fname_date = "c:/Inetpub/wwwroot/MissionKC/data/latest_datetime.txt";
        char *fname_count = "c:/Inetpub/wwwroot/MissionKC/data/counter.txt";
        char * pch;  //receive token
        char ch;
        char cdata[7][20];
        char TempBuffer[256];
        char str[4];
        char strDate[20],straDate[num_Motes][20];
        char strMoteID[3];
        int  intMoteID, MoteID_tbl[num_Motes];
        char strBattery[5];
        char strCh[12][5];
        float fCh[num_Motes][12];
        char strCh0on[5], strCh0off[5];
        char strCh1on[5], strCh1off[5];
        char strCh2on[5], strCh2off[5];
        char strCh3on[5], strCh3off[5];
        char strCh4on[5], strCh4off[5];
        char strCh5on[5], strCh5off[5];
        long int number;
        float fBattery,fbat[num_Motes];
        float fCh0on,fCh0off,Prediction[num_Motes];
```

```
int   i, j, k;
int   counter, f_append;
int   year, month, day, hour, minute, second;
int   year_o, month_o, day_o, hour_o, minute_o, second_o;

FILE *fp_ftp, *fp_read, *fp_append, *fp_rended, *fp_date, *fp_count;
counter=0;
f_append=0;
//get lastest date from date file. (this date file was created and updated  by a java program (backbone.java))
while((fp_date = fopen(fname_date, "r")) == NULL) {
        printf("Error Opening Date File.\n");
        Sleep(100); //pause 0.1 s
}
fgets(TempBuffer, sizeof(TempBuffer), fp_date);
fflush(fp_date);
fclose(fp_date);
pch = strtok (TempBuffer,"- :");
i=0;
while (pch != NULL){
        strcpy(cdata[i],pch);
        i++;
        pch = strtok (NULL, "- :");
}
year_o = strtol(cdata[0],NULL,10);
month_o = strtol(cdata[1],NULL,10);
day_o = strtol(cdata[2],NULL,10);
hour_o = strtol(cdata[3],NULL,10);
minute_o = strtol(cdata[4],NULL,10);
second_o = strtol(cdata[5],NULL,10);
//resume data left in append file
while((fp_count = fopen(fname_count, "r")) == NULL) {
        printf("Error Opening Counter File.\n");
        Sleep(100); //pause 0.1 s
```

```
	}
	strcpy(TempBuffer,"");
	fgets(TempBuffer, sizeof(TempBuffer), fp_count);
	fflush(fp_count);
	fclose(fp_count);
	memset (str, '\0', 3);
	strncpy (str, TempBuffer, 2);

	if(strcmp(str,"01") == 0){
		counter=1;
	}
	//initial data value to zeros
	for (j=0; j<num_Motes; j++){
		MoteID_tbl[j]=0;
		for(i=0; i<12;i++){
			fCh[j][i]=0;
		}
	}
	//start to check incoming data forever
for(;;){//endless loop
		//open ftp file as binary file for reading
		while((fp_ftp = fopen(fname_ftp, "rb")) == NULL) {
			printf("Error Opening ftp File.\n");
			Sleep(100); //pause 0.1 s
		}
		//open read file as binary file for writing
		while((fp_read = fopen(fname_read, "wb")) == NULL) {
			printf("Error Opening read File.\n");
			Sleep(100); //pause 0.1 s
		}
		// copy the file from ftp to read
		while(!feof(fp_ftp)) {
			ch = fgetc(fp_ftp);
			if(ferror(fp_ftp)) {
```

```
                        printf("Error reading ftp file.\n");
                        exit(1);
                }
                if(!feof(fp_ftp)) fputc(ch, fp_read);
                if(ferror(fp_read)) {
                        printf("Error writing destination file.\n");
                        exit(1);
                }
        }
        fflush(fp_read);
        fclose(fp_read);
        fflush(fp_ftp);
        fclose(fp_ftp);  // Close the file
        while((fp_read = fopen(fname_read, "r")) == NULL) {
                printf("Error re-opening read File.\n");
                Sleep(100); //pause 0.1 s
        }
        k=0; //flag
        //reset buffers
        strcpy(TempBuffer,"");
        //from following while loop, we get the last line of data rended, and this is
enough for display purpose
        while( fgets(TempBuffer, sizeof(TempBuffer), fp_read) != NULL ) {
                //check date
                memset (str, '\0', 4);
                strncpy (str, TempBuffer, 3);
                if(strcmp(str,"[20") == 0){
                        memcpy(strDate,TempBuffer+1,19);
                        //Check to see if it is a new date
                        if(k==0){//this k flag make sure it checking the datetime for the
first date line of the data_for_read.txt only
                                k=1;
                                //if date time equal or less then, do nothing
                                pch = strtok (strDate,"/ :");
```

```
                                        i=0;
                                        while (pch != NULL){
                                                strcpy(cdata[i],pch);
                                                i++;
                                                pch = strtok (NULL, "/ :");
                                        }
                                        year = strtol(cdata[0],NULL,10); //convert char to
integer
                                        month = strtol(cdata[1],NULL,10);
                                        day = strtol(cdata[2],NULL,10);
                                        hour = strtol(cdata[3],NULL,10);
                                        minute = strtol(cdata[4],NULL,10);
                                        second = strtol(cdata[5],NULL,10);
                                        if(!CompareDate(year, month , day, hour, minute,
second, year_o, month_o, day_o, hour_o, minute_o, second_o))
                                                break;
                                        year_o = year;
                                        month_o = month;
                                        day_o = day;
                                        hour_o = hour;
                                        minute_o = minute;
                                        second_o = second;
                                        f_append=1; //flag for append new data
                                }
                        }else if (strcmp(str,"7e4") ==0){
                                //escape
                                escape(TempBuffer);
                                //mote ID
                                memset(strMoteID,'\0',3);
                                memcpy(strMoteID, TempBuffer+18,2);
                                intMoteID = strtol(strMoteID,NULL,16); //hex to dec
                                //battery
                                memset(strBattery,'\0',5);
                                memcpy(strBattery, TempBuffer+22,4);
```

```
                    interchange(strBattery);
                    number = strtol (strBattery,NULL,16);
                    fBattery=number/1000.0;
                    //get channel values
                    for (i=0; i<12; i++){
                            memset(strCh[i],'\0',5);
                            memcpy(strCh[i],TempBuffer+26+i*4,4);
                            interchange(strCh[i]);
                            number = strtol(strCh[i],NULL,16);
                            for (j=0; j<num_Motes;j++){//seperate data by Mote ID
                                    if (intMoteID==(j+1)){
                                            fCh[j][i]=EUVoltage(number);
                                            fbat[j]=fBattery;
                                            strcpy(straDate[j],strDate);
                                            MoteID_tbl[j]=1;
                                    }
                            }
                    }
            }
    }//end of while loop
    fflush(fp_read);
    fclose(fp_read);
    if(f_append){
            //delete data in rend file
            while((fp_rended = fopen(fname_rended, "w")) == NULL) {
                    printf("Error Opening rended File.\n");
                    Sleep(100); //pause 0.1 s
            }
            fprintf(fp_rended,"");
            fflush(fp_rended);
            fclose(fp_rended);
            //append data to rend file
            while((fp_rended = fopen(fname_rended, "a")) == NULL) {
                    printf("Error re-opening rended File.\n");
```

```
                Sleep(100); //pause 0.1 s
            }
            for (i=0; i<num_Motes; i++){
                if(MoteID_tbl[i]){
                    switch(i){
                        case 0: fCh[i][6]=fCh[i][6]*10; //use diffrent
paramater of voltage divider, 10 is the ideal value
                                if (fCh[i][4]<=2.49){

        Prediction[i]=((Mote1_IR45_A*(fCh[i][0]-
fCh[i][1])+Mote1_IR45_B)+(Mote1_ORA45_A*(fCh[i][2]-
fCh[i][3])+Mote1_ORA45_B)+(Mote1_ORA180_A*(fCh[i][4]-
fCh[i][5])+Mote1_ORA180_B))/3;
                                }else{Prediction[i]=99999;}
                                    break;
                        case 1: fCh[i][6]=fCh[i][6]*10;
                                if (fCh[i][4]<=2.49){

        Prediction[i]=((Mote2_IR45_A*(fCh[i][0]-
fCh[i][1])+Mote2_IR45_B)+(Mote2_ORA45_A*(fCh[i][2]-
fCh[i][3])+Mote2_ORA45_B)+(Mote2_ORA180_A*(fCh[i][4]-
fCh[i][5])+Mote2_ORA180_B))/3;
                                }else{Prediction[i]=99999;}
                                break;
                    }
                    if(Prediction[i]==99999){
                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %s %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],"N/A",straDate[i]);
                    }else if (Prediction[i]<=0.00){
                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %s %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],"0.00",straDate[i]);
                    }else if (Prediction[i]<10.00){
```

```
                                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %3.2f %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],Prediction[i],straDate[i]);
                                }else if (Prediction[i]<100.00){
                                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %4.2f %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],Prediction[i],straDate[i]);
                                }else if (Prediction[i]<1000.00){
                                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %5.2f %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],Prediction[i],straDate[i]);
                                }else {
                                        fprintf(fp_rended, "%d %3.2f %5.4f %5.4f
%5.4f %4.2f %6.2f %s\n", i+1, fbat[i], fCh[i][0]-fCh[i][1], fCh[i][2]-fCh[i][3],fCh[i][4]-
fCh[i][5],fCh[i][6],Prediction[i],straDate[i]);
                                }
                                fflush(fp_rended);
                        }
                }
                fflush(fp_rended);
                fclose(fp_rended);
                //copy to create a new append file or append to an existing append file
                while((fp_read = fopen(fname_read, "r")) == NULL) {
                        printf("Error the third time opening read File.\n");
                        Sleep(100); //pause 0.1 s
                }
                if (counter==0){//copy file
                        //open append file for binary writing, this will erase old data in
append file
                        while((fp_append = fopen(fname_append, "wb")) == NULL) {
                                printf("Error Opening Append File.\n");
                                Sleep(100); //pause 0.1 s
                        }
                }else{//append file
```

```
		//open read file for appending, this will keep old data
		while((fp_append = fopen(fname_append, "a")) == NULL) {
			printf("Error Opening append File.\n");
			Sleep(100); //pause 0.1 s
		}
}
// copy the file from read to append
while(!feof(fp_read)) {
		ch = fgetc(fp_read);
		if(ferror(fp_read)) {
			printf("Error reading read file.\n");
			exit(1);
		}
		if(!feof(fp_read)) fputc(ch, fp_append);
		if(ferror(fp_append)) {
			printf("Error writing append file.\n");
			exit(1);
		}
}
f_append=0;
counter++;
fflush(fp_append);
fclose(fp_append);
//read and set resuming flag (resume from computer shutdown or restart)
while((fp_count = fopen(fname_count, "r")) == NULL) {
		printf("Error re-opening count File.\n");
		Sleep(100); //pause 0.1 s
}
strcpy(TempBuffer,"");
fgets(TempBuffer, sizeof(TempBuffer), fp_count);
fflush(fp_count);
fclose(fp_count);
memset (str, '\0', 3);
strncpy (str, TempBuffer, 2);
```

```
		if(strcmp(str,"00") == 0){
			//write resuming flag
			while((fp_count = fopen(fname_count, "w")) == NULL) {
				printf("Error 3th-opening count File.\n");
				Sleep(100); //pause 0.1 s
			}
			fprintf(fp_count, "%s", "01");
			fflush(fp_count);
			fclose(fp_count);
		}
		fflush(fp_read);
		fclose(fp_read);
	}
	if (counter>=1){//wrtie flag to let java program insert data into DB
		while((fp_count = fopen(fname_count, "r")) == NULL) {
			printf("Error 6th-Opening count File.\n");
			Sleep(100); //pause 0.1 s
		}
		strcpy(TempBuffer,"");
		fgets(TempBuffer, sizeof(TempBuffer), fp_count);
		fflush(fp_count);
		fclose(fp_count);
		memset (str, '\0', 3);
		strncpy (str, TempBuffer, 2);
		if(strcmp(str,"11") != 0){// reset counter when counter file is not 11
			while((fp_count = fopen(fname_count, "w")) == NULL) {
				printf("Error 7th-Opening count File.\n");
				Sleep(100); //pause 0.1 s
			}
			fprintf(fp_count, "%s", "11");
			fflush(fp_count);
			fclose(fp_count);
			counter=0;
		}
```

```
                }
                Sleep(30000); //pause 30 s
        } //end of endless loop
}
escape(char strLine[])
{
        char ch1[1];
        char ch2[1];
        char strBlk[2];
        char NewLine[256];
        int  bhead =0;
        int i,j;
        memset (NewLine,'\0',256);
        for (i =0; i<strlen(strLine); i=i+2){
                strcpy(ch1,"a\0");
                memmove(ch1,strLine+i,1);
                strcpy(strBlk,ch1);
                if(i+1<strlen(strLine)){
                        strcpy(ch2,"a\0");
                        memmove(ch2,strLine+i+1,1);
                        strcat(strBlk,ch2);
                        if(strcmp(strBlk,"7d")==0){
                                bhead=1;
                        }else if(bhead==1 && strcmp(strBlk,"5e")==0){
                                strcpy(strBlk,"7e");
                                memset(NewLine+strlen(NewLine)-2,'\0',2);
                                bhead=0;
                        }else if(bhead==1 && strcmp(strBlk,"5d")==0){
                                strcpy(strBlk,"");
                                bhead=0;
                        }else{
                                bhead=0;
                        }
                }else{
```

```
                    strcpy(strBlk,ch1);
                }
                strcat(NewLine,strBlk);
        }
        strcpy(strLine,NewLine);
        return;
}
interchange(char str[]){
        char strtemp[5];
        memset(strtemp,'\0',5);
        memcpy(strtemp,str+2,2);
        strncat(strtemp,str,2);
        strcpy(str,strtemp);
        return;
}
float EUVoltage(int num){//Engineering Unit for Voltage
        if(num==0)
                return 0.0;
        return num*2.5/4096.0;
}
int CompareDate(int y,int m ,int d,int h,int mm,int s, int y_o,int m_o,int d_o,int h_o,int mm_o,int
s_o){//compare two datetimes
        if (y>y_o)
                return 1;
        if (y<y_o)
                return 0;
        if (m>m_o)
                return 1;
        if (m<m_o)
                return 0;
        if (d>d_o)
                return 1;
        if (d<d_o)
                return 0;
```

```
        if (h>h_o)
                return 1;
        if (h<h_o)
                return 0;
        if (mm>mm_o)
                return 1;
        if (mm<mm_o)
                return 0;
        if (s>s_o)
                return 1;
        if (s<s_o)
                return 0;
        return 0; //for everything are same
}
```

File Name: bkj_gd.c

```
/***
 * bkj_gd.c  - creating images with the GD library used for display on the Internet
  * Compiling and running this file looks like this.
 *   $ gcc -o  bkj_gd  bkj_gd.c  -lgd
 *   $ ./bkj_gd
 * Note that the "-lgd" means "include the gd library".
 * Wei Han, Aug 30, 2007
 ***********************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
#include <c:/GnuWin32/include/gd.h>
#include "bkj_gd.h"
#include "bkj.h"
// See the bottom of this code for a discussion of some output possibilities.
char *filename_mote_01 = "C:/Inetpub/wwwroot/MissionKC/images/mote01.png";
char *filename_mote_02 = "C:/Inetpub/wwwroot/MissionKC/images/mote02.png";
char *filename_12V = "C:/Inetpub/wwwroot/MissionKC/images/Battery_12V.png";
```

```
char *filename_PreCon = "C:/Inetpub/wwwroot/MissionKC/images/PredConce.png";
char *fileRead = "C:/Inetpub/wwwroot/MissionKC/data/data_for_record.txt";
int main(){
	FILE		*outfile_mote[num_Motes], *outfile_bat, *outfile_precon, *fp;	// defined in stdio
	char		 TempBuffer[256];
	char		* pch;
	char		cdata[20][20];
	gdImagePtr	image_mote[num_Motes], image_bat, image_precon;		// a GD image object
	int		white[num_Motes], blue[num_Motes], red[num_Motes], green[num_Motes], black[num_Motes], goldenrod[num_Motes], DarkOliveGreen4[num_Motes], DarkViolet[num_Motes], DarkOrange2[num_Motes], SlateBlue[num_Motes], gray84[num_Motes], gray[num_Motes][255];		// some GD colors
	int		white_batt, blue_batt, red_batt, green_batt, black_batt, goldenrod_batt, DarkOliveGreen4_batt, DarkViolet_batt, DarkOrange2_batt, SlateBlue_batt, gray84_batt, gray_batt[255];		// some GD colors
	int		white_precon, blue_precon, red_precon, green_precon, black_precon, goldenrod_precon, DarkOliveGreen4_precon, DarkViolet_precon, DarkOrange2_precon, SlateBlue_precon, gray84_precon, gray_precon[255];		// some GD colors
	int			colorCode;
	int			i, j, x, y;
	int			hours,minutes,seconds;
	int			xzero, xall, xall_old, xflag=0, oldflag=0;
	int			eflag_mote, eflag_batt, eflag_precon[num_Motes];
	int			xaxile, yaxile;
	float			fvalue[num_Motes][13], fvalue_old[num_Motes][13];
	int			intMoteID=0;
	int			IR45[num_Motes], IR45_old[num_Motes], ORA45[num_Motes], ORA45_old[num_Motes], ORA180[num_Motes], ORA180_old[num_Motes];
	int			PreCon[num_Motes], PreCon_old[num_Motes];
	int			Bat_12V, Bat_12V_old;
	//Open record file for reading
```

```
	if((fp = fopen(fileRead, "r")) == NULL) {
		printf("Error Opening File.\n");
		return(1);
	}
	//inital the values to zeros
	for (i=0;i<num_Motes;i++){
		eflag_precon[i]=0;
	}
	eflag_mote=0;
	eflag_batt=0;
	for (i=0;i<num_Motes;i++){
		image_mote[i] = gdImageCreate(IMAGE_WIDTH, IMAGE_HEIGHT);
	}
	image_bat = gdImageCreate(IMAGE_WIDTH, IMAGE_HEIGHT);
	image_precon = gdImageCreate(IMAGE_WIDTH, IMAGE_HEIGHT);
	//define colors for mote
	for (j=0;j<num_Motes;j++){
		white[j] = gdImageColorAllocate(image_mote[j], 255,255,255);    //  1st is
background
		blue[j]  = gdImageColorAllocate(image_mote[j], 0,0,255);        //  (red,green,blue)
		red[j] = gdImageColorAllocate(image_mote[j], 255,0,0);
		green[j] = gdImageColorAllocate(image_mote[j], 0,100,0);
		black[j] = gdImageColorAllocate(image_mote[j], 0,0,0);
		gray84[j] = gdImageColorAllocate(image_mote[j], 209, 209, 209);
		goldenrod[j] = gdImageColorAllocate(image_mote[j], 218,165,32);
		DarkOliveGreen4[j] = gdImageColorAllocate(image_mote[0], 110,139,61);
		DarkViolet[j] = gdImageColorAllocate(image_mote[j], 148,0,211);
		DarkOrange2[j] = gdImageColorAllocate(image_mote[j], 238, 118, 0);
		for (i=0; i<255; i++){
			gray[j][i] = gdImageColorAllocate(image_mote[j], i,i,i);
		}
	}
	//define colors for battery 12 V
	white_batt = gdImageColorAllocate(image_bat, 255,255,255);    //  1st is background
```

```
        blue_batt  = gdImageColorAllocate(image_bat, 0,0,255);       //  (red,green,blue)
        red_batt = gdImageColorAllocate(image_bat, 255,0,0);
        green_batt = gdImageColorAllocate(image_bat, 0,100,0);
        black_batt = gdImageColorAllocate(image_bat, 0,0,0);
        gray84_batt = gdImageColorAllocate(image_bat, 209, 209, 209);
        DarkOliveGreen4_batt = gdImageColorAllocate(image_bat, 110,139,61);
        DarkViolet_batt = gdImageColorAllocate(image_bat, 148,0,211);
        DarkOrange2_batt = gdImageColorAllocate(image_bat, 238, 118, 0);
        SlateBlue_batt = gdImageColorAllocate(image_bat, 132, 112, 255);
        for (i=0; i<255; i++){
                gray_batt[i] = gdImageColorAllocate(image_bat, i,i,i);
        }

        //define colors for predicted concentration
        white_precon = gdImageColorAllocate(image_precon, 255,255,255);    //  1st is
background
        blue_precon  = gdImageColorAllocate(image_precon, 0,0,255);       //  (red,green,blue)
        red_precon = gdImageColorAllocate(image_precon, 255,0,0);
        green_precon = gdImageColorAllocate(image_precon, 0,100,0);
        black_precon = gdImageColorAllocate(image_precon, 0,0,0);
        gray84_precon = gdImageColorAllocate(image_precon, 209, 209, 209);
        DarkOliveGreen4_precon = gdImageColorAllocate(image_precon, 110,139,61);
        DarkViolet_precon = gdImageColorAllocate(image_precon, 148,0,211);
        DarkOrange2_precon = gdImageColorAllocate(image_precon, 238, 118, 0);
        SlateBlue_precon = gdImageColorAllocate(image_precon, 132, 112, 255);
        for (i=0; i<255; i++){
                gray_precon[i] = gdImageColorAllocate(image_precon, i,i,i);
        }
        //start a loop to read record file one line at each time
        strcpy(TempBuffer,"");
        while( fgets(TempBuffer, 256, fp) != NULL ){
                pch = strtok (TempBuffer,"- :");
                i=0;
                while (pch != NULL){
```

```
                    strcpy(cdata[i],pch);
                    if (i==18){//check date time, 18 is the index of the last item from one
line
                        hours = strtol(cdata[16],NULL,10);
                        minutes = strtol(cdata[17],NULL,10); //hex to dec
                        seconds = strtol(cdata[18],NULL,10);
                        seconds = hours*3600+minutes*60+seconds;
                            if (xflag==0){//this flag make sure to draw frame only
once
                                xzero=720-seconds/120;
                                xflag=1;

                                for(i=0;i<num_Motes;i++){
                                    //draw horizontal lines for motexx.png
                                    gdImageLine(image_mote[i],
RIGHT,yscale+BORDER, LEFT,yscale+BORDER,  gray84[i]);
                                    gdImageLine(image_mote[i],
RIGHT,2*yscale+BORDER, LEFT,2*yscale+BORDER,  gray84[i]);

                                    //draw frame for all motes
                                    gdImageLine(image_mote[i],
LEFT,TOP,     RIGHT,TOP,    blue[i]); // draw lines in image
                                    gdImageLine(image_mote[i],
RIGHT,TOP,    RIGHT,BOTTOM, blue[i]); //  +------------------+
                                    gdImageLine(image_mote[i],
RIGHT,BOTTOM, LEFT,BOTTOM,  blue[i]); //  |0,0       WIDTH,0|
                                    gdImageLine(image_mote[i],
LEFT,BOTTOM,  LEFT,TOP, blue[i]); //  |0,HEIGHT
                                }
                                //draw horizontal lines for Battery_12V.png
                                for (i=1; i<12;i++){
                                    gdImageLine(image_bat,
RIGHT,i*10+BORDER, LEFT,i*10+BORDER, gray84_batt);
                                }
```

```
                                        //draw frame for Battery_12V.png
                                        gdImageLine(image_bat, LEFT,TOP,
RIGHT,TOP,    blue_batt); // draw lines in image
                                        gdImageLine(image_bat, RIGHT,TOP,
RIGHT,BOTTOM, blue_batt); //  +-----------------+
                                        gdImageLine(image_bat, RIGHT,BOTTOM,
LEFT,BOTTOM,  blue_batt); //  |0,0       WIDTH,0|
                                        gdImageLine(image_bat, LEFT,BOTTOM,
LEFT,TOP, blue_batt); //  |0,HEIGHT

                                        //draw horizontal lines for PredConce.png
                                        for (i=1; i<10;i++){
                                                gdImageLine(image_precon,
RIGHT,i*12+BORDER, LEFT,i*12+BORDER, gray84_precon);
                                        }
                                        //draw frame for PredConce.png
                                        gdImageLine(image_precon, LEFT,TOP,
RIGHT,TOP,    blue_precon); // draw lines in image
                                        gdImageLine(image_precon, RIGHT,TOP,
RIGHT,BOTTOM, blue_precon); //  +-----------------+
                                        gdImageLine(image_precon, RIGHT,BOTTOM,
LEFT,BOTTOM,  blue_precon); //  |0,0       WIDTH,0|
                                        gdImageLine(image_precon, LEFT,BOTTOM,
LEFT,TOP, blue_precon); //  |0,HEIGHT
                                }
                        xall = seconds/120;
                        intMoteID = (int)atof(cdata[1])-1; //make mote ID start from 0
                        //get value
                        for (j=0; j<13;j++){
                                fvalue[intMoteID][j]=atof(cdata[j+2]);
                        }
                        if (xflag==2){//draw value, we have 150-30=120 pixles, 40
means 40 pixles represent 1 volt
```

```
                                        IR45[intMoteID] =
fringe((int)floor(40*(fvalue[intMoteID][0]-fvalue[intMoteID][1])));
                                        IR45_old[intMoteID]
=fringe((int)floor(40*(fvalue_old[intMoteID][0]-fvalue_old[intMoteID][1])));
                                        ORA45[intMoteID] =
fringe((int)floor(40*(fvalue[intMoteID][2]-fvalue[intMoteID][3])));
                                        ORA45_old[intMoteID] =
fringe((int)floor(40*(fvalue_old[intMoteID][2]-fvalue_old[intMoteID][3])));
                                        ORA180[intMoteID] =
fringe((int)floor(40*(fvalue[intMoteID][4]-fvalue[intMoteID][5])));
                                        ORA180_old[intMoteID] =
fringe((int)floor(40*(fvalue_old[intMoteID][4]-fvalue_old[intMoteID][5])));
                                        //calculate predicted concentration
                                        //draw value, we have 150-30=120 pixles, 83.33 means 1
pixles represent 83.33 mg/L
                                        switch (intMoteID){
                                                case
0:PreCon[0]=fringe((int)floor(((Mote1_IR45_A*(fvalue[0][0]-
fvalue[0][1])+Mote1_IR45_B)+(Mote1_ORA45_A*(fvalue[0][2]-
fvalue[0][3])+Mote1_ORA45_B)+(Mote1_ORA180_A*(fvalue[0][4]-
fvalue[0][5])+Mote1_ORA180_B))/(3*83.33)));
PreCon_old[0]=fringe((int)floor(((Mote1_IR45_A*(fvalue_old[0][0]-
fvalue_old[0][1])+Mote1_IR45_B)+(Mote1_ORA45_A*(fvalue_old[0][2]-
fvalue_old[0][3])+Mote1_ORA45_B)+(Mote1_ORA180_A*(fvalue_old[0][4]-
fvalue_old[0][5])+Mote1_ORA180_B))/(3*83.33)));
                                                        break;
                                                case
1:PreCon[1]=fringe((int)floor(((Mote2_IR45_A*(fvalue[1][0]-
fvalue[1][1])+Mote2_IR45_B)+(Mote2_ORA45_A*(fvalue[1][2]-
fvalue[1][3])+Mote2_ORA45_B)+(Mote2_ORA180_A*(fvalue[1][4]-
fvalue[1][5])+Mote2_ORA180_B))/(3*83.33)));
PreCon_old[1]=fringe((int)floor(((Mote2_IR45_A*(fvalue_old[1][0]-
fvalue_old[1][1])+Mote2_IR45_B)+(Mote2_ORA45_A*(fvalue_old[1][2]-
```

```
fvalue_old[1][3])+Mote2_ORA45_B)+(Mote2_ORA180_A*(fvalue_old[1][4]-
fvalue_old[1][5])+Mote2_ORA180_B))/(3*83.33)));
                                                    break;
                                        }
                                        //12V battery (choose which mote get the 12V battery
data),
                                        //10 pixles represent 1 volt, the second 10 is the
parameter for voltage divider, it should be a value afer measure your voltage divider
                                        Bat_12V = (int)floor(10*10*fvalue[0][6])-20;
                                        Bat_12V_old = (int)floor(10*10*fvalue_old[0][6])-20;

                                        if(oldflag>3){//do not draw anything when xall_old is
null, the first two iternations should be ignored
                                                //draw the value
                                                eflag_mote =
drawValue(image_mote[intMoteID], xall, xall_old, xzero, IR45[intMoteID],
IR45_old[intMoteID], DarkViolet[intMoteID], eflag_mote);
                                                eflag_mote =
drawValue(image_mote[intMoteID], xall, xall_old, xzero, ORA45[intMoteID],
ORA45_old[intMoteID], green[intMoteID], eflag_mote);
                                                eflag_mote =
drawValue(image_mote[intMoteID], xall, xall_old, xzero, ORA180[intMoteID],
ORA180_old[intMoteID], DarkOrange2[intMoteID], eflag_mote);
                                                if(fvalue[intMoteID][4]<2.49){//check if
ORA180 is out of water which means no rain, and hence no need to draw the prediction values
                                                        switch(intMoteID){
                                                                case 0:colorCode=green_precon;
                                                                        break;
                                                                case
1:colorCode=DarkOrange2_precon;
                                                                        break;
                                                        }
```

```
                                                    eflag_precon[intMoteID] =
drawValue(image_precon, xall, xall_old, xzero, PreCon[intMoteID], PreCon_old[intMoteID],
colorCode, eflag_precon[intMoteID]);
                                                }
                                                if(intMoteID==0)
                                                    eflag_batt = drawValue(image_bat, xall,
xall_old, xzero, Bat_12V, Bat_12V_old, DarkOliveGreen4_batt, eflag_batt);
                                            }
                                    }
                                    xflag = 2;
                                    xall_old = xall;
                                    oldflag++;
                                    for(j=0; j<13;j++){
                                        fvalue_old[intMoteID][j] = fvalue[intMoteID][j];
                                    }
                            }
                            i++;
                            pch = strtok (NULL, " :");
                    }
            }
    //draw vertical lines for all motes on x-coordinate
            for(j=0;j<num_Motes;j++){
                    for (i=0; i<25; i++){
                            gdImageLine(image_mote[j],
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM+4,
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM, blue[j]);
                    }
                    gdImageLine(image_mote[j], margin(xzero)+EDGE+BORDER,TOP,
margin(xzero)+EDGE+BORDER,BOTTOM, red[j]);
            }
    //draw vertical lines for Battery_12V.png
            for (i=0; i<25; i++){
```

```
                gdImageLine(image_bat,
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM+4,
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM, blue_batt);
        }
        gdImageLine(image_bat, margin(xzero)+EDGE+BORDER,TOP,
margin(xzero)+EDGE+BORDER,BOTTOM, red_batt);
   //draw vertical lines for PredConce.png
        for (i=0; i<25; i++){
                gdImageLine(image_precon,
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM+4,
margin(xzero+i*xscale)+EDGE+BORDER,BOTTOM, blue_precon);
        }
        gdImageLine(image_precon, margin(xzero)+EDGE+BORDER,TOP,
margin(xzero)+EDGE+BORDER,BOTTOM, red_precon);
        //draw hours
        for (j=0; j<num_Motes; j++){
                //draw char zero
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
gdImageSetPixel(image_mote[j], y+LEFT+margin(xzero)-2, x+BOTTOM+6, gray[j][zero[x][y]]);
                                if (j==0)
gdImageSetPixel(image_bat, y+LEFT+margin(xzero)-2, x+BOTTOM+6, gray_batt[zero[x][y]]);
gdImageSetPixel(image_precon, y+LEFT+margin(xzero)-2, x+BOTTOM+6,
gray_precon[zero[x][y]]);
                        }
                }
                //draw char one
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+xscale)-2, x+BOTTOM+6, gray[j][one[x][y]]);
                                if (j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+xscale)-2, x+BOTTOM+6, gray_batt[one[x][y]]);
```

```
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+xscale)-2, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                //draw char two
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+2*xscale)-2, x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+2*xscale)-2, x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+2*xscale)-2, x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                //draw char three
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+3*xscale)-2, x+BOTTOM+6, gray[j][three[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+3*xscale)-2, x+BOTTOM+6, gray_batt[three[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+3*xscale)-2, x+BOTTOM+6, gray_precon[three[x][y]]);
                        }
                }
                //draw char four
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+4*xscale)-2, x+BOTTOM+6, gray[j][four[x][y]]);
                                if(j==0)
```

```
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+4*xscale)-2, x+BOTTOM+6, gray_batt[four[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+4*xscale)-2, x+BOTTOM+6, gray_precon[four[x][y]]);
                        }
                }
                //draw char five
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+5*xscale)-2, x+BOTTOM+6, gray[j][five[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+5*xscale)-2, x+BOTTOM+6, gray_batt[five[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+5*xscale)-2, x+BOTTOM+6, gray_precon[five[x][y]]);
                        }
                }
                //draw char six
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+6*xscale)-2, x+BOTTOM+6, gray[j][six[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+6*xscale)-2, x+BOTTOM+6, gray_batt[six[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+6*xscale)-2, x+BOTTOM+6, gray_precon[six[x][y]]);
                        }
                }
                //draw char seven
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
```

```
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+7*xscale)-2, x+BOTTOM+6, gray[j][seven[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+7*xscale)-2, x+BOTTOM+6, gray_batt[seven[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+7*xscale)-2, x+BOTTOM+6, gray_precon[seven[x][y]]);
                        }
                }
                //draw char eight
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+8*xscale)-2, x+BOTTOM+6, gray[j][eight[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+8*xscale)-2, x+BOTTOM+6, gray_batt[eight[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+8*xscale)-2, x+BOTTOM+6, gray_precon[eight[x][y]]);
                        }
                }
                //draw char nine
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+9*xscale-2), x+BOTTOM+6, gray[j][nine[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+9*xscale-2), x+BOTTOM+6, gray_batt[nine[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+9*xscale-2), x+BOTTOM+6, gray_precon[nine[x][y]]);
                        }
                }
                //draw char ten
```

```
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+10*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+10*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+10*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+10*xscale), x+BOTTOM+6, gray[j][zero[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+10*xscale), x+BOTTOM+6, gray_batt[zero[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+10*xscale), x+BOTTOM+6, gray_precon[zero[x][y]]);
                        }
                }
                //draw char eleven
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+11*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+11*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+11*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
```

```
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+11*xscale), x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+11*xscale), x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+11*xscale), x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                //draw char twelve
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+12*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+12*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+12*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+12*xscale), x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+12*xscale), x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+12*xscale), x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
```

```
                //draw char thirteen
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+13*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+13*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+13*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+13*xscale), x+BOTTOM+6, gray[j][three[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+13*xscale), x+BOTTOM+6, gray_batt[three[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+13*xscale), x+BOTTOM+6, gray_precon[three[x][y]]);
                        }
                }
                //draw char fourteen
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+14*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+14*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+14*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
```

```
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+14*xscale), x+BOTTOM+6, gray[j][four[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+14*xscale), x+BOTTOM+6, gray_batt[four[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+14*xscale), x+BOTTOM+6, gray_precon[four[x][y]]);
                        }
                }
                //draw char fifteen
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+15*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+15*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+15*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+15*xscale), x+BOTTOM+6, gray[j][five[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+15*xscale), x+BOTTOM+6, gray_batt[five[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+15*xscale), x+BOTTOM+6, gray_precon[five[x][y]]);
                        }
```

```
		}
		//draw char sixteen
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+16*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
				if(j==0)
					gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+16*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
					gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+16*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+16*xscale), x+BOTTOM+6, gray[j][six[x][y]]);
				if(j==0)
					gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+16*xscale), x+BOTTOM+6, gray_batt[six[x][y]]);
					gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+16*xscale), x+BOTTOM+6, gray_precon[six[x][y]]);
			}
		}
		//draw char seventeen
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+17*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
				if(j==0)
					gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+17*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
					gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+17*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
```

```
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                        gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+17*xscale), x+BOTTOM+6, gray[j][seven[x][y]]);
                        if(j==0)
                                gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+17*xscale), x+BOTTOM+6, gray_batt[seven[x][y]]);
                                gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+17*xscale), x+BOTTOM+6, gray_precon[seven[x][y]]);
                }
        }
        //draw char eighteen
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                        gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+18*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                        if(j==0)
                                gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+18*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                                gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+18*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                        gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+18*xscale), x+BOTTOM+6, gray[j][eight[x][y]]);
                        if(j==0)
                                gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+18*xscale), x+BOTTOM+6, gray_batt[eight[x][y]]);
                                gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+18*xscale), x+BOTTOM+6, gray_precon[eight[x][y]]);
```

```
                }
            }
            //draw char nineteen
            for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                    gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+19*xscale)-5, x+BOTTOM+6, gray[j][one[x][y]]);
                    if(j==0)
                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+19*xscale)-5, x+BOTTOM+6, gray_batt[one[x][y]]);
                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+19*xscale)-5, x+BOTTOM+6, gray_precon[one[x][y]]);
                }
            }
            for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                    gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+19*xscale), x+BOTTOM+6, gray[j][nine[x][y]]);
                    if(j==0)
                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+19*xscale), x+BOTTOM+6, gray_batt[nine[x][y]]);
                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+19*xscale), x+BOTTOM+6, gray_precon[nine[x][y]]);
                }
            }
            //draw char twenty
            for (x=0; x<8; x++){
                for (y=0; y<5; y++){
                    gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+20*xscale)-5, x+BOTTOM+6, gray[j][two[x][y]]);
                    if(j==0)
                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+20*xscale)-5, x+BOTTOM+6, gray_batt[two[x][y]]);
```

```
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+20*xscale)-5, x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+20*xscale), x+BOTTOM+6, gray[j][zero[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+20*xscale), x+BOTTOM+6, gray_batt[zero[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+20*xscale), x+BOTTOM+6, gray_precon[zero[x][y]]);
                        }
                }
                //draw char twenty-one
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+21*xscale)-5, x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+21*xscale)-5, x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+21*xscale)-5, x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+21*xscale), x+BOTTOM+6, gray[j][one[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+21*xscale), x+BOTTOM+6, gray_batt[one[x][y]]);
```

```
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+21*xscale), x+BOTTOM+6, gray_precon[one[x][y]]);
                        }
                }
                //draw char twenty-two
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+22*xscale)-5, x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+22*xscale)-5, x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+22*xscale)-5, x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+22*xscale), x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+22*xscale), x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+22*xscale), x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                //draw char twenty-three
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+23*xscale)-5, x+BOTTOM+6, gray[j][two[x][y]]);
                                if(j==0)
```

```
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+23*xscale)-5, x+BOTTOM+6, gray_batt[two[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+23*xscale)-5, x+BOTTOM+6, gray_precon[two[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+LEFT+margin(xzero+23*xscale), x+BOTTOM+6, gray[j][three[x][y]]);
                                if(j==0)
                                        gdImageSetPixel(image_bat,
y+LEFT+margin(xzero+23*xscale), x+BOTTOM+6, gray_batt[three[x][y]]);
                                        gdImageSetPixel(image_precon,
y+LEFT+margin(xzero+23*xscale), x+BOTTOM+6, gray_precon[three[x][y]]);
                        }
                }
        }
        //draw char hour
        for (x=0; x<8; x++){
                for (y=0; y<21; y++){
                        gdImageSetPixel(image_mote[0], y+LEFT+360, x+BOTTOM+17,
gray[0][charhour[x][y]]);
                        gdImageSetPixel(image_mote[1], y+LEFT+360, x+BOTTOM+17,
gray[1][charhour[x][y]]);
                        gdImageSetPixel(image_bat, y+LEFT+360, x+BOTTOM+17,
gray_batt[charhour[x][y]]);
                        gdImageSetPixel(image_precon, y+LEFT+360, x+BOTTOM+17,
gray_precon[charhour[x][y]]);
                }
        }

        //draw inlet and outlet label
```

```
        gdImageLine(image_precon, LEFT+5, BOTTOM+20, LEFT+20,BOTTOM+20,
green_precon);
        gdImageLine(image_precon, LEFT+55, BOTTOM+20, LEFT+70,BOTTOM+20,
DarkOrange2_precon);
        //draw inlet char
        for (x=0; x<8; x++){
                for (y=0; y<19; y++){
gdImageSetPixel(image_precon, y+LEFT+25, x+BOTTOM+17, gray_precon[charinlet[x][y]]);
                }
        }
        //draw outlet char
        for (x=0; x<8; x++){
                for (y=0; y<27; y++){
gdImageSetPixel(image_precon, y+LEFT+75, x+BOTTOM+17, gray_precon[charoutlet[x][y]]);
                }
        }
        //draw char 0V
        for (j=0;j<num_Motes;j++){
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
gdImageSetPixel(image_mote[j], y+LEFT-16, x+BOTTOM-3, gray[j][zero[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
gdImageSetPixel(image_mote[j], y+LEFT-10, x+BOTTOM-3, gray[j][charV[x][y]]);
                        }
                }
                //draw char 1V
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
gdImageSetPixel(image_mote[j], y+LEFT-16, x+BOTTOM-yscale-3, gray[j][one[x][y]]);
                        }
                }
```

```
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
gdImageSetPixel(image_mote[j], y+LEFT-10, x+BOTTOM-yscale-3, gray[j][charV[x][y]]);
                        }
                }
                //draw char 2V
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
gdImageSetPixel(image_mote[j], y+LEFT-16, x+BOTTOM-2*yscale-3, gray[j][two[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
gdImageSetPixel(image_mote[j], y+LEFT-10, x+BOTTOM-2*yscale-3, gray[j][charV[x][y]]);
                        }
                }
                //draw char 3V
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
gdImageSetPixel(image_mote[j], y+LEFT-16, x+BOTTOM-3*yscale-3, gray[j][three[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
gdImageSetPixel(image_mote[j], y+LEFT-10, x+BOTTOM-3*yscale-3, gray[j][charV[x][y]]);
                        }
                }
        }
        //draw ylabel for prediction concentration
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
0*yscale_prec-3, gray_precon[zero[x][y]]);}
        }
        for (x=0; x<8; x++){
```

```
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
1*yscale_prec-3, gray_precon[one[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
2*yscale_prec-3, gray_precon[two[x][y]]);}
        }

        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
3*yscale_prec-3, gray_precon[three[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
4*yscale_prec-3, gray_precon[four[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
5*yscale_prec-3, gray_precon[five[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
6*yscale_prec-3, gray_precon[six[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
7*yscale_prec-3, gray_precon[seven[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
8*yscale_prec-3, gray_precon[eight[x][y]]);}
        }
        for (x=0; x<8; x++){
```

```
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
9*yscale_prec-3, gray_precon[nine[x][y]]);}
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-16, x+BOTTOM-
10*yscale_prec-3, gray_precon[one[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){gdImageSetPixel(image_precon, y+LEFT-10, x+BOTTOM-
10*yscale_prec-3, gray_precon[zero[x][y]]);}
        }
        for (x=0; x<15; x++){
                for (y=0; y<9; y++){gdImageSetPixel(image_precon, y+LEFT-25, x+BOTTOM-
6*yscale_prec-3, gray_precon[chargdL[x][y]]); }
        }
        //draw char 2V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-0*yscale_bat-3, gray_batt[two[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<7; y++){
gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-0*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 4V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-2*yscale_bat-3, gray_batt[four[x][y]]);
                }
        }
        for (x=0; x<8; x++){
```

```
                for (y=0; y<7; y++){
gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-2*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 6V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-4*yscale_bat-3, gray_batt[six[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<7; y++){
gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-4*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 8V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-6*yscale_bat-3, gray_batt[eight[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<7; y++){
gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-6*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 10V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-22, x+BOTTOM-8*yscale_bat-3, gray_batt[one[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
```

```
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-8*yscale_bat-3, gray_batt[zero[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<7; y++){
gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-8*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 12V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-22, x+BOTTOM-10*yscale_bat-3, gray_batt[one[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-10*yscale_bat-3, gray_batt[two[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<7; y++){gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-
10*yscale_bat-3, gray_batt[charV[x][y]]);
                }
        }
        //draw char 14V for battery
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-22, x+BOTTOM-12*yscale_bat-3, gray_batt[one[x][y]]);
                }
        }
        for (x=0; x<8; x++){
                for (y=0; y<5; y++){
gdImageSetPixel(image_bat, y+LEFT-16, x+BOTTOM-12*yscale_bat-3, gray_batt[four[x][y]]);
                }
```

```
	}
	for (x=0; x<8; x++){
		for (y=0; y<7; y++){gdImageSetPixel(image_bat, y+LEFT-10, x+BOTTOM-
12*yscale_bat-3, gray_batt[charV[x][y]]);}
	}
	for(j=0; j<num_Motes; j++){
		//draw line and char IR45
		xaxile = 10;
		yaxile = 10;
		gdImageLine(image_mote[j], EDGE+BORDER+xaxile,BOTTOM+yaxile+10,
EDGE+BORDER+xaxile+20,BOTTOM+yaxile+10, DarkViolet[j]);
		for (x=0; x<8; x++){
			for (y=0; y<7; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+21, x+BOTTOM+yaxile+7, gray[j][charI[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<7; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+26, x+BOTTOM+yaxile+7, gray[j][charR[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+34, x+BOTTOM+yaxile+7, gray[j][four[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+40, x+BOTTOM+yaxile+7, gray[j][five[x][y]]);
			}
```

```
		}
		//draw line and char ORA45
		xaxile = xaxile +60;
		yaxile = 10;
		gdImageLine(image_mote[j], EDGE+BORDER+xaxile,BOTTOM+yaxile+10,
EDGE+BORDER+xaxile+20,BOTTOM+yaxile+10, green[j]);

		for (x=0; x<8; x++){
			for (y=0; y<7; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+23, x+BOTTOM+yaxile+7, gray[j][charO[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<7; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+30, x+BOTTOM+yaxile+7, gray[j][charR[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<7; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+38, x+BOTTOM+yaxile+7, gray[j][charA[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
				gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+46, x+BOTTOM+yaxile+7, gray[j][four[x][y]]);
			}
		}
		for (x=0; x<8; x++){
			for (y=0; y<5; y++){
```

```
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+52, x+BOTTOM+yaxile+7, gray[j][five[x][y]]);
                        }
                }
                //draw line and char ORA180
                xaxile = xaxile + 70;
                yaxile = 10;
                gdImageLine(image_mote[j], EDGE+BORDER+xaxile,BOTTOM+yaxile+10,
EDGE+BORDER+xaxile+20,BOTTOM+yaxile+10, DarkOrange2[j]);

                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+23, x+BOTTOM+yaxile+7, gray[j][charO[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+30, x+BOTTOM+yaxile+7, gray[j][charR[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<7; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+38, x+BOTTOM+yaxile+7, gray[j][charA[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+46, x+BOTTOM+yaxile+7, gray[j][one[x][y]]);
                        }
                }
```

```
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+52, x+BOTTOM+yaxile+7, gray[j][eight[x][y]]);
                        }
                }
                for (x=0; x<8; x++){
                        for (y=0; y<5; y++){
                                gdImageSetPixel(image_mote[j],
y+EDGE+BORDER+xaxile+58, x+BOTTOM+yaxile+7, gray[j][zero[x][y]]);
                        }
                }
        }
 // Finally, write the image out to a file.
 //printf("Creating output file '%s'.\n", filename_mote);
 outfile_mote[0] = fopen(filename_mote_01, "wb");
 gdImagePng(image_mote[0], outfile_mote[0]);
 fclose(outfile_mote[0]);
 outfile_mote[1] = fopen(filename_mote_02, "wb");
 gdImagePng(image_mote[1], outfile_mote[1]);
 fclose(outfile_mote[1]);
 outfile_bat = fopen(filename_12V, "wb");
 gdImagePng(image_bat, outfile_bat);
 fclose(outfile_bat);
 outfile_precon = fopen(filename_PreCon, "wb");
 gdImagePng(image_precon, outfile_precon);
 fclose(outfile_precon);
 /**********
  * Notes about the output :
  *  1. "wb" here means "write binary"; the 'binary' part only
  *     applies to Windows but doesn't hurt on unix boxes.
  *  2. If this had be a true color image then probably a JPEG would
  *     be a better choice than a PNG; in that case you'd just say
  *       gdImageJpeg(image, outfile, quality);
```

```
   *     where 0<=quality<=100, or quality=-1 for the libjpeg default.
   *  3. To send the output image to standard output, you'd just say
   *       gdImagePng(image, stdout);
   *     in which case you'd run the program like this :
   *       $ ./GD_example > outputfile.png
   **/
}
int margin (int x){
        if (x>720){
                x=x-720;
        }
        return x;
}
int drawValue(gdImagePtr image, int xall, int xall_old, int xzero, int Value, int Value_old, int
color, int eflag){
        if ((xall>(720-xzero)) && (xall_old>(720-xzero))){
                gdImageLine(image, LEFT+xzero-(720-xall), BOTTOM-Value, LEFT+xzero-
(720-xall_old), BOTTOM-Value_old, color);
                eflag=1;
        }else if (xall>(720-xzero)){
                gdImageLine(image, LEFT+xzero-(720-xall), BOTTOM-Value,
LEFT+xzero+xall_old, BOTTOM-Value_old, color);
                eflag=1;
        }else{
                if(eflag==0){//this flag indicate when to stop drawing
                        gdImageLine(image, LEFT+xall+xzero, BOTTOM-Value,
LEFT+xall_old+xzero, BOTTOM-Value_old, color);
                }
        }
        return eflag;
}
int fringe(int val){
        if (val<0)
                val=0;
```

```
        if(val>120)//120 pixels represents 3 Volt
                val=120;
        return val;
}
```

Appendix D - Datalogger programs

File Name: Send_PB1_LittleKitten_12100_v09.CR2

```
'CR200 Series Datalogger
'date:
'program author:Wei Han
'Declare Public Variables
Public SI_in(60)
Public Response
Public f1_1
Public batt_volt
'Declare Constants
Const MAXVALUES = 60
Const TERMCHAR = 35
'Main Program
BeginProg
  Delay(180, sec)
  RealTime( SI_in , 0 )
  Print(-2,9600,SI_in(1),"/",SI_in(2),"/",SI_in(3))
  Print(-2,9600,"/", SI_in(4),"/", SI_in(5), "/")
  Print(-2,9600, SI_in(6),CHR$(13))
   SI_in(1)=0
   Scan (1,Sec)
          Battery (batt_volt)
    f1_1=0
            Print (2, 9600)
    SerialInput (SI_in(),MAXVALUES,TERMCHAR,$)
    If SI_in(1)<>0 Then
     If SI_in(1)=4000 Then 'time synchronization
       RealTime( SI_in , 0 )
       Print(-2,9600,SI_in(1),"/",SI_in(2),"/",SI_in(3))
       Print(-2,9600,"/", SI_in(4),"/", SI_in(5), "/")
       Print(-2,9600, SI_in(6),CHR$(13))
```

```
    SI_in(1)=0
  Else
    While f1_1=0
    Wend
    'voltage for gateway
    SI_in(3) = batt_volt
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(1), 10,fDest1,1,6,6,00000)
    Wend
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(11),10,fDest2,1,6,6,00000)
    Wend
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(21),10,fDest3,1,6,6,00000)
    Wend
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(31),10,fDest4,1,6,6,00000)
    Wend
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(41),10,fDest5,1,6,6,00000)
    Wend
    Response=-1
    While Response<>0
      SetValue(Response,SI_in(51),10,fDest6,1,6,6,00000)
    Wend
    Response=-1
    SI_in(2)=1
    While Response<>0
      SetValue(Response,SI_in(2),1,flag,1,6,6,00000)
```

```
        Wend
        SI_in(1)=0
      EndIf
    EndIf
        NextScan
EndProg
```

File Name: Relay_PB6_Riley_Cico_v07.CR2

```
'CR200 Series Datalogger
'date:
'program author:Wei Han
'Declare Public Variables
Public Res
Public fDest1(10),fDest2(10),fDest3(10),fDest4(10),fDest5(10),fDest6(10)
Public flag, f6_6
Public batt_volt
'Define Subroutines
        Sub GetData
                Res=-1
                If fDest1(1)=0
                  Delay(3,sec)
                EndIf
                While Res<>0
                  SetValue(Res,fDest1,10,Dest1,1,2,2,00000)
                Wend
                Res=-1
                If fDest2(1)=5120
                  Delay(3,sec)
                EndIf
                While Res<>0
                  SetValue(Res,fDest2,10,Dest2,1,2,2,00000)
                Wend
                Res=-1
                If fDest3(1)=5120
```

```
  Delay(3,sec)
EndIf
While Res<>0
  SetValue(Res,fDest3,10,Dest3,1,2,2,00000)
Wend
Res=-1
If fDest4(1)=5120
  Delay(3,sec)
EndIf
'voltage for repeater
fDest4(10)=batt_volt
While Res<>0
        SetValue(Res,fDest4,10,Dest4,1,2,2,00000)
Wend
Res=-1
If fDest5(1)=0
  Delay(3,sec)
EndIf
While Res<>0
        SetValue(Res,fDest5,10,Dest5,1,2,2,00000)
Wend
Res=-1
If fDest6(1)=5120
  Delay(3,sec)
EndIf
While Res<>0
        SetValue(Res,fDest6,10,Dest6,1,2,2,00000)
Wend
Res=-1
fDest1(2)=1
While Res<>0
        SetValue(Res,fDest1(2),1,flag,1,2,2,00000)
Wend
fDest1(1)=0
```

```
            fDest2(1)=5120
            fDest3(1)=5120
            fDest4(1)=5120
            fDest5(1)=0
            fDest6(1)=5120
        EndSub
'Main Program
BeginProg
flag=1
fDest1(1)=0
Scan (1,Sec)
        Battery (batt_volt)
         f6_6=0

         SetValue(Res,flag,1,f1_1,1,1,1,00000)
         Delay(3,sec)
         If fDest1(1)<>0 Then
                flag=0
               While f6_6=0
               Wend
                Call GetData
        EndIf
NextScan
EndProg
```

File Name: Receive_PB2_Central_Riley_v08.CR2

```
'CR200 Series Datalogger
'date:
'program author:Wei Han
'Declare Public Variables
Public batt_volt
Public Res
Public Dest1(10),Dest2(10),Dest3(10),Dest4(10),Dest5(10),Dest6(10)
Public flag
'Define Subroutines
```

```
Sub GetData
        Print(2,9600,"$")
        If Dest1(10)=5120
          Delay(3,sec)
        EndIf
        Print(2,9600,Dest1(1),Dest1(2),Dest1(3),Dest1(4),Dest1(5))
        Print(2,9600,Dest1(6),Dest1(7),Dest1(8),Dest1(9),Dest1(10))
        If Dest2(10)=5120
          Delay(3,sec)
        EndIf
        Print(2,9600,Dest2(1),Dest2(2),Dest2(3),Dest2(4),Dest2(5))
        Print(2,9600,Dest2(6),Dest2(7),Dest2(8),Dest2(9),Dest2(10))
        If Dest3(10)=5120
          Delay(3,sec)
        EndIf
        Print(2,9600,Dest3(1),Dest3(2),Dest3(3),Dest3(4),Dest3(5))
        Print(2,9600,Dest3(6),Dest3(7),Dest3(8),Dest3(9),Dest3(10))
        If Dest4(10)=5120
          Delay(3,sec)
        EndIf
        Print(2,9600,Dest4(1),Dest4(2),Dest4(3),Dest4(4),Dest4(5))
        Print(2,9600,Dest4(6),Dest4(7),Dest4(8),Dest4(9),Dest4(10))
        If  Dest5(10)=5120
          Delay(3,sec)
        EndIf
        Print(2,9600,Dest5(1),Dest5(2),Dest5(3),Dest5(4),Dest5(5))
        Print(2,9600,Dest5(6),Dest5(7),Dest5(8),Dest5(9),Dest5(10))
        If Dest6(10)=0
          Delay(3,sec)
        EndIf
        'voltage for center
           Dest6(9) = batt_volt
        Print(2,9600,Dest6(1),Dest6(2),Dest6(3),Dest6(4),Dest6(5))
        Print(2,9600,Dest6(6),Dest6(7),Dest6(8),Dest6(9),Dest6(10))
```

```
        Print(2,9600,"#")
        Dest1(1)=0
        Dest1(7)=0
        Dest1(10)=5120
        Dest2(10)=5120
        Dest3(6)=0
        Dest3(10)=5120
        Dest4(10)=5120
        Dest5(5)=0
        Dest5(10)=5120
        Dest6(10)=0
        flag=1
EndSub
'Main Program
BeginProg
    flag=1
    Dest1(1)=0
    Scan (1,Sec)
        Battery (batt_volt)
        SetValue(Res,flag,1,f5_5,1,5,5,00000)
        Delay(3,sec)
        If Dest1(1)<>0 Then
            flag=0
            Call GetData
        EndIf
        SetValue(Res,flag,1,f6_6,1,6,6,00000)
        Delay(3,sec)
        If Dest1(1)<>0 Then
            flag=0
            Call GetData
        EndIf
    NextScan
EndProg
```

Appendix E - Time synchronization program

File Name: syncdate.c

```
/* This program is used to receive data from CR206 via RS-232 cable
then save it to a local file on the Stargate.
Stargate will then use this file to adjust its datetime to current.
Author: Wei Han
Date: Nov 5, 2008 2:43:00 PM
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#define TYPEDATA "./typedata" //save incoming data into this file
int cMonth(char[], int);
int main(){
        FILE *f1;
        char str[25];
        char *pch;
        char cdate[6][10];
        char ccmonth[4];
        int i,k;
        unsigned int j;
        int month;
        fd_set read_file_descr;
        struct timeval timeout;
        do{
                k=0;
                FD_ZERO(&read_file_descr);
                FD_SET(fileno(stdin),&read_file_descr);
                timeout.tv_sec = 120;//wait for 120 seconds
                timeout.tv_usec = 0;
                int error=select(1,&read_file_descr,0,0,&timeout);
```

```
//This program waits for 120 seconds and then contiune.
//The select function blocks the calling process until there is activity on any of the specified sets of file descriptors, or until the timeout period has expired.
if (error==0){
        printf("timeout\n");
        return 0;
}else if (error<0){
        printf("error\n");
        return 0;
}else{
        //open typedata file for writing
        if (!(f1 = fopen(TYPEDATA, "w+"))) {
                perror ("Couldn't open typedata file.\n");
                exit(1);
        }
        memset(str,'\0',25);
        scanf("%s",str);
        if (strlen(str)>=16){
                pch = strtok (str, "/");
                i=0;
                while(pch != NULL){
                        strcpy(cdate[i],pch);
                        i++;
                        pch = strtok (NULL,"/");
                }
        }
        //check for valid numbers
        for (i=0; i<6; i++){
                for (j=0; j<strlen(cdate[i]); j++){
                        if(!((cdate[i][j] >='0') && (cdate[i][j]<='9'))){
                                puts("Not a numeric value.\n");
                                k=1;
                                break;
                        }
```

```
                        }
                        if (k)
                                break;
                }
                if(!k){
                        month = strtol(cdate[1],NULL,10);
                        if (month<1 || month>12)
                                return 0;
                        if(!cMonth(ccmonth, month)){
                                fprintf(f1,"date --set \"%s %s %s:%s:%s %s\"",ccmonth,
cdate[2], cdate[3], cdate[4],cdate[5],cdate[0]);  //writes
                        }
                }
                fclose(f1);
        }
    } while (k);
    return 0;
}
int cMonth(char cm[], int n){
    switch(n){
        case 1:
            strcpy(cm, "Jan");
            break;
        case 2:
            strcpy(cm, "Feb");
            break;
        case 3:
            strcpy(cm, "Mar");
            break;
        case 4:
            strcpy(cm, "Apr");
            break;
        case 5:
            strcpy(cm, "May");
```

```
                break;
        case 6:
                strcpy(cm, "Jun");
                break;
        case 7:
                strcpy(cm, "Jul");
                break;
        case 8:
                strcpy(cm, "Aug");
                break;
        case 9:
                strcpy(cm, "Sep");
                break;
        case 10:
                strcpy(cm, "Oct");
                break;
        case 11:
                strcpy(cm, "Nov");
                break;
        case 12:
                strcpy(cm, "Dec");
                break;
        default:
                return 1;
    }
    return 0;
}
```

File Name: syndate.sh

```
#echo Welcome
/home/xbow/apps/syncdate/syncdate
chmod +rx typedata
source typedata
```

Appendix F - Java server program

File Name: KSUServer.java

```
/*This program use socket to listen port 2008 and store incoming data from RAVEN modem.
Wei Han
10:10:00 am Dec 8, 2008 */
import java.net.*;
import java.io.*;
public class KSUServer {
   public static void main(String[] args) throws IOException {
      ServerSocket serverSocket = null;
      boolean listening = true;
      try {
         serverSocket = new ServerSocket(2008);
      } catch (IOException e) {
         System.err.println("Could not listen on port: 2008.");
         System.exit(-1);
      }
          System.out.println("Waiting for a connection...");
         while (listening){
                  new KSUMultiServerThread(serverSocket.accept()).start();
          }
      serverSocket.close();
   }
}
```

File Name: KSUMultiServerThread.java

```
/*This program handles incoming data, converts them to engineering unit and stores them to a
mySQL database
          Wei Han
          Last updated: Nov 5, 2009
          Checked and Modified by Xu Wang       Jan 5, 2010 */
import java.net.*;
import java.io.*;
```

```
import java.sql.*;
import java.util.*;
import java.text.*;
public class KSUMultiServerThread extends Thread {
   private Socket socket = null;
   public KSUMultiServerThread(Socket socket) {
         super("KSUMultiServerThread");
         this.socket = socket;
         System.out.println("Accepted a connection from: "+ socket.getInetAddress());
   }
   public void run() {
         try {
            PrintWriter out = new PrintWriter(System.out, true);
            BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
                  final int Num_Packets = 3;
                  int i=0,k=0,j,count;
                  int init=0; //Modified by Xu Wang, for rain gauge
                  //Modified by Xu Wang, for rain gauge
                  int[] pt = new int[256]; // pt for  temporary precipitation to save the last
precipitation of different motes
                  int ptsn=0; // pt's serial number, pt[ptsn], fetch from MoteID
                  boolean r=false; //rain gauge
                  // Modified by Xu Wang, for CRC check, add PakBusAdd and SeqNum for the
2nd and 3rd data packets
                  String pakbusadd="";
                  String seqnum="";
                  boolean[] v = new boolean[Num_Packets];
                  int[] intdata = new int[60];
                  int[] crcRemote = new int[Num_Packets];
                  String[] results = new String[Num_Packets];
                  String[] crcCheck = new String[Num_Packets];
                  String[] strCheck = new String[Num_Packets];
                  String strSQL;
```

```
String inputLine, token;
Calendar c = Calendar.getInstance();
int year;
final double d=0.04; //Distance between two LEDs
final int sf=279; //Mote sampling frequency 279/s for one channel, 140/s for two
channels
NumberFormat formatsensorbatt = new DecimalFormat("0.00");
NumberFormat formatsensordata = new DecimalFormat("0.0000");
NumberFormat formatradiobatt = new DecimalFormat("00.00");
NumberFormat formatvelocity = new DecimalFormat("000.00");
CRC16 my_crc = new CRC16();
results[0]=results[1]=results[2]="";
crcCheck[0]=crcCheck[1]=crcCheck[2]="";
crcRemote[0]=crcRemote[1]=crcRemote[2]=0;
v[0]=v[1]=v[2]=false;
//Modified by Xu Wang, for rain gauge, initial pt[] array
for (ptsn=0; ptsn<256; ptsn++){
        pt[ptsn] = 0;
}
while ((inputLine = in.readLine()) != null) {
        StringTokenizer st = new StringTokenizer(inputLine);
        Connection conn = null;
search:
        while (st.hasMoreTokens()) {
          token=st.nextToken();
          //Check the beginning
          if (token.equals("$")){
                results[k]="";
                crcCheck[0]="";
                crcRemote[0]=0;
                i=0;
          }
          if (i==3 || i==40 ||i==59){
                 intdata[i-1]  = (int)(Float.valueOf(token).floatValue()*100.0);
```

```
                    if (intdata[i-1]>5000)
                        intdata[i-1] = 5000;
                }
                //Validate data and convert String to integer
                if (i > 0 && i<61){
                    if (!(i==3 || i==40 ||i==59)){
                        try {
                            intdata[i-1] = Integer.parseInt(token);
                        }
                        catch (NumberFormatException nfe) {
                            // exit loop
                            out.println("Invalid data");
                            break search;
                        }
                    }
                        //prepare string for crc check
                        //Modified by Xu Wang
                        if (i==1) {
                            pakbusadd = token;
                            }
                        if (i==2) {
                            seqnum = token;
                            }
                        if (!(i==3 || i == 22 || i==40 || i==41 || i==59 ||
i==60)){//do not check gateway, repeater and central battery as well as crc data
                            crcCheck[k] = crcCheck[k] + token;
                        }
                        switch (i) {
                            case 3:  //gateway battery
                            case 21: //ch13
                            case 40: //repeater battery
                            case 59: //central battery
                                //Reserved field, do nothing
                                break;
```

```
                                        case 4:
                                           if (intdata[i-1]<0 || intdata[i-1]>366){

        results[0]=results[1]=results[2]="";
                                    if (conn != null){
                                        try
                                        {
                                        conn.close ();
                                        }catch (Exception e) { // ignore close errors
                                        }
                                        }
                                                break search;
                                                }
                                                year = c.get(Calendar.YEAR); //get year
                                                JulianDate JD = new JulianDate(year,
intdata[i-1]);
                                                JD.convert();
                                                results[k] = results[k] + """+ JD.cDate +
", ";
                                                break;
                                        case 23:
                                        case 42:
                                           if (intdata[i-1]<0 || intdata[i-1]>366){

        results[0]=results[1]=results[2]="";
                                    if (conn != null){
                                        try
                                        {
                                        conn.close ();
                                        }catch (Exception e) { // ignore close errors
                                        }
                                        }
                                                break search;
                                                }
```

```
                                        year = c.get(Calendar.YEAR); //get year
                                        JulianDate JD2 = new JulianDate(year,
intdata[i-1]);
                                        JD2.convert();
                                        results[k] = results[k] + intdata[0]+", "+
intdata[1] + ", '" + JD2.cDate + ", ";
                                        break;
                                case 5:
                                case 24:
                                case 43:
                                  if (intdata[i-1]<0 || intdata[i-1]>86400){

        results[0]=results[1]=results[2]="";
                                if (conn != null){
                                        try
                                        {
                                        conn.close ();
                                        }catch (Exception e) { // ignore close errors
                                        }
                                        }
                                        break search;
                                        }
                                TimeString TS = new TimeString(intdata[i-1]);
                                        TS.convert();
                                        results[k] = results[k] + TS.ctime + "', ";
                                        break;
                                case 6://check group ID
                                case 25:
                                case 44:
                                        if (intdata[i-1]==125){
                                                v[k]=false;
                                        }else{
                                                v[k]=true;
                                        }
```

```
                                        results[k] = results[k] + intdata[i-1] + ", ";
                                                break;
                                        // Modified By Xu Wang
                                        case 7: // Get mote ID
                                        case 26:
                                        case 45:
                                                ptsn = intdata[i-1];
                                        results[k] = results[k] + intdata[i-1] + ", ";
                                                break;
                                        case 8: //Mote battery
                                        case 27:
                                        case 46:
                                                if (v[k]){
                                                        results[k] = results[k] +
formatsensordata.format(intdata[i-1]*2.5/4096.0) + ", ";
                                                }else{
                                                        results[k] = results[k] +
formatsensorbatt.format(intdata[i-1]/1000.0) + ", ";
                                                }
                                                break;
                                        //Modified By Xu Wang
                                        case 16: //Rain gauge
                                        case 35:
                                        case 54:
                                                if(v[k]){
                                                        results[k] = results[k] +
formatsensordata.format(intdata[i-1]*2.5/4096.0) + ", ";
                                                }else{
                                                        if (pt[ptsn] == 0){
                                                                if (!r){
                                                                        pt[ptsn] =
intdata[i-1];
                                                                        results[k] =
results[k] + formatsensordata.format(0) + ", ";
```

```
                                                    r = true;
                                                    }else {
                                                    pt[ptsn] =
intdata[i-1];
                                                    results[k] =
results[k] + formatsensordata.format(pt[ptsn] * 0.254) + ", ";
                                                    }
                                                }else if (pt[ptsn] == intdata[i-
1]){
                                                    results[k] = results[k] +
formatsensordata.format(0) + ", ";
                                                }else if (pt[ptsn] > intdata[i-1]){
                                                    results[k] = results[k] +
formatsensordata.format(0) + ", ";
                                                    pt[ptsn] = 0;
                                                }else {results[k] = results[k] +
formatsensordata.format((intdata[i-1] - pt[ptsn]) * 0.254) + ", ";
                                                   pt[ptsn]=intdata[i-1];
                                                }
                                            }
                                            break;
                                        //Modified By Xu Wang
                                        case 17:// Thermocouple
                                        case 36:
                                        case 55:
                                            if(v[k]){
                                                results[k] = results[k] +
formatsensordata.format(intdata[i-1]*2.5/4096.0) + ", ";
                                            }else{
                                                results[k] = results[k] +
formatradiobatt.format((intdata[i-1]*2.5/4096.0/0.2473-0.011)*25) + ", ";
                                            }
                                            break;
                                        case 18: //Velocity
```

```
                                        case 37:
                                        case 56:
                                                if(v[k]){
                                                        results[k] = results[k] +
formatsensordata.format(intdata[i-1]*2.5/4096.0) + ", ";
                                                }else{
                                                        if (intdata[i-1] > 1){
                                                                results[k] = results[k] +
formatvelocity.format(d*sf/(intdata[i-1]-1)) +", ";
                                                        }else{
                                                                results[k] = results[k] +
formatvelocity.format(0.0) +", ";
                                                        }
                                                }
                                                break;
                                        case 9:
                                        case 10:
                                        case 11:
                                        case 12:
                                        case 13:
                                        case 14:
                                        case 15:
                                        case 19:
                                        case 20:
                                        case 28:
                                        case 29:
                                        case 30:
                                        case 31:
                                        case 32:
                                        case 33:
                                        case 34:
                                        case 38:
                                        case 39:
                                        case 47:
```

```
                                        case 48:
                                        case 49:
                                        case 50:
                                        case 51:
                                        case 52:
                                        case 53:
                                        case 57:
                                        case 58:
                                                results[k] = results[k] +
formatsensordata.format(intdata[i-1]*2.5/4096.0) + ", ";
                                                break;
                                        case 22:
                                                crcRemote[k] = intdata[i-1];
                                                k=1; //indicate this is the end of the first
group of data.
                                                results[k]="";
                                                crcCheck[k]= pakbusadd + seqnum;
                                                crcRemote[k]=0;
                                                break;
                                        case 41:
                                                crcRemote[k] = intdata[i-1];
                                                k=2; //indicate the end of the second
group of data.
                                                results[k]="";
                                                crcCheck[k]= pakbusadd + seqnum;
                                                crcRemote[k]=0;
                                                break;
                                        case 60:
                                                crcRemote[k] = intdata[i-1];
                                                k=0; //indicate the third group of data.
                                                break;
                                        default:
                                                results[k] = results[k] + intdata[i-1] + ",
";
```

```
                                        break;
                                }
                        }
                        //Check the end
                        if (token.equals("#")){
                                for (j=0; j<Num_Packets; j++){
                                        out.println(results[j]);
                                        out.println(crcCheck[j]);
                                        out.println(crcRemote[j]);
                        if
(crcRemote[j]==my_crc.getValue(crcCheck[j],crcCheck[j].length())){
                                                strCheck[j]= "0";
                                }else{
                                                strCheck[j]= "1";
                                   }
                                }
                                //Insert into mySQL database
                                try
                               {
                                String userName = "root";
                                String password = "bae840-05";
                                String url = "jdbc:mysql://129.130.45.77:3308/estcp";
                                Class.forName ("com.mysql.jdbc.Driver").newInstance ();
                                conn = DriverManager.getConnection (url, userName,
password);
                                Statement s = conn.createStatement ();

                                for (j=0; j<Num_Packets; j++){

                                                        if (v[j]){
                                                                strSQL="INSERT
INTO tblvelocity_riley (PakBusAdd, SeqNum, DateTime, PacketID, MoteID,"
                                                                                + "
Ch0, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7, Ch8, Ch9,"
```

```
                                                                    + "
Ch10, Ch11, Ch12, CRC)"
                                                                    + "
VALUES(" + results[j] + strCheck[j] + ")";
                                                        }else{
                                                            strSQL="INSERT
INTO tblresults (PakBusAdd, SeqNum, DateTime, GroupID, MoteID,"
                                                                    + "
MoteBatt, Ch1, Ch2, Ch3, Ch4, Ch5, Ch6, Ch7, Ch8, Ch9,"
                                                                    + "
Ch10, Ch11, Ch12, Ch14, GatewayBatt, RepeaterBatt, CenterBatt)"
                                                                    + "
VALUES(';

                                    strSQL = strSQL + results[j]
                                                + strCheck[j] + ", "
                                                +
formatradiobatt.format(intdata[2]/100.0) + ", "
                                                +
formatradiobatt.format(intdata[39]/100.0) + ", "
                                                +
formatradiobatt.format(intdata[58]/100.0) + ")";
                                                    }
                                                     count = s.executeUpdate (strSQL);
                                            }
                                            s.close ();
                                        }
                                catch (Exception e)
                                {
                                   System.err.println(e);
                                   System.err.println ("Cannot connect to database server");
                                }
                                finally
                                {
```

```
                                        if (conn != null)
                                        {
                                           try
                                           {
                                              conn.close ();
                                           }
                                           catch (Exception e) { // ignore close errors
                                           }
                                        }
                                    }
                                     out.println("Data seq # " + intdata[1] + " received
successfully.\n");
                                     i=0;
                                    }
                                    i++;
                                  }
                    }
                    out.close();
                    in.close();
                    socket.close();
                } catch (IOException e) {
                        if (e instanceof SocketException){
                                System.out.println("Waiting for a connection...");
                        }else{
                        e.printStackTrace();
                        }
                }
        }
}
```