Sensor web language for dynamic sensor networks

by

Gunita Saini and Bradford G. Nickerson

TR09-196, 15 Jan, 2010

Faculty of Computer Science

University of New Brunswick

Fredericton, N.B. E3B 5A3

Canada

Phone: (506) 453-4566

Fax: (506) 453-3566

Email: fcs@unb.ca

www: http://www.cs.unb.ca

Contents

1 Introduction

Sensor Web Language (SWL) [1, 5] is an extensible, object oriented language supporting robust message passing among various components: sensor nodes, gateway nodes, communication computers, LINUX server and web browser. SWL was first developed by University of New Brunswick (UNB) for sensor web configuration and query processing in hierarchical sensor networks in 2004.

SWL supports hierarchical routing using the same message structure through the sensor web. An SWL user can send a request for an on-site reading from the browser. The request is then received by the base station which, in turn, sends the request to the gateway. The gateway relays the request to the appropriate sensor node. The sensor node gets the reading from the appropriate sensor and sends the reading to the gateway.

The gateway then relays the reading back to the base station which, in turn, sends the response to the browser. SWL [2] provided an initial user interface to send a request to, and get response from the field in near real time. An initial implementation of the SWL compiler made it easy to configure the sensor web. Sensor nodes have to route messages to the base station through one gateway node. Command line development environments and a visual editor for Java, JavaCC, MySQL and nesC (running on TinyOS) were used in the software development.

SWL is modified various times to make it dynamic, robust and reliable. For clarity, we have divided SWL in versions, where,

- Version 1.0 is SWL for mesh architecture,
- Version 1.5 is SWL for mesh architecture with Acknowlegdement (ACK), and
- Version 2.0 is SWL for mesh architecture with ACK and Announce capabilities.

1.1 Mesh SWL

Zhongwei Sun presented a 'Sensor Web Language (SWL) [6, 10] for mesh architecture in 2005. The mesh SWL software architecture builds on the previous hierarichal SWL architecture. She added a full mesh architecture in the application layer of SWL. SWL toolkit was developed using Eclipse for generating standard SWL code for sensor networks. The browser generated by SWL compiler automatically places the gateway and sensor nodes in their appropiate geographical location. A square button represented a

gateway and a triangular button represented the sensor nodes. Sensors were represented by small buttons around the gateway and the sensor nodes. A MySQL database was designed for the SWL server. For mesh architecture in SWL, 5 new message types and 13 objects were added in SWL. This provided more reliable sensor web communication. Test results showed that a sensor web language supporting mesh architecture provides a sound foundation for flexible and reliable sensor web communication.

1.2 Fuzzy SWL

Ke Deng [4, 7] presented a sensor network programming platform using fuzzy logic to improve the responsiveness of Wireless Sensor Networks (WSNs). A fuzzy controller model is used to dynamically control the rate of observation of environmental variables. Differing rates arise from changing environmental conditions. Inferencing using linguistic rules provides a compact, human friendly way to represent the knowledge base for controlling environmental sensor networks. This approach reduces gateway communication and thus reduces energy consumption. Each sensor node can have multiple sensors attached to it. network state changes are transmitted only among sensors residing on different sensor nodes. This also results in less energy consumption. The fuzzy SWL was build over the previous non fuzzy SWL. Hence the fuzzy SWL grammar extends on the previous SWL grammar. Conclusions made after implementing Fuzzy SWL were:

- The message reception rate of fuzzy SWL was approximately 95%,
- The power consumption (with -2.7 dBm transmission power) overhead of fuzzy SWL was approximately 15% when compared with Crossbow's benchmark without fuzzy SWL (using 0 dBm transmission power) ,
- The SWL fuzzy control system made monitoring more frequent during heavy rainfall, and conserved energy during light or no rainfall, and
- The RAM and ROM usage of fuzzy SWL increased linearly as the number of rules increased.

The experimental results confirmed that fuzzy SWL control system is feasible and effective for environmental monitoring sensor networks.

1.3 Reliable protocol for communication in WSN

John Paul Arp [8] presented a Disseminated ACKnowledgement (DACK) protcol for data collection in WSNs. The goal is to improve the reliability

of data collection in WSNs using dissemination to achieve end-to-end acknowledgement of data samples. Objectives of the thesis were to minimize the energy consumption of the wireless nodes while communicating and to minimize the data loss during communication thereby enhancing reliablity.

1.4 Dynamic Adaptation

In WSNs nodes appear and disappear, and WSN communication protocols (such as Zigbee, 6LowPan and DYMO) in tier 1 and 2 of the WSN (shown in Figure 1) are designed to adapt to this dynamic environment [9]. The goal of this report is to document the changes made in SWL toolkit for dynamically adapting a changing WSN to the the tiers 3 and 4 of the WSN (i.e the Base Station and the Web Browser).

To dynamically adapt a changing WSN to the base station (tier 3) of WSN, changes are made in SWLMote code, base station code, the SWLcompiler and the SWLdatabase. For dynamic adaptation on the web browser (tier 4) of WSN, without recompiling the whole code, we propose to integrate the OGC standard Sensor Observation Service (SOS) [3] as an interface for adding real time sensor updates into the web.

2 Existing single hop structure

The updated SWL message grammar can be seen in Table 2. A new Request type RequestAnnounce is added in the grammar to recognize the RequestAnnounce message sent from the base station to the sensor nodes. A new Response type SWLAnnounce is also added. NodeConfig consists of the NodeUUID, latitude, longitude, session ID, number of sensors attached to the node and the sensor UUIDs. The sessionID and UUIDs are further explained in the sections 6.2.1 and 6.2.2, respectively. The RequestAnnounce and ResponseAnnounce message are further explained in sections 8.1 and 8.2, respectively.

The +n sign in grammar rules 21, 49, 50, 51 from Table 2 indicate that precisely 'n' instances of the preceeding token appear.

Figure 2 shows the modified SWLMessage structure including two added fields for Announce. The two added fields are message_size and sessionid discussed in sections 3 and 8.1, respectively. AckSeqNo is the acknowlegement sequence number, an unsigned 1 byte int. Sender_id is the id of the node sending the message, an unsigned 2 byte int. SwlMsgType is the message type, an unsigned 1 byte int. All message types defined in SWL are shown

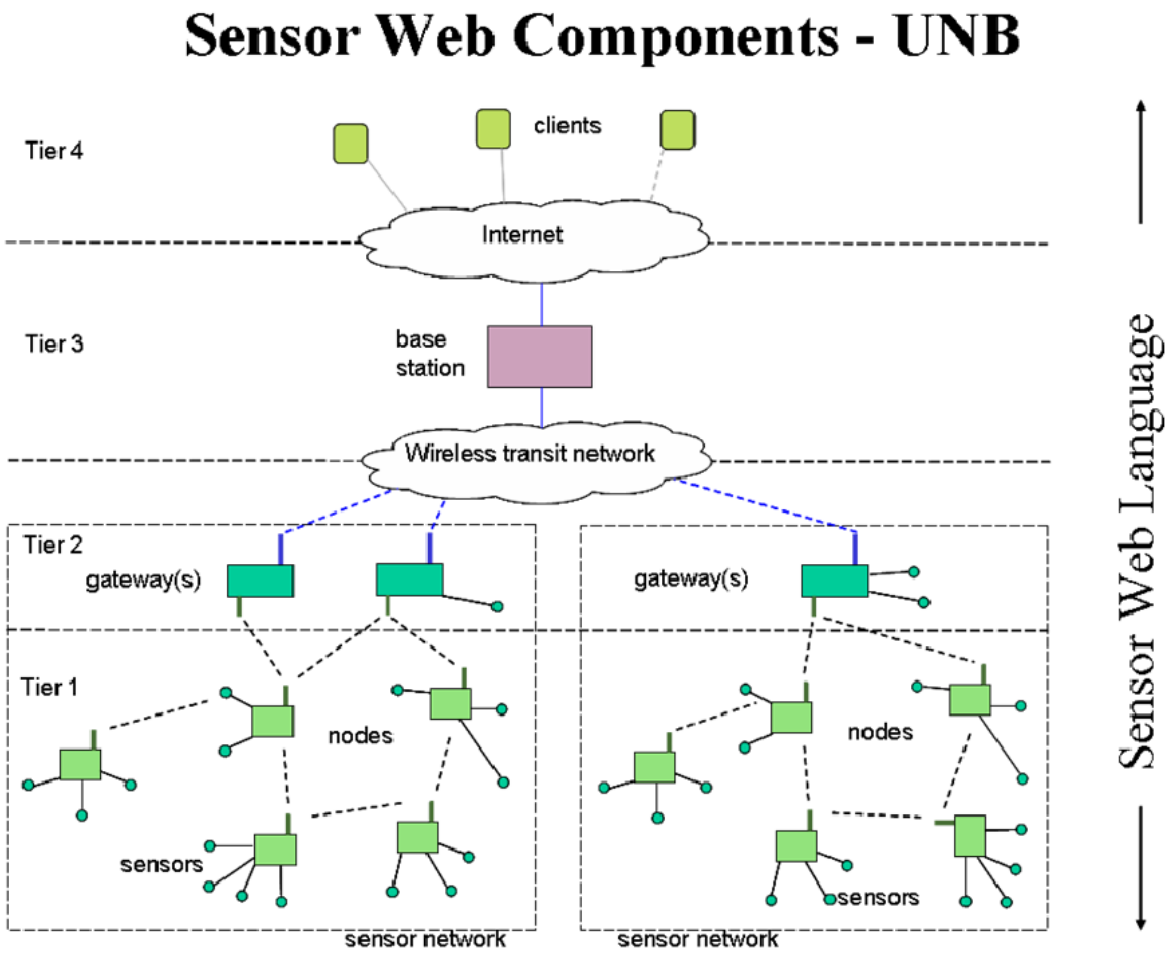

Figure 1: WSN architecture from ([9]).

```
typedef struct SWLMsg {
uint8_t ackSeqNo;
uint16_t senderID;
uint8_t swlMsgType;
uint8_t swlMsgID;
uint16_t swlSeqNo;
uint8_t message_size;
uint8_t sessionid;
uint8_t payload[22];
} SWLMsg;
```

Figure 2: Modified SWLMsg structure extended from [10].

1	*Goal*	::=	*MainClass* [{ *ClassDeclaration* }+ | { *Action* }+] EOF
2	*MainClass*	::=	sensornet *Identifier* { *MainDecl* }*
3	*MainDecl*	::=	main (*Arg*) { { *VarDecl* }* { *Statement* }* }
4	*ClassDeclaration*	::=	*ClassSimpleDecl* | *ClassExtendsDecl*
5	*ClassSimpleDecl*	::=	class *Identifier* { { *VarDecl* }* { *Constructor* }* { *MethodDecl* }* }
6	*ClassExtendsDecl*	::=	class *Identifier* extends *Identifier* { { *Statement* }* { *VarDecl* }* { *Constructor* }* { *MethodDecl* }* }
7	*Constructor*	::=	*Identifier* ({ *Arg* }*) { { *SuperCon* }* { *Statement* }* }
8	*SuperCon*	::=	super ({ *Param* }*) ;
9	*Action*	::=	request { *SensorNetId* { *Request* }* } | response { *SensorNetId* { *Response* }* } | report { *SensorNetId* { *Response* }* } | alert { *SensorNetId* { *Response* }* } | changeI { *SensorNetId* { *Response* }* } | switchG { *SensorNetId* { *Response* }* }
10	*SensorNetId*	::=	sensornet (*Identifier*)
11	*Request*	::=	*RequestBasic* | *RequestArray* | *RequestConfig* | *RequestAnnounce*
12	*RequestBasic*	::=	{ *Identifier.IdRest* ([*Param*])};
13	*RequestArray*	::=	{ *ArrayRef.IdRest* ([*Param*])};
14	*RequestConfig*	::=	{ Configuration;}
15	*RequestAnnounce*	::=	{*Identifier*.ga();}
16	*Response*	::=	*ResponseConstructor* | *ResponseArrayCon* | *ResponseArrayValue* | *ResponseConfig* | *ResponseAnnounce*
17	*ReponseConstructor*	::=	{*Identifier.IdRest* ([*Param*]);}
18	*ResponseArrayCon*	::=	{*ArrayRef.IdRest* ([*Param*]);}
19	*ResponseValue*	::=	{ *Identifier.IdRest* = *Exp* ;}
20	*ResponseConfig*	::=	{Configuration{{*varDecl*}*{*Exp*}*}}
21	*ResponseAnnounce*	::=	{nid={*Hexdigit*}+16;lt={ N | S}*deg,min,sec*;s={*IntegerType*};ln={ E | W }*deg,min,sec*;n={*IntegerType*}(;SID={*Hexdigit*+})} | { *NodeConfig* }
22	*NodeConfig*	::=	{Configuration{{*varDecl*}*{*Exp*}*}}
		:	
		:	
49	*deg*	::=	{*Digit*}+2
50	*min*	::=	{*Digit*}+2
51	*sec*	::=	{*Digit*}+5
52	*IntegerType*	::=	int
		:	
		:	
98	*Digit*	::=	{0 | ... | 9}
97	*HexDigit*	::=	{0 | ... | 9 | a | ... | f}

Table 2: A portion of Adaptive SWL Grammar extended from [4, 10].

in Table 3. Every SWL message is split into packets which fit in the payload within a TinyOS packet of size 22 bytes.

Table 3: SWL message types.

Message	SWL Msg. Type
ALIVE	0
REQUEST	1
RESPONSE	2
REPORT	3
SWITCHG	4
ALERT	5
CHANGEI	8
CHECKSTAT	9
RXREADY	10
TXDONE	11
SLEEP	12
WAIT	13
RESET	14
ANNOUNCE	15

3 The sensor node program structure

There are 2 sensor node programs - SWLMote.nc and SWLMoteM.nc and 2 header files - SWLMsg.h and config.h. SWLMote.nc is a top-level configuration file and the source file which the nesC compiler uses to generate an executable file. SWLMoteM.nc provides the implementation of the SWLMote application. SWLMsg.h defines the structure of SWLMsg and the AckMsg. Figure 3 shows a diagrammatic view of the sensor node program compilation process.

As seen in the figure the mote program is installed and compiled on the motes using the Makefile. An example makefile is shown in Figure 4.

In figure 3 the config.h file is generated by the compiler. The Config.h file is a unique file for each sensor node. It has the nodeUUID, latitude, longitude, number of sensors attached to the node and description of the sensors attached. The description contains the sensorUUIDs, sensor type, sensor channel, sensor bit and sensor parameter. An example config.h file is shown in Figure 5 for a sensor node with three sensors (temperature, waterfall and battery) attached.

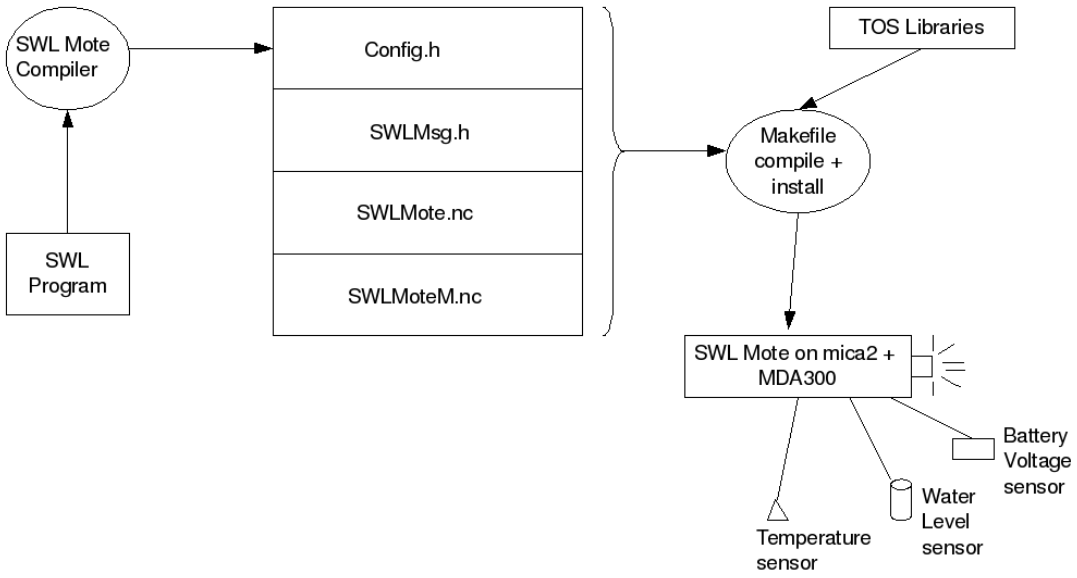

Figure 3: The sensor node program compilation process.

```
COMPONENT=SWLMote
SENSORBOARD=mda300
XBOWROOT={TOSROOT}/contrib/xbow/tos
XBOWBETAROOT={TOSROOT}/contrib/xbow/beta/tos
PFLAGS = -I$(XBOWBETAROOT)/platform/mica2
PFLAGS += -I$(XBOWBETAROOT)/CC1000RadioPulse
PFLAGS += -I$(XBOWBETAROOT)/lib/ReliableRoute_Low_Power
$(SENSORBOARD)
PFLAGS += -I$
/opt/tinyos-1.x/contrib/xbow/beta/tos/sensorboards/mda300
PFLAGS += -I$(XBOWBETAROOT)/interfaces
PFLAGS += -I$(XBOWBETAROOT)/system
PFLAGS += -I$(XBOWBETAROOT)/lib
PFLAGS += -I$(TOSROOT)/tos/lib/Queue
PFLAGS += -I$(TOSROOT)/tos/lib/Broadcast
PFLAGS += -I$ /opt/tinyos-1.x/contrib/ucb/tos/lib/RandomMLCG
PROGRAMMER_EXTRA_FLAGS = -v=2
include $TOSROOT/contrib/xbow/apps/MakeXbowlocal
include $TOSROOT/apps/Makerules
```

Figure 4: Makefile for compiling and installing a sensor node program on the wireless nodes.

```
#define nodeUUID "46f52dc76c6042319dde600f48d39d53"
#define LAT_DEG "N45"
#define LAT_MIN "95"
#define LAT_SEC "599999"
#define LONG_DEG "W66"
#define LONG_MIN "63"
#define LONG_SEC "339128"
#define SENSOR1_UUID "c6cf046dc1fa4e6381b2f8375ec863db"
#define SENSOR2_UUID "cfdf449efd554fe0be2bb6b29ec46778"
#define SENSOR3_UUID "391e816927484bcbb39a548ccc88bc6c"
#define NUM_SENSORS 3
enum {
REPORT_INT = 15, // Report Interval = 15 seconds
SENSOR1_TYPE = BATTERY,
SENSOR1_CHANNEL = 0,
SENSOR1_BIT = 0x0001, // A unique bit for each sensor
SENSOR1_INT = 20, // Sample Interval = 20 seconds
SENSOR1_PARAM = SAMPLER_DEFAULT,
SENSOR2_TYPE = ANALOG,
SENSOR2_CHANNEL = 0,
SENSOR2_BIT = 0x0002,
SENSOR2_INT = 900,
SENSOR2_PARAM = EXCITATION_25,
SENSOR3_TYPE = TEMPERATURE,
SENSOR3_CHANNEL = 0,
SENSOR3_BIT = 0x0004,
SENSOR3_INT = 900,
SENSOR3_PARAM = SAMPLER_DEFAULT,
};
```

Figure 5: Sample Config.h file.

An example SWLMsg.h file can be seen in Appendix A. The announce message type is defined as 15. We have added 2 more parameters: sessionid and message_size in the SWL message. Both are integer type. If a message is sent in 6 parts then the message_size will be 6. We have also defined a struct SessionParam to keep track of the sessionid.

The announce message consists of the nodeUUID, latitude, longitude, sessionID, number of sensors attached to the node and the sensor's UUID. The SWLMoteM program takes the NodeUUID, latitude, longitude, number of sensors and sensorUUID from the config.h file. When SWLMoteM recieves a RequestAnnounce message for a node, the node responds back with a SWLAnnounce message for that node. The Announce message is divided into a header consisting of three packets plus one additional packet for each sensor attached to the sensor node.

Each part of the SWLAnnounce message is written into the SWLMsg payload and then sent to the gateway, which further relays the message to the base station. Each packet of the SWLAnnounce message is sent only if an acknowledgement of the previous message is recieved by the SWLmoteM program. An example SWLMsg is shown in Table 3 containing a 6-packet SWLAnnounce message. In the payload column the values inside the square brackets is the length of that field in bytes.

Table 4: Example SWLMsg containing the SWLAnnounce message split into six packets. Each packet fits in the payload within a TinyOS packet. nid=nodeUUID, lt=latitude, s=sessionID, ln=longitude, n=number of sensors attached to the node, SID=sensorUUID, 15 is SWLAnnounce message type.

Sender ID	SWL Msg. Type	Msg. ID	SWL Seq No.	Message size	Session ID	Payload
1	15	1	0	6	66	nid=[16];
1	15	1	1	6	66	lt=[3],[2],[5];s=[3];
1	15	1	2	6	66	ln=[4],[2],[5];n=[2];
1	15	1	3	6	66	SID=[16];
1	15	1	4	6	66	SID=[16];
1	15	1	5	6	66	SID=[16];

4 Gateway mote

The gateway mote program is built on the previous Mesh SWL gateway mote program. The gateway mote code was modified by John Paul for his

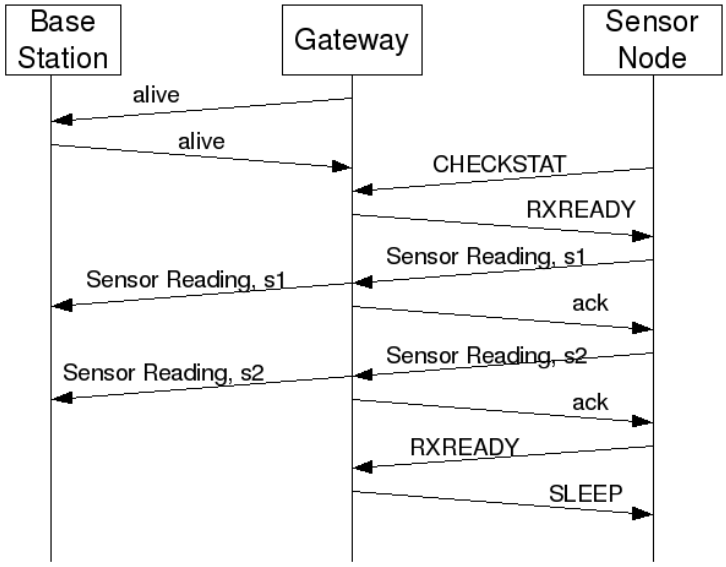

Figure 6: The gateway, sensor node and base station communication when a connection with the base station is generated.

DACK protocol [8]. In the current gateway mote code, before starting communication the gateway sends an alive message to the base station to check if it is alive. If the gateway recieves the alive message back, the gateway sends RXREADY message as a response to the sensor node's CHECKSTAT message. Then the sensor nodes start communication by sending their sensor readings to the gateway. The gateway passes these sensor readings to the base station and sends an acknowledgement to the sensor nodes. The CHECKSTAT communication mechanism is shown in the Figures 6 and 7.

Figure 8 shows a diagrammatic view of the gateway mote program structure. A makefile uses the nesc gateway programs and TOS libraries to install an compile the SWL gateway mote program.

For the SWLAnnounce message, we used the single hop gateway mote program used by Zhongwei Sun [10] and written by John Paul Arp. The only addition is that the gateway forwards any SWLAnnounce message it receives from the sensor nodes to the base station.

5 Base station

The base station consists of programs - SWLAnnounce.java, Moteindex.java, SWLResponseCollector.java, SWLAnnResponseCollector.java, and AnnResponseMsg.java.

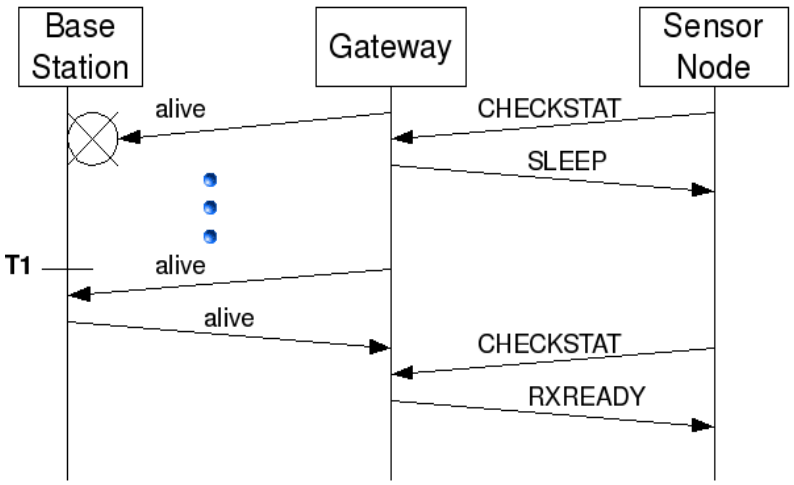

Figure 7: The gateway, sensor node and base station communication when the base station is not operational and then becomes operational at time T1.

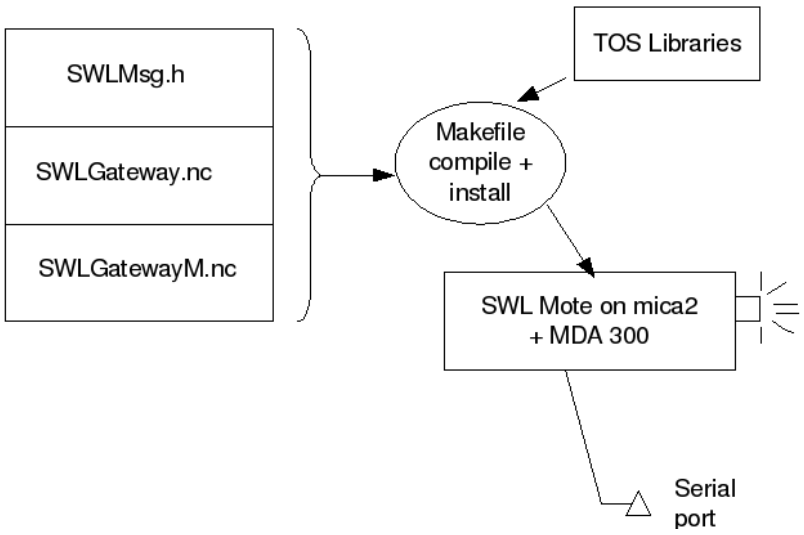

Figure 8: The gateway mote program structure.

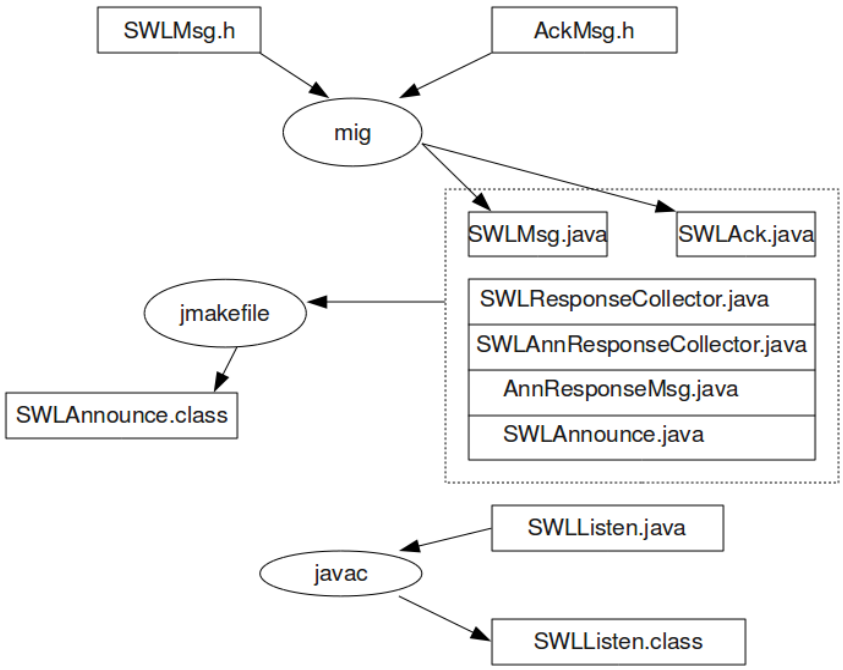

Figure 9: The Base station program structure.

SWLAnnounce.java is the central program for the Announce operation and talking to the SWL database. SWLAnnounce.java is discussed in length in the next section. SWLResponseCollector.java, SWLAnnResponseCollector.java and AnnResponseMsg.java are object classes used by SWLAnnounce.java. SWLResponseCollector.java and SWLAnnResponseCollector.java collect the whole response in an array while messages are received one by one by SWLAnnounce.java. SWLResponseCollector.java collects RESPONSE messages and SWLAnnResponseCollector.java collects ANNOUNCE messages. The length of the ANNOUNCE message is always three + number of sensors attached to a sensor node. Moteindex.java is also an object class called by the SWLAnnounce.java. Moteindex.java is also discussed in the next section. The base station program structure is shown in Figure 9. mig is a message interface generator for nesc. mig is a tool to generate code that processes tinyOS messages. Here mig generates SWLMsg.java and AckMsg.java (for base station use) from SWLMsg.h and AckMsg.h (TinyOS header files) respectively.

5.1 SWLAnnounce

SWLAnnounce.java receives and sends SWLMessages from and to the gateway. The gateway further forwards messages from and to the sensor nodes.

If messages received by SWLAnnounce are acknowledge (ACK) messages, then SWLAnnounce prints the message on standard output.

When communication is setup between the sensor node, the gateway and the base station and SWLAnnounce start receiving messages, SWLAnnounce saves the session(s) of that node with a Sender ID in a hashtable. If a new node enters the network, it has a new Sender ID, not present in the hashtable. SWLAnnounce now knows that a new node has entered the network and, needs to announce itself. SWLAnnounce sends a RequestAnnounce packet to the gateway. The gateway then forwards the RequestAnnounce to the sensor node. If a node is restarted, the Sender ID node will have a new session ID not present in the hash table. With this SWLAnnounce comes to know that a node is restarted and needs to announce itself. Moteindex.java is an empty class having only two member variables, id and sessionid. Moteindex.java is called by SWLAnnounce to save the latest session ID of each senderID into Moteindex.java's member variables. Moteindex.java is attached in Appendix B.3. For clearer understanding, pseudocode for the node announce process is shown in Algorithm 1. The complete code is in the messageReceived() function of program SWLAnnounce.java attached in Appendix B.1.

SWLAnnResponseCollector.java is shown in Appendix B.2. SWLAnnResponseCollector is an object class used by SWLAnnounce to save the Announce messages in an array.

SWLAnnounce also connects to the database. After recieving the announce message from the nodes, SWLAnnounce interacts with the database answerind queries and storing data. The database operations done by SWLAnnounce using MySQL are discussed further in Section 7. SWLAnnounce.java can be seen in Appendix B.1. Figure 10 shows the SWLAnnounce program communication with the gateway, the sensor nodes and MySQL database.

5.1.1 RequestAnnounce message

An example Request Announce message sent from the base station to a node via the gateway is shown in Figure 11. The message is sent by the base station to sensor node 3 to announce itself via the gateway.

5.1.2 SWLAnnounce message

An example Announce message sent as response to the RequestAnnounce message is shown in Figure 12.
In Figure 12, nid is the node UUID, lt is Latitude of the node, s is the sessionID of that node, ln is the longitude, n is the number of sensors attached to the node and SID is the Sensor UUID of a sensor attached to the node. The

```
input : int id // Sender ID
int sess_id // session ID
Hashtable Sessiontable
Moteindex mi          /* mi is object of class Moteindex */


Hashtable<Integer, Moteindex> Sessiontable = new
Hashtable<Integer, Moteindex>()
                                  /* defined the hashtable */
if Sessiontable.containsKey(id) then          /* If Sessiontable
contains the current Sender ID */

    if mi.sessionid ≠ sess_id then       /* compare the current
    session ID with the moteindex's session ID  */

        mi.id=id
        mi.sessionid=sess_id
            /* The values ID and session ID are updated in
        the Moteindex class */
        SWLAnnounce announce = new SWLAnnounce()
        announce.sendMsg()
         /* sendMsg is a function to send Announce message
        to the node with SenderID=id */
    end
else
    mi.id=id
    mi.sessionid=sess_id
        /* Save new Sender ID and session ID in Moteindex
    class */
    SWLAnnounce announce = new SWLAnnounce()
    announce.sendMsg()  /* Sends Announce message to a new
    node with SenderID=id */
end
```

Algorithm 1: Pseudo code to describe announcing mechanism in the messageRecieved() function of program SWLAnnounce.java.

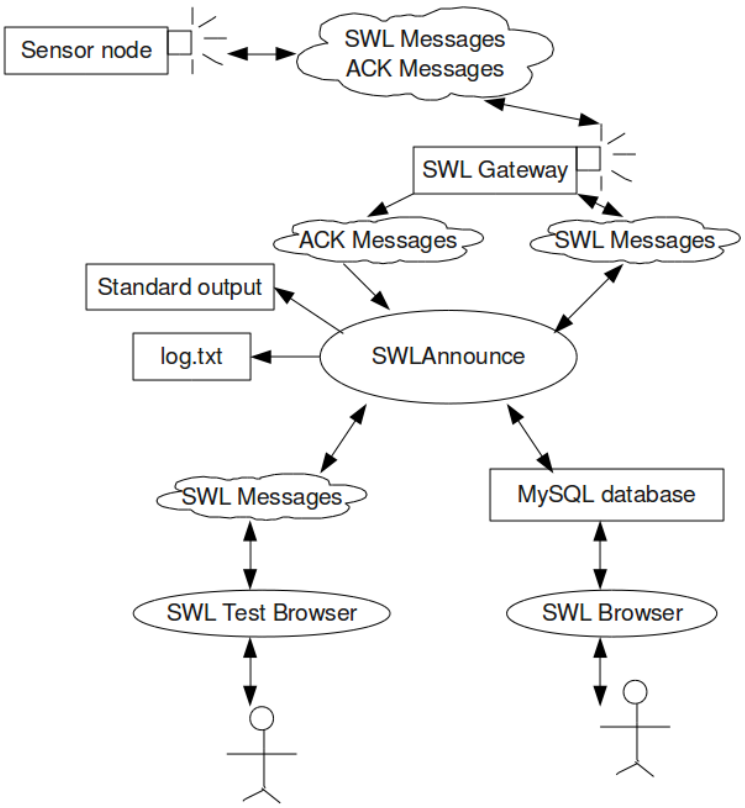

Figure 10: SWLAnnounce program interface with the sensor node, gateway and the SWL test browser.

```
request{
s3.ga();
}
```

Figure 11: Example SWL Request Announce message.

```
response{
nid=46f52dc76c6042319;
lt=N45,94,599999;s=97;
ln=W66,63,339128;n=3;
SID=c6cf046dc1fa4e638;
SID=cfdf449efd554fe0b;
SID=391e816927484bcbb;
}
```

Figure 12: Example SWL response to a request announce message.

latitude and longitude of a node is written as lt=degrees, minutes, seconds. Seconds are recorded using units of 1/10000 arc seconds.

5.2 Comparison of SWLAckListner (Version 1.5) program to SWLListner (Version 1.0) base station program

The SWLAckListner (Version 1.5) program is built on the SWLListner (Version 1.0) base station program. The modifications are as follows:

1. Messages sent to the base station by the sensor nodes are send via the gateway. The Gateway sends an acknowledgement for these messages to the sensor node [8].

2. All the messages from the base station to the nodes are first buffered in the gateway. The gateway forwards node messages when a sensor node wakes up for communication.

3. The format of the messages are changed as shown in sections 5.1.1 and 5.1.2.

6 Browser Applet communication

There are two browsers called SWL Browser and SWL Test Browser. Figure 13 illustrates the SWL Browser and SWL Test Browser communication with other SWL modules. The SWL Browser communicates with the servlet which in turn communicates with the SWL database. SWL Test Browser directly communicates with the SWLAnnounce program in the base station. The SensorNetworkBridge (See Appendix C.1) is a java program enabling serial

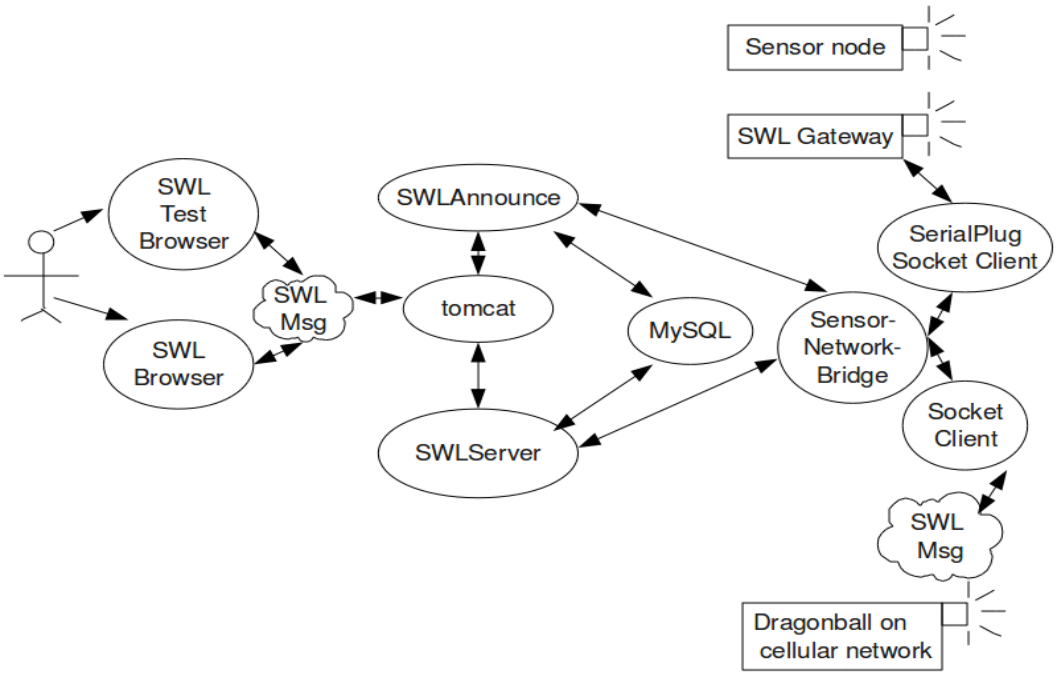

Figure 13: SWL Browser and SWL Test Browser communication with the SWLServer and MySQL modules.

communication on a com port when the gateway is directly connected to the base station. SerialPlug (See Appendix C.2) is a java program connecting the SensorNetworkBridge with the gateway. Socket Client another java program connecting the SensorNetworkBridge with the SWL Gateway or a cellular network.

7 Database population

A MySQL database was designed for recording sensor readings on the linux based base station. The intial relational schema designed for the SWL is defined in [10]. Initially there were 11 tables in the Sensor Web Language database. Now one more table 'SESSION' was added and four tables, 'SENSOR', 'SENSORNODE', 'GATEWAY', and 'POSIT' tables were modified. The updated relational database schema is shown in Figure 14. UUID of the sensor node and the sensor are unique for each sensor node and sensor respectively thoughout the database. Every component (eg. sensor, sensornode and gateway) has a posit_id, that is the position in latitude and longitude for each component is defined.

1. **SESSION**

id	SenderID	SessionID	datetime

2. **NETWORK**

id	GCD_id	name	description

3. **SENSOR**

id	manufacturer	description	model	name	unit	Attached_Node	posit_id	UUID

4. **SENSORNODE**

id	manufacturer	model	name	type	attached_GCD_id	sensor_network_id	posit_id	UUID

5. **GATEWAY**

id	sensor_network_id	sensornode_id	GCD_id	GCC_id	posit_id

6. **GATEWAYCD**

id	manufacturer	model	name	sensor_network_id	gateway_id

7. **GCC**

id	manufacturer	model	name	sensor_network_id	gateway_id

8. **CALIBRATION**

id	start_date	end_date	derivation

9. **COMP**

id	comp_id	observation_id

10. **COMP_OBSERVATION**

id	comp_value	timestamp	calibration_id

11. **SENSOR**

id	sensor_id	raw_value	derived_value	timestamp	caliberation_id	no_of_readings

12. **POSIT**

id	component_id	latitude	longitude	height_above_sea	height_above_ground	start_date	end_date

Figure 14: Updated relational database schema from [10].

All SWL queries are run using MySQL embedded in the java program (SWLAnnounce). SWLAnnounce queries and updates four database tables, as follows:

1. GATEWAY table
   Contains an additional field posit_id.

2. SESSION table
   The nodeID (SenderID), node's sessionID and datetime are saved in this table. SessionID of each node with time stamps is stored. If a node enters the WSN or the sessionID of a node changes, then a new entry is added to the SESSION table.

3. SENSORNODE table
   Two fields posit_id and UUID were added in SENSORNODE table. SWLAnnounce checks the node UUID, obtained from the SWLAnnounce message, with that in the SENSORNODE table. If present, SWLAnnounce retrieves the record from the database and passes the node's attributes to the browser. If the UUID is not present then SWLAnnounce inserts the node UUID in the SENSORNODE table.

4. SENSOR table
   IN SENSOR table also two fields, posit_id and UUID were added. The SWLAnnounce program queries the SENSOR table in the same way SWLAnnounce queried the SENSORNODE table. SWLAnnounce checks the sensor UUID, obtained from the SWLAnnounce message, with that in the SENSOR table. If present, SWLAnnounce retrieves the record from the database and passes the node's attributes to the browser. If the UUID is not present then SWLAnnounce inserts the sensor UUID in the SENSOR table.

5. POSIT table
   POSIT table contains the position of each component in the WSN. A new field component_id was added. Component_id is the UUID of a sensor node or a sensor or a SWL gateway node. SWLAnnounce checks the sensor node and sensor's UUID, obtained from the SWLAnnounce message, with that in the POSIT table. If present the SWLAnnounce compares the latitude and the longitude, obtained from the SWLAnnounce message, of that UUID with the ones in the POSIT table. If not almost equal ($\pm\epsilon$), the SWLAnnounce ends the record by setting the end date as the current date and inserts a new record of component's new latitude and longitude in the POSIT table. If the component

UUID is not present in the POSIT table then SWLAnnounce inserts the UUID with component's latitude and longitude into the table.

8 Unique Identifiers

A new message named AnnounceRequest is introduced. This message will be send by a sensor node to the base station when it joins the WSN. To achieve this two new unique identifiers are added in the current SWL message packet. They are session id and UUID.

8.1 Session ID

Session ID is a 1 byte unique integer defined for each node in the network. While sensor nodes communicate with the base station, nodes send their session ID and sender ID to the base station. If the base station encounters a new sender ID or for that sender ID a new session ID, base station will assume a new node has entered the network. Then the base station will send a Request Announce message to the that particular sensor node asking that node to announce itself. The session id is stored in node's flash memory.

8.2 UUID

UUID [11] is a 16byte universally unique identifier used for uniquely identifying each sensor node type and sensors attached to it throughout the network. The node's uuid and the sensor's uuid are also saved in the node's flash memory. Whenever the node aanounces itself to the base station in responce to the RequestAnoounce message from the base station the node sends a ResponseAnnounce message containing node's UUID and the UUIDs of all the sensors attached to the node.

The sensornode and sensor tables in the database are modified with one more column, UUID. The UUID will uniquely identify each node type and each sensor type. When ResponseAnnounce message is send by the nodes to the base station the UUID's of the nodes and sensors will be compared to get more information of that particular node and sensor from the database tables sensor and sensornode tables.

Through this the base station will have all the information of the new node and sensors attached to the node entering the WSN.

Figure 15 displays the communication between the base station, gateway and the nodes when WSN changes:

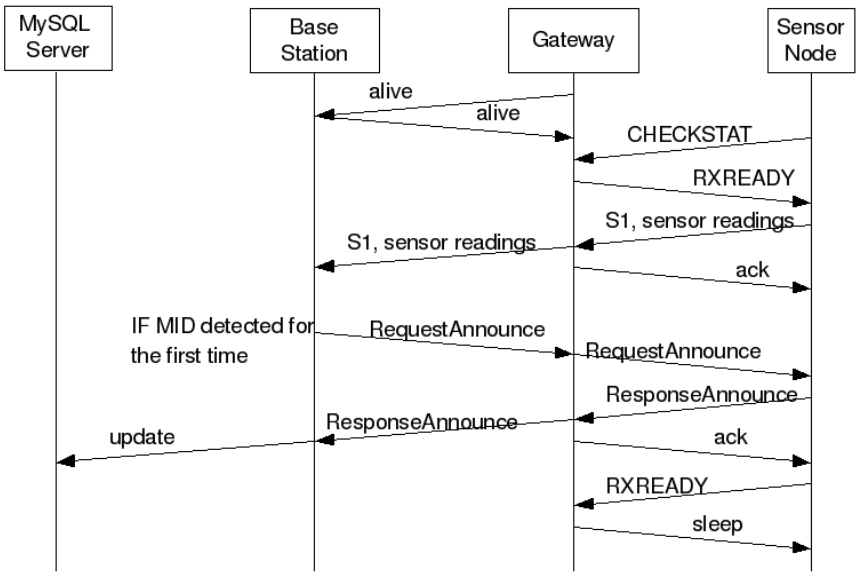

Figure 15: Communication between the MySQL Server, Base station and nodes to announce a new node in the network.

9 Conclusion

During the base station and the node communication, the node sends it's nodeid (senderID) and the sessionID to the base station via the gateway. Whenever the base station gets a new node id or a new session id with respective to the same node id, the base station sends a RequestAnnounce message to the node. SWL compiler is updated to add Request Announce message. The node sent it's nodeUUID, sensorUUID, latitude, longitude, number of sensors attached to the node to the base station as Announce message. With the nodeUUID and sensorUUID, base station queries the SWL database for the rest of node and it's sensors information. If the database does not contain any record of that nodeUUID and sensorUUID, SWLAnnounce program updates the database by adding the nodeUUID and sensorUUID in the SENSOR and SENSORNODE table. The node location values are added to the POSIT table. The sessionID of each node are saved in Session table. At last this updated information is shown in the SWL Web browser. With this dynamic data adaption in WSN is achieved, that is when a new node enters into the network it automatically pops up in the web browser. To ensure end to end reliability DACK protocol is proposed to be used.

References

[1] J.-P. Arp and B. G. Nickerson, *A user friendly toolkit for building robust environmental sensor networks*, CNSR '07: Proceedings of the Fifth Annual Conference on Communication Networks and Services Research (Fredericton, NB, Canada), IEEE Computer Society, 2007, pp. 76–84.

[2] J. P. Arp, B. G. Nickerson, J. Lu, and Z. W. Sun, *Sensor web language communications protocol*, Tech. Report TR04-167, UNB Faculty of Computer Science, May 16 2004.

[3] Mark Priest Arthur Na, *Sensor observation service*, OpenGIS Implementation Standard, Version 1.0 (2007).

[4] Ke Deng, *Improving responsiveness of the sensor webs*, Master's thesis, The University of New Brunswick, Fredericton, NB, Canada, November 2008.

[5] B. G. Nickerson and J. Lu, *A language for wireless sensor webs*, CNSR '04: Proceedings of the Second Annual Conference on Communication Networks and Services Research (CNSR'04) (Fredericton, NB, Canada), IEEE Computer Society, May 19-21 2004, pp. 293–300.

[6] B. G. Nickerson, Z. Sun, and J. Arp, *A sensor web language for mesh architectures*, CNSR '05: Proceedings of the 3rd Annual Communication Networks and Services Research Conference (CNSR'05) (Halifax, NS, Canada), IEEE Computer Society, 2005, pp. 269–274.

[7] Bradford G. Nickerson and Ke Deng, *An adaptive programming model for environmental sensor networks using fuzzy logic*, CNSR '08: Proceedings of the 3rd Annual Communication Networks and Services Research Conference (CNSR'08), 2008, pp. 350–357.

[8] Arp John Paul and Bradford G. Nickerson, *The disseminated acknowledgement (dack) protocol for data collection in wireless sensor networks*, Poster at the 11th Annual GEOIDE Scientific Conference (Vancouver, B.C.), May 27-29 2009.

[9] Gunita Saini, *Automated sensor web adaptation to sensor networks*, UNB Faculty of Computer Science MCS thesis proposal (2009).

[10] Z. Song, *Mesh architecture for environmental sensor webs*, Master's thesis, The University of New Brunswick, Fredericton, NB, Canada, September 2005.

[11] Wikipedia, *Universally unique identifier*, September 2009.

A Header File - SWLMsg.h

SWLMsg.h defines the structure of SWLMsg and AckMsg. The enum below, assignes the SWL message_size types to the SWLmessages. Announce message is assigned 15 as the message type. The structures SWLMsg and AckMsg define the SWLMsg packet.

```
#include "TosTime.h"
#define BATTERY_PORT 7

enum {
  ALIVE    = 0,
  REQUEST    = 1,
  RESPONSE    = 2,
  REPORT    = 3,
  SWITCHG    = 4,
  ALERT    = 5,
  CHANGEI    = 8,
  CHECKSTAT    = 9,
  RXREADY   = 10,
  TXDONE    = 11,
  SLEEP    = 12,
  WAIT    = 13,
  RESET    = 14,
  ANNOUNCE        =  15,
// Add TSYNC message for helping time sync
};

enum {
 UP = 1,
 DOWN = 0,
};

enum {
 TEST_ID0 = unique("ByteEEPROM"),
 TEST_ID1 = unique("ByteEEPROM"),
 TEST_ID2 = unique("ByteEEPROM"),
 TEST_ID3 = unique("ByteEEPROM"),
 TEST_ID4 = unique("ByteEEPROM")
};

enum {
 AM_SWLMSG = 8,
```

```
 AM_ACKMSG = 9,
};

typedef struct SWLMsg {
    uint8_t ackSeqNo;
    uint16_t senderID; // 0x007E = TOS_UART_ADDR = Base Station
    uint8_t swlMsgType;
    uint8_t swlMsgID;
    uint16_t swlSeqNo;
    uint8_t message_size;
    uint8_t sessionid;
    uint8_t payload[22];
} SWLMsg;

typedef struct ACKMsg {
    uint8_t swlMsgType;
    uint8_t ackSeqNo;
    uint16_t receiverAddress;
} ACKMsg;

typedef struct SessionParam {
uint8_t sessionid;
}SessionParam;
```

B Base station code

B.1 SWLAnnounce.java

SWLAnnounce.java is the base station program. SWLAnnounce sends and recieves messages to a sensor node and a gateway. For announce mechanism, SWLAnnounce keep track of all nodes in WSN. If a node restarts or a new node comes in the WSN, SWLAnnounce asks that node to announce itself. The SWLAnnounce program also queries and updates the mysql database.

```
import java.sql.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import javax.comm.*;
import net.tinyos.util.*;
import net.tinyos.message.*;
import net.tinyos.packet.*;
```

```
public class SWLAnnounce implements MessageListener
{
  MoteIF moteif;
  SWLResponseCollector swlRC;
  SWLAnnResponseCollector swlARC;
  AnnResponseMsg ARM;
  FileOutputStream out;
  PrintStream p;
  Vector swlLines;
  PrintWriter log;
  Hashtable<Integer, Moteindex> Sessiontable;
  Connection conn;
  Statement stmt;
  ResultSet rs;
  short gateway;
  short swlMsgID;
  short sessionid;
  short largestID = 256;
  short payload[] = {0,0,0,0,0,0,0,0,
                                                0,0,0,0,0,0,0,0,
                                                    0,0,0,0,0,0};

  public SWLAnnounce()
  {
      int i;
      gateway = 0;
      swlMsgID = 0;
      swlLines = new Vector();
      swlRC = new SWLResponseCollector();
      swlARC = new SWLAnnResponseCollector(13);
      ARM = new AnnResponseMsg();
      Sessiontable = new Hashtable<Integer, Moteindex>();
      Calendar cal = Calendar.getInstance();

      String logdate = "log_"+cal.get(Calendar.DAY_OF_MONTH) + "-" +
                              (cal.get(Calendar.MONTH) + 1) + "-" +
                              cal.get(Calendar.YEAR) + ".dat";

      try
      {
         log = new PrintWriter( new FileWriter( new File (logdate), true) );
         moteif = new MoteIF(PrintStreamMessenger.err);
         //moteif = new MoteIF(BuildSource.makePhoenix("
```

```
  network@localhost:9001", PrintStreamMessenger.err));
       if (moteif == null)
      {    System.out.println("Invalid packet source ");
           System.exit(2);
      }
      }
    catch (Exception e)
    {    System.out.println("Failed to access mote: " + e);
         System.exit(2);
    }

    try
    {
       Class.forName("com.mysql.jdbc.Driver").newInstance();
       conn = DriverManager.getConnection( "jdbc:mysql://localhost/swl",
                                           "swluser", "swlsecret" );
       stmt = conn.createStatement();

       log = new PrintWriter( new FileWriter( new File (logdate), true) );

       moteif = new MoteIF(PrintStreamMessenger.err);
       if (moteif == null)
       {    System.out.println("Invalid packet source");
            System.exit(2);
       }
    }
      catch (Exception e)
      {    System.out.println("Failed to access mote: " + e);
           System.exit(2);
      }
}

public void run()
{
     moteif.registerListener(new SWLMsg(), this);
     moteif.registerListener(new ACKMsg(), this);
     moteif.start();
}

public void sendMsg(String cmd)
{
     SWLMsg msg1 = new SWLMsg();
```

```
try
{
     File file = new File(cmd);
     BufferedReader br = new BufferedReader(new FileReader(file));

     int numLines = 0;
     br.readLine();
     String line =  br.readLine();

     while((line!=null) || !(line.trim().compareTo("}")==0))
     {    StringTokenizer st = new StringTokenizer(line, " ");
          String str = st.nextToken();
          swlLines.add(str);
          line = br.readLine();
          if (line.trim().compareTo("}")==0) {break;}
     }

}
catch(IOException e)
{    e.printStackTrace();
}

int val = 0;
int rxTime = 10;

try
{
    swlRC = new SWLResponseCollector(swlLines.size());
    //swlARC = new SWLAnnResponceCollector(13);

    for (int i = 0; i < swlLines.size(); i++)
    {
         char tstWordArray[] = ((String)swlLines.get(i)).toCharArray();

         // convert the string into a list of integers.
         for (int j = 0; j < payload.length; j++)
         {    if (j < tstWordArray.length)
              {    payload[j] = (short)tstWordArray[j];
              }
              else
              {    payload[j] = 0;
              }
         }
```

```
                SWLMsg rmsg = new SWLMsg();
                rmsg.set_swlMsgType((short)1);
                rmsg.set_swlMsgID(swlMsgID);
                rmsg.set_swlSeqNo((short)(i+1));
                rmsg.set_senderID(0x007E);
                rmsg.set_payload(payload);
                //moteif.send(MoteIF.TOS_BCAST_ADDR, rmsg);
                moteif.send(gateway, rmsg);

                int crc = Crc.calc(rmsg.toString().getBytes(),
                      rmsg.toString().getBytes().length - 2);

                System.out.println(tstWordArray + "\t" +
                                                    rmsg.get_swlMsgType() + "\t" +
                                                    rmsg.get_swlMsgID() + "\t" +
                                                    rmsg.get_swlSeqNo() + "\t" +
                                                    crc);
                 Thread.sleep(350);
            }

     }
     catch(Exception e)
     {    e.printStackTrace();
     }
}

public void messageReceived(int to, Message m)
{
    //System.out.print("Recieved a message:");
    Calendar newCal = Calendar.getInstance();

    String time = newCal.get(Calendar.HOUR_OF_DAY) + ":" +
                      newCal.get(Calendar.MINUTE) + ":" +
                      newCal.get(Calendar.SECOND);

    if (m instanceof SWLMsg)
    {
        SWLMsg msg = (SWLMsg)m;
        int id = (int)msg.get_senderID();
        int sess_id = (int)msg.get_sessionid();
        int seqNo = (int)msg.get_swlSeqNo();
        String payload = msg.getString_payload();
```

```
int nodeid;

if (Sessiontable.containsKey(id))
{
     Moteindex mi = Sessiontable.get(id);

     if (mi.sessionid != sess_id)
     {
         System.out.println("Announce");
         mi.id=id;
         mi.sessionid=sess_id;
         SWLAnnounce announce = new SWLAnnounce();
         swlARC = new SWLAnnResponseCollector(13);

         try
         {
           // Create a new file output stream
              out = new FileOutputStream("TestAnnounce2.txt");

           // Connect print stream to the output stream
               p = new PrintStream( out );
               p.println ("request{");
               p.println ("s"+id+".ga();");
               p.println ("}");
               p.close();
         }
         catch (Exception e)
         {
                System.err.println ("Error writing to file");
         }

         try
         { announce.sendMsg("TestAnnounce2.txt");
         }
         catch(Exception e){
           e.printStackTrace();
         }
     }
}
else
{   Moteindex mi = new Moteindex();
    mi.id=id;
    mi.sessionid=sess_id;
```

```
    Sessiontable.put(id, mi);
    /*SWLtest announce1 = new SWLtest();
    try
    { announce1.run("TestAnnounce.txt");
    }
    catch(Exception e){
      e.printStackTrace();
    }*/
}
if(msg.get_swlMsgType() == 0)
{
   try
   {   msg.set_senderID(0x007E);
       moteif.send(id, msg);
   }
   catch(Exception e)
   {   e.printStackTrace();
   }
 }

 System.out.println(time + ":\t" + (int)msg.get_swlMsgID() + "\t"
                                 + (int)msg.get_swlSeqNo() + "\t"
                                 + (int)msg.get_swlMsgType() + "\t"
                                 + (int)msg.get_sessionid() + "\t"
                                 + (int)msg.get_senderID() + "\t"
                                 + msg.getString_payload());
 log.println(time + ":\t"+ (int)msg.get_swlMsgID() + "\t"
                         + (int)msg.get_swlSeqNo() + "\t"
                         + (int)msg.get_swlMsgType() + "\t"
                         + (int)msg.get_sessionid() + "\t"
                         + msg.getString_payload());

 if (msg.get_swlMsgType() == 2)
 {
     short incoming[] = msg.get_payload();
     char incChar[] = new char[incoming.length];

     for (int i = 0; i < incoming.length; i++)
     {    incChar[i] = (char)incoming[i];
     }

     String input = new String(incChar);
     swlRC.addLine((int)msg.get_swlSeqNo(), input);
```

```
            if(swlRC.ready())
            {    System.out.println("response ready:");
                 System.out.println(swlRC.toString());
                 //System.exit(0);
            }
        }
        if(msg.get_swlMsgType() == 15)
        {
             short incoming[] = msg.get_payload();
             char incChar[] = new char[incoming.length];

             for (int i = 0; i < incoming.length; i++)
             {    incChar[i] = (char)incoming[i];
             }

             String input = new String(incChar);
             swlARC.addLine((int)msg.get_swlSeqNo(), input);

             if(swlARC.ready())
             {    System.out.println("response ready:");
                  System.out.println(swlARC.toString());
                  ARM.parseAnnResponse(id, swlARC.totalLines, swlARC.response);
                  databaseOperations(id);
                  //System.exit(0);
             }
        }
    }

    if (m instanceof ACKMsg)
    {    ACKMsg msg = (ACKMsg)m;
         System.out.println("\t ACK! \t" + (int)msg.get_swlMsgType() + "\t"
                                          + (int)msg.get_ackSeqNo() + "\t"
                                          + (int)msg.get_receiverAddress() );
         log.println("\t ACK! \t" + (int)msg.get_swlMsgType() + "\t"
                                   + (int)msg.get_ackSeqNo() + "\t"
                                  + (int)msg.get_receiverAddress());
    }
}

public void databaseOperations(int SenderID)
{
    String dnUUID="";
```

```
int stot_rec = 0;
int ntot_rec = 0;
int postot_rec = 0;
int j=0;
String node_lat="";
String node_lon="";
int dsID[] = new int[ARM.nsensors];
String dsmanufacturer[] = new String[ARM.nsensors];
String dsdescrip[] = new String[ARM.nsensors];
String dsmodel[] = new String[ARM.nsensors];
String dsname[] = new String[ARM.nsensors];
String dsunit[] = new String[ARM.nsensors];
int dsattnodeid[] = new int[ARM.nsensors];
String dsUUID[] = new String[ARM.nsensors];
String dspositID[] = new String[ARM.nsensors];

//Query for Sensor Node table
try
{
    rs = stmt.executeQuery("SELECT DISTINCT UUID from SENSORNODE where id="
                                                                    +SenderID);
    while(rs.next()) {
       dnUUID = rs.getString("UUID");
    }
    if(ARM.nUUID.equals(dnUUID))
    {
        rs = stmt.executeQuery("SELECT * from SENSORNODE where UUID="+"'"
                                                          +ARM.nUUID+"'");
        while(rs.next()) {
            int dnID = rs.getInt("id");
            String dnmanufacturer = rs.getString("manufacturer");
            String dnmodel = rs.getString("model");
            String dnname = rs.getString("name");
            String dntype = rs.getString("type");
            int dnGCDID = rs.getInt("attached_GCD_id");
            int dnnetID = rs.getInt("sensor_network_id");
            String dnpositID = rs.getString("posit_id");
            dnUUID = rs.getString("UUID");
            //System.out.println(dnID+" "+dnmanufacturer+" "+dnmodel+" "+dnname
                              +" "+dntype+" "+dnGCDID+" "+dnnetID+" "+dnUUID);
        }
    }
    else
```

```
 {
      rs = stmt.executeQuery("SELECT count(*) as num_record from SENSOR");
      while(rs.next()) {
      ntot_rec = rs.getInt("num_record");
      }
      int ncount = ntot_rec +1;
      stmt.executeUpdate("INSERT INTO SENSORNODE(ID,"
                                      +"UUID)"
                                      +"VALUES"+"("+ncount+","
                                      +"'"+ARM.nUUID+"')");
      System.out.println("Writing to database complete!!!");
 }
 //Query for Sensor table
rs = stmt.executeQuery("SELECT DISTINCT UUID from SENSOR");
while(j<ARM.nsensors)
{
    while(rs.next()) {
        dsUUID[j] = rs.getString("UUID");
        break;
     }
     j++;
 }

 for(int i=0; i<ARM.nsensors;i++)
 {
     if(ARM.sUUID[i].equals(dsUUID[i]))
     {
        rs = stmt.executeQuery("SELECT * from SENSOR where UUID="+"'"
                                                +ARM.sUUID[i]+"'");
        while(rs.next()) {
        dsID[i] = rs.getInt("id");
        dsmanufacturer[i] = rs.getString("manufacturer");
        dsdescrip[i] = rs.getString("description");
        dsmodel[i] = rs.getString("model");
        dsname[i] = rs.getString("name");
        dsunit[i] = rs.getString("unit");
        dsattnodeid[i] = rs.getInt("attached_Node_id");
        dsUUID[i] = rs.getString("UUID");
        dspositID[i] = rs.getString("posit_id");
    //System.out.println(dsID[i]+" "+dsmanufacturer[i]+" "+dsdescrip[i]+" "
  +dsmodel[i]+" "+dsname[i]+" "+dsunit[i]+" "+dsattnodeid[i]+" "+dsUUID[i]);
         }
     }
```

```
            else
            {
               rs = stmt.executeQuery("SELECT count(*) as num_record from SENSOR");
               while(rs.next()) {
               stot_rec = rs.getInt("num_record");
               }
               int scount = stot_rec +1;
               stmt.executeUpdate("INSERT INTO SENSOR(ID,"
                                        +"UUID)"
                                        +"VALUES"+"("+scount+","
                                        +"'"+ARM.sUUID[i]+"')");
               System.out.println("Writing to database complete!!!");
            }
         }

         //Query for POSIT table
         rs = stmt.executeQuery("SELECT latitude, longitude from POSIT where UUID='"
                                                                    +ARM.nUUID+"'");
         while(rs.next()) {
         node_lat = rs.getString("latitude");
         node_lon = rs.getString("longitude");
         }

         if(!(ARM.lttot.equals(node_lat) && ARM.lntot.equals(node_lon)))
         {
             rs = stmt.executeQuery("SELECT count(*) as num_record from POSIT");
             while(rs.next())
             {
                 postot_rec = rs.getInt("num_record");
             }
             int poscount = postot_rec +1;
             stmt.executeUpdate("update POSIT SET end_date =CURDATE() where UUID='"
+ARM.nUUID+"'");
             stmt.executeUpdate("INSERT INTO POSIT(ID,"
                                                  +"UUID,"
                                                  +"LATITUDE,"
                                                  +"LONGITUDE,"
                                                  +"START_DATE)"
                                        + "VALUES" +"("+poscount+","
                                                  +"'"+ARM.nUUID+"',"
                                                  +"'"+ARM.lttot+"',"
                                                  +"'"+ARM.lntot+"',"
                                                  +"CURDATE()"+")");
```

```
                System.out.println("Writing to the database complete!!!!!!");
            }
        }
        catch(Exception e)
        {
            e.printStackTrace();
        }
    }

    public static void main(String args[])
    {
         if (args.length != 1)
         {    System.err.println("Usage: java SWLAnnounce [SWLmessage]");
              System.exit(2);
         }
         System.out.println("\nSWLAnnounce started. Running request: "+args[0] );

         SWLAnnounce reader = new SWLAnnounce();

         try
         {    reader.run();
              reader.sendMsg(args[0]);
         }
         catch (Exception e) {
              e.printStackTrace();
         }
    }
}
```

B.2 SWLAnnResponceCollector.java

SWLAnnResponseCollector.java is an object class used by SWLAnnounce.java. SWLAnnResponseCollector is used to collect the Announce messages.

```
import java.lang.*;
import java.util.*;
import java.util.*;
import java.io.*;

public class SWLAnnResponseCollector
{
    String response[];
    boolean collected[];
    public int totalLines;
```

```
public int j=0, count=0;

public SWLAnnResponseCollector () {}

public SWLAnnResponseCollector (int n)
{
    totalLines = n+2;
    response = new String[totalLines];
    collected = new boolean[totalLines];

    response[0]= "response{";
    collected[0] = true;

    for (int i = 1; i < totalLines-2; i++)
    {    response[i] = "";
         collected[i] = false;
    }

    response[totalLines-1] = "}";
    collected[totalLines-1] = true;
}

public void reset(int n)
{
    totalLines = n+2;
    response = new String[totalLines];
    collected = new boolean[totalLines];
}

public void addLine(int lineNo, String line)
{
    response[lineNo] = line;
    collected[lineNo] = true;

    if (collected[3] == true)
    {
       String id = response[3];
       int num_sensor = 0;
       StringTokenizer st = new StringTokenizer(id, ";");
       String long1 = st.nextToken(); if(!st.hasMoreTokens()) return;
       String sensor = st.nextToken();
       num_sensor = Integer.parseInt(sensor.substring(2));
       int tot_size;
```

```
            tot_size = 3 + num_sensor;
            totalLines = tot_size+2;
            response[totalLines-1] = "}";
            collected[totalLines-1] = true;
        }
    }

    public boolean ready()
    {
          boolean result = true;
          for (int i = 0; i < totalLines; i++)
          {    if (collected[i] == false)
               {    result = false;
               }
          }
          return result;
     }

     public String toString()
     {    String str = "";
          for (int i = 0; i < totalLines; i++)
          {    str = str + response[i] + "\n";
          }
          return str;
     }
}
```

B.3 Moteindex.java

Moteindex.java is an object class for the SWLAnnounce. While, the SWLAnnounce communicates with the sensor nodes, Moteindex.java saves the session ID of each node with respect to the sender ID.

```
import java.util.*;

public class Moteindex
{
  public int id;
  public int sessionid;

  public Moteindex()
  { id = 0;
    sessionid = 0;
```

```
  }
  public Moteindex(int i, int s)
  { id = i;
    sessionid = s;
  }
};
```

C Browser Applet code

C.1 SensorNetworkBridge

SensorNetworkBridge is a java program enabling serial communication on a com port when the gateway is directly connected to the base station.

```
import java.io.*;
public class SensorNetworkBridge
{   public static void main(String[] args)
        {   Bridge bridge = new Bridge();
            BaseStationSocketServer b = new BaseStationSocketServer(9003, bridge);
            ClientSocketServer c = new ClientSocketServer(9001, bridge);
            bridge.setBaseStationSocketServer(b);
            bridge.setClientSocketServer(c);

            c.start();
            b.start();
    }
}
```

C.2 SerialPlug

SerialPlug is a java program connecting the SensorNetworkBridge with the gateway.

```
import java.io.*;
import java.net.*;
import java.util.*;
import javax.comm.*;


public class SerialPlug extends Thread implements SerialPortEventListener
{
     CommPortIdentifier portId;
     InputStream serialInputStream;
```

```
OutputStream serialOutputStream;
SerialPort serialPort;
Socket socket;
Thread readThread;

public static void main(String[] args)
{    SerialPlug sp = new SerialPlug();
     sp.start();
}

public SerialPlug()
{
    try
    {
        //System.out.println(CommPortIdentifier.getPortIdentifiers());
        portId = CommPortIdentifier.getPortIdentifier("/dev/ttyS0");
        //socket = new Socket("131.202.243.10", 9003);
        socket = new Socket("127.0.0.1", 9003);
     }
     catch (Exception e)
     {
         e.printStackTrace();
     }
     try
     {
         serialPort = (SerialPort) portId.open("ClientSerialServerApp", 2000);
     }
     catch (PortInUseException e)
     {    e.printStackTrace();
     }

     try
     {
          serialInputStream = serialPort.getInputStream();
          serialOutputStream = serialPort.getOutputStream();
     }
     catch (IOException e)
     {}

     try
     {
          serialPort.addEventListener(this);
     }
```

```
            catch (TooManyListenersException e)
            {}

        serialPort.notifyOnDataAvailable(true);

        try
        {
            //serialPort.setSerialPortParams( 19200,
            serialPort.setSerialPortParams( 57600,
                            SerialPort.DATABITS_8,
                            SerialPort.STOPBITS_1,
                            SerialPort.PARITY_NONE );
        }
        catch (UnsupportedCommOperationException e)
        {}

        readThread = new Thread(this);
        readThread.start();
    }

      /* Method declaration**/
    public void run()
    {
      try
      {
       BufferedReader streamReader = new BufferedReader(new InputStreamReader(socket.
                                                                    getInputStream()));
       while (true)
       {
          try { Thread.sleep(20); } // don't hog cpu
          catch (Exception e) { e.printStackTrace(); }
          if (socket.getInputStream().available() > 0)
          {
             while (socket.getInputStream().available() > 0)
             {
                 int b = socket.getInputStream().read();
                 serialOutputStream.write(b);
                 serialOutputStream.flush();

             }
          }
        }
      }
```

```
        catch (Exception e)
        { e.printStackTrace();
        }
    }

    public void serialEvent(SerialPortEvent event)
    {
        switch (event.getEventType())
        {
            case SerialPortEvent.BI:
            case SerialPortEvent.OE:
            case SerialPortEvent.FE:
            case SerialPortEvent.PE:
            case SerialPortEvent.CD:
            case SerialPortEvent.CTS:
            case SerialPortEvent.DSR:
            case SerialPortEvent.RI:
            case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
            break;
            case SerialPortEvent.DATA_AVAILABLE:
            byte[] readBuffer = new byte[20];

            try
            {
                while (serialInputStream.available() > 0)
                {
                    int b = serialInputStream.read();
                    socket.getOutputStream().write(b);
                    socket.getOutputStream().flush();
                }
             }
             catch (IOException e)
             {}
           break;
        }
    }
}
```