Start Guide for Multi-hop Communication

Setting up multi-hop communication for the ISHM Software

Illinois Structural Health Monitoring Project

January 2011

ILLINOIS STRUCTURAL HEALTH MONITORING PROJECT
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
OPEN SYSTEMS LABORATORY & SMART STRUCTURES TECHNOLOGY LABORATORY

Table of Contents

1. Introduction

This document will guide you through the process of installing and configuring the Illinois Structural Health Monitoring Project (ISHMP) Toolsuite with multi-hop communication capability.

This guide will allow you to:

- Compile and program the motes with multi-hop communication capability
- Fine-tune network settings for efficient multi-hop communication
- Test multi-hop communication
- Trouble-shoot the network

This guide assumes that both TinyOS and the ISHMP Services Toolsuite have been successfully completed. Comprehensive guides for installation of TinyOS and the ISHMP Services Toolsuite are available on the Illinois Structural Health Monitoring Project (ISHMP) website at: http://shm.cs.uiuc.edu/documentation.html. The first part of the *Getting Started for Advanced Users and Developers* guide walks you through the process of setting up the PC environment required to interface with the Imote2. The second part of the Advanced Users guide walks you through the installation of the ISHMP services Toolsuite. The *User's Guide* is designed for the general users who want to build basic structural health monitoring (SHM) application with wide functionality of ISHMP Services Toolsuite.

All applications in the ISHMP Services Toolsuite have multi-hop communication capability. For the sake of simplicity, examples in this guide are based on a specific application: "RemoteSensing." For complete instructions on how to install and work with this application please refer to the *User's Guide*.

Comments and questions:

If you have questions about the software or this guide, or run into problems, please join us on the Imote2 discussion forum: http://vibration.shef.ac.uk/imote2_forum.

2. Equipment/Parts

Each smart sensor node consists of an Imote2 that provides the radio and processor, a sensor board that provides the sensing capability, and a battery board that provides an interface between the power source (batteries) and the Imote2. Figure 1 shows the top and bottom view of the Imote2 (left), the Imote2 stacked on a battery board with an external antenna (middle), and a sensor board with the Imote2 (right). The use of an external antenna is optional. Table 1 provides a list of the items needed to create a network of Imote2s.

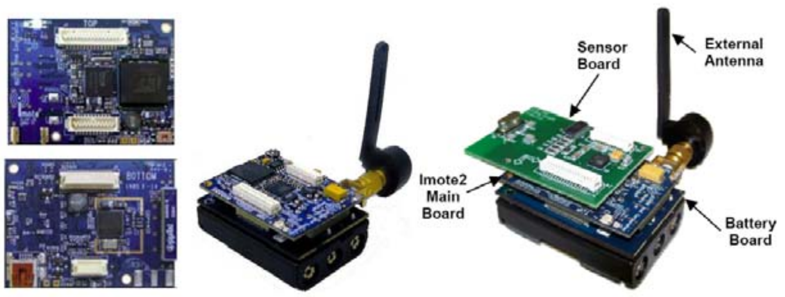


Figure 1. Composition of smart sensor node.

Table 1. Required components for programming and using a network of Imote2s

Item	Source/Vendor	Part No.
Imote2	MEMSIC	IPR2400[1]
Battery board	MEMSIC	IBB2400[2]
USB A to USB mini-B cable	MEMSIC or others	
Debug/Interface Board	MEMSIC	IIB2400
ISM sensor board OR ITES sensor board	MEMSIC	ISM400 ITS400

You will need at least two Imote2s to create a "network." One Imote2 connects to the PC via USB and acts as the gateway between the PC and your network of remote sensors. In this guide, the Imote2 that is connected to the PC will be referred to as the *gateway node* and the nodes that make up the network will be referred to as the *leaf nodes* (see Figure 2).

[1] There are two versions of the Imote2 available from MEMSIC. DO NOT order the .NET edition (IPR2410), which comes preloaded with software that is incompatible with the ISHMP software.

[2] Battery boards come with the Imote2s when purchased from MEMSIC.

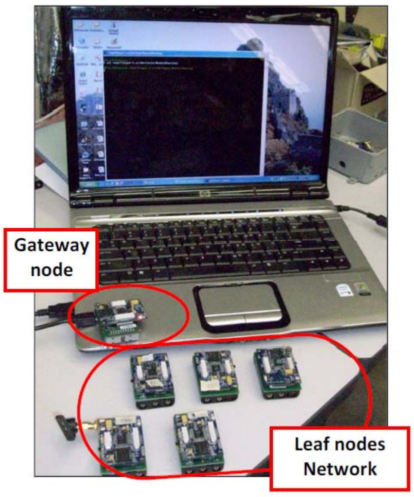


Figure 2.Sample network.

The interface/debug board (IIB, pictured in Figure 3 without (left) and with (right) the Imote2) is required for data collection from the gateway node to the PC. It provides two serial port (UART) interfaces over the USB connection to your PC, one of which communicates debug commands and output, and the other which communicates data.

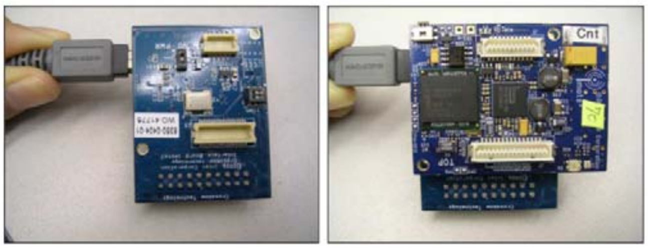


Figure 3. Interface board (IIB2400).

The Imote2s are programmed via USB. Please refer to the *Getting Started for Advanced Users and Developers* guide for more information regarding how USB and the IIB are used to program/interact with the Imote2 (http://shm.cs.uiuc.edu/documentation.html).

2.1. Sensor Board Options

There are two options for measuring acceleration with the Imote2. One is the ITS400 sensor board (pictured in Figure 4, left). The second is the ISM400 (formerly SHM-A) sensor board (pictured in Figure 4, right), which was developed at the University of Illinois at Urbana-Champaign. Both sensor boards are available from MEMSIC and supported by the software provided in the ISHMP Services Toolsuite. In addition to performance differences, the sensor boards support different sampling rates and have different minimum battery requirements.

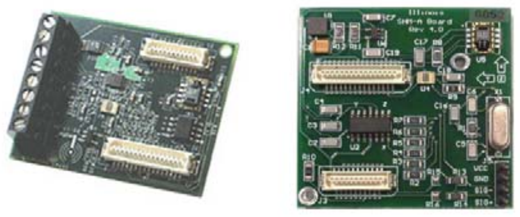


Figure 4. Sensor board: ITS400(left), ISM400(right).

The nominal sampling rates for the ITS400 sensor board are shown in Table 2. The minimum battery voltage (when using the IBB2400 battery board) is 3.6V. More information on the ITS400 can be found on the MEMSIC website at http://memsic.com/support/documentation/wireless-sensor-networks/category/7-datasheets.html?download=137%3Aits400.

Table 2. Supported sampling rates for the ITS400 sensor board[3].

Sample Rate (Hz)	Digital cut-off frequency (Hz)
280 (40)	70 (10)
560 (160)	140 (40)
1120(640)	280 (160)
4480 (2560)	1120 (640)

The default sampling rates supported by the ISHMP software for the ISM400 sensor board are given in Table 3. Additional sampling rates and channel configurations may be created. Please refer to the "ISM400 Advanced User's Guide" for instructions on creating new parameters. The minimum battery voltage required for the ISM400 sensor board is 3.7V (when using the IBB2400 battery board). More information can be found in the datasheet for the ISM400 board (http://shm.cs.uiuc.edu/hardware.html).

Table 3. Supported default sampling rates for the ISM400 sensor board

Sample Rate (Hz)	Digital cut-off frequency (Hz)
25	10
50	20
100	40
280	70

[3] From the data sheet for the accelerometer on the ITS400 sensor board: http://www.st.com/stonline/products/literature/ds/10175.pdf.

2.2. Battery Board

The battery board that comes with the Imote2s from MEMSIC takes three AAA batteries connected in series. The nominal voltage of one AAA battery is 1.5V, thus the nominal voltage supplied by three batteries is 4.5V. Typical new AAA batteries actually start out with higher than the nominal voltage output. The result is that three new AAA batteries can exceed the maximum battery voltage allowed by the IBB2400 batter boards of 4.7V. If this happens, a safeguard mechanism on the battery board will stop it from supplying power to the Imote2. In this case, it may be necessary to use one slightly used battery with two newer batteries to ensure the voltage is below the maximum allowed. The voltage coming from the batteries may easily be tested with a voltmeter as shown in Figure 5.

Figure 5. Battery voltage check with a voltage meter.

3. Software

We have developed a new method for reliable multi-hop communication that takes into account the topology and link failure to optimize the throughput while minimizing the energy consumption. We have identified the principal factors affecting the performance of multi-hop routing, and developed a variant of the AODV protocol to provide multi-hop routing and data transfer.

This guide has been written for use with the open-source software available through the Illinois Structural Health Monitoring Project (ISHMP) at http://shm.cs.uiuc.edu/. This section provides a general description of our multi-hop algorithm and describes the steps required for compiling and running applications with multi-hop capability. Section 4 describes parameter tuning, section 5 describes steps for multi-hop test and finally section 6 describes troubleshooting.

3.1. Ad-hoc On-demand Distance Vector routing

Ad-hoc On-demand Distance Vector routing (AODV) is one of the most widely studied and popular routing algorithms. As a prime characteristic of the AODV routing protocol, route discovery packets are disseminated only when necessary. Using this protocol, there is no need for general topology maintenance, i.e., only local connectivity information is preserved. Another advantage of this routing protocol is that route information is stored only on nodes that are involved in routing. In this protocol, when a node needs to send a message to another node in the network, it searches its route table for a route to the destination. If there is no route, a RREQ message is initiated. The RREQ message is then advertised into the network. Once a node that has a route to the destination receives the RREQ message, it generates a RREP message. Once receiving the RREP messages, the source node chooses a path with the minimum number of hops. Figure 6 shows an example of AODV route discovery method.

To prevent RREQ packets from traveling all across the sensor network indefinitely, the lifetime, and therefore the maximum distance a RREQ packet can travel is limited by the Time-To-Live (TTL) value.

Figure 6. An example of AODV route discovery method. Here, node A is the source and I is the destination. The source node disseminates a route request message. Nodes that receive the RREQ message rebroadcast; this process is repeated until the request reaches the destination node, or a node that has a route to the destination. At this point a route reply message is sent back to the source, notifying it that the route was found.

We have designed and implemented a data transfer service that allows for reliable multi-hop communication among the WSSN nodes as an extension to the currently available services in the ISHMP Toolsuite. Multi-hop communication is implemented underneath the reliable communication protocol. Moreover, as a method for further optimizing the performance of reliable data transfer, we have modified the original AODV routing scheme with the aim of providing multi-hop routes with higher reliability than those offered by the standard algorithm. Notably, this service is compatible with all existing applications that currently use the single-hop reliable data transfer service.

3.2. Getting started

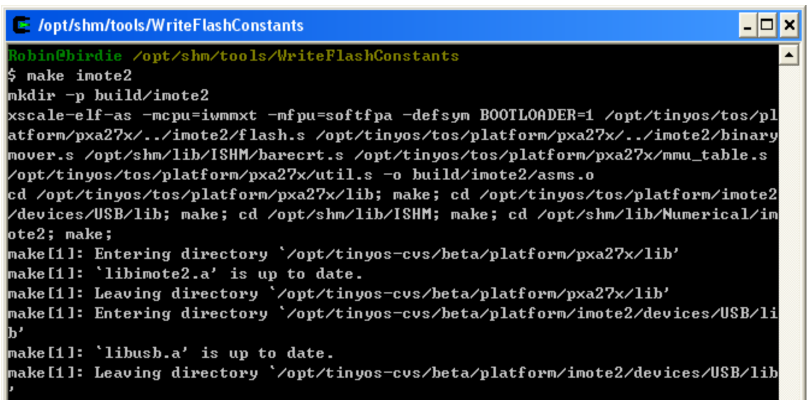
To ensure success in the use of the software provided by the ISHMP Services Toolsuite, it is important that the Getting Started for Advanced Users and Developers guide has been successfully completed (http://shm.cs.uiuc.edu/documentation.html). Also, ensure that you have downloaded the most recent version of the ISHMP Services Toolsuite (http://shm.cs.uiuc.edu/software.html).

3.3. Compiling and programming the motes

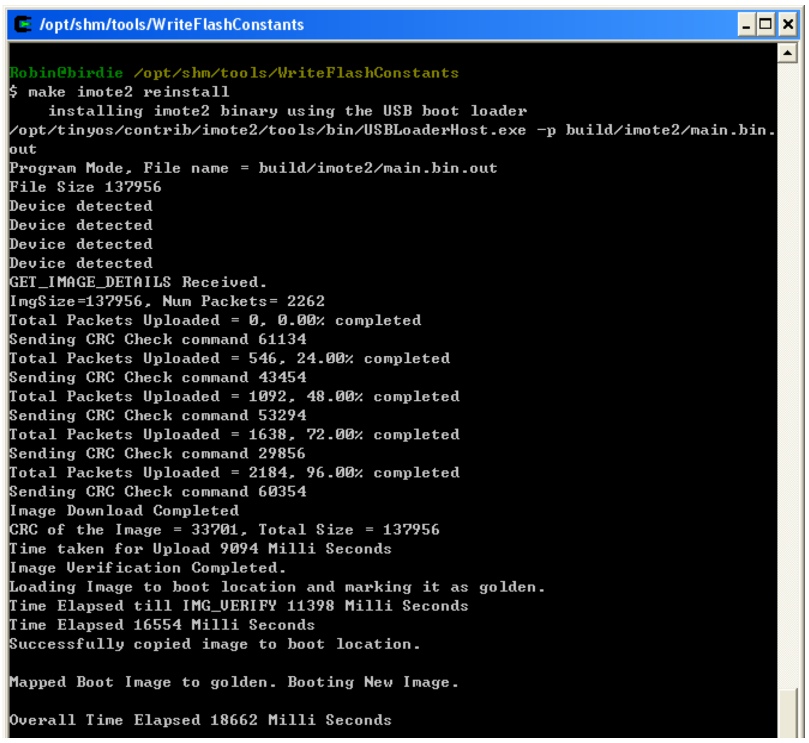
The following steps guide you through application compilation and mote programming with multi-hop capability.

Step (1): Installing FlashConstants

Install *WriteFlashConstants* (/opt/shm/tools/WriteFlashConstants) application on your mote by running "`make imote2`" and "`make imote2 reinstall`" command. This will automatically determine if the flash constants are up to date, and update them if necessary. Green LED indicates that the flash constants are up to date. Repeat this for every mote before installing the application. For detailed instructions on how to install *WriteFlashConstants* please refer to *Getting Started for Advanced Users and Developers* guide.



```
/opt/shm/tools/WriteFlashConstants
Robin@birdie /opt/shm/tools/WriteFlashConstants
$ make imote2
mkdir -p build/imote2
xscale-elf-as -mcpu=iwmmxt -mfpu=softfpa -defsym BOOTLOADER=1 /opt/tinyos/tos/pl
atform/pxa27x/../imote2/flash.s /opt/tinyos/tos/platform/pxa27x/../imote2/binary
mover.s /opt/shm/lib/ISHM/barecrt.s /opt/tinyos/tos/platform/pxa27x/mmu_table.s
/opt/tinyos/tos/platform/pxa27x/util.s -o build/imote2/asms.o
cd /opt/tinyos/tos/platform/pxa27x/lib; make; cd /opt/tinyos/tos/platform/imote2
/devices/USB/lib; make; cd /opt/shm/lib/ISHM; make; cd /opt/shm/lib/Numerical/im
ote2; make;
make[1]: Entering directory `/opt/tinyos-cvs/beta/platform/pxa27x/lib'
make[1]: `libimote2.a' is up to date.
make[1]: Leaving directory `/opt/tinyos-cvs/beta/platform/pxa27x/lib'
make[1]: Entering directory `/opt/tinyos-cvs/beta/platform/imote2/devices/USB/li
b'
make[1]: `libusb.a' is up to date.
make[1]: Leaving directory `/opt/tinyos-cvs/beta/platform/imote2/devices/USB/lib
'
```



```
/opt/shm/tools/WriteFlashConstants
Robin@birdie /opt/shm/tools/WriteFlashConstants
$ make imote2 reinstall
    installing imote2 binary using the USB boot loader
/opt/tinyos/contrib/imote2/tools/bin/USBLoaderHost.exe -p build/imote2/main.bin.
out
Program Mode, File name = build/imote2/main.bin.out
File Size 137956
Device detected
Device detected
Device detected
Device detected
GET_IMAGE_DETAILS Received.
ImgSize=137956, Num Packets= 2262
Total Packets Uploaded = 0, 0.00% completed
Sending CRC Check command 61134
Total Packets Uploaded = 546, 24.00% completed
Sending CRC Check command 43454
Total Packets Uploaded = 1092, 48.00% completed
Sending CRC Check command 53294
Total Packets Uploaded = 1638, 72.00% completed
Sending CRC Check command 29856
Total Packets Uploaded = 2184, 96.00% completed
Sending CRC Check command 60354
Image Download Completed
CRC of the Image = 33701, Total Size = 137956
Time taken for Upload 9094 Milli Seconds
Image Verification Completed.
Loading Image to boot location and marking it as golden.
Time Elapsed till IMG_VERIFY 11398 Milli Seconds
Time Elapsed 16554 Milli Seconds
Successfully copied image to boot location.

Mapped Boot Image to golden. Booting New Image.

Overall Time Elapsed 18662 Milli Seconds
```

Step (2): Installing the application

In order to install the application with multi-hop capability you need to change the makefile so that multi-hop is enabled. As an example, consider the `RemoteSensing` application without snooze alarm.

Open the Makefile (shm/tools/RemoteSensing/Makefile) with a text editor and ensure that the correct sensor board is selected (by typing the sensorboard's name right after `SENSORBOARD =`). To enable multi-hop, uncomment the corresponding line (`USE_MULTIHOP = 1`) by removing the "#" from the beginning of the line. The Makefile should look something like the text in Figure 7 (when using the ISM400, formerly SHM-A, sensor board).

Compile and install the application on the gateway node and leaf nodes. For instructions on how to compile and install applications please refer to the *User's Guide*.

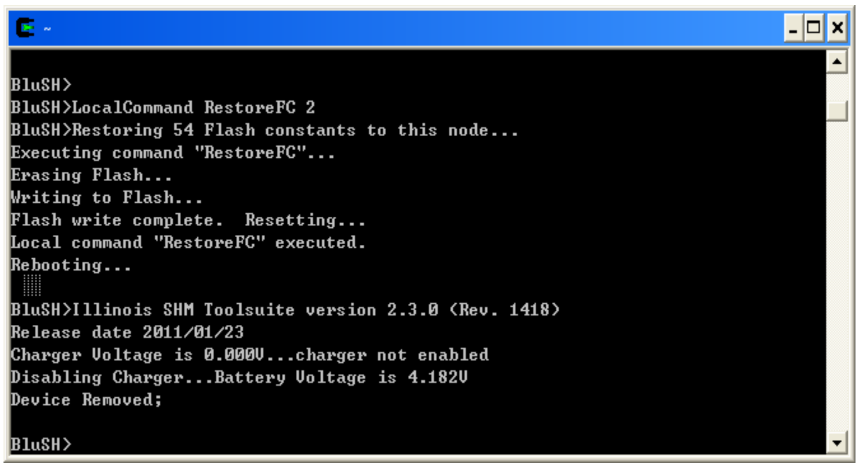
```
COMPONENT = RemoteSensing

# supported sensorboards: ITS400CA, ITS400CB, SHM_A, SHM_H, SHM_DAQ
SENSORBOARD = SHM_A

# configuration parameters, uncomment only if changing default
values
#UART_SPEED = UART_BAUD_921600
#SHMA_CUSTOM_FILTER_RATE = 1000
#RFPOWER = 31
#RFCHANNEL = 25

# optional components, uncomment to enable
USE_WATCHDOG = 1
#USE_SNOOZE_ALARM = 1
#USE_CHARGER_CONTROL = 1
USE_MULTIHOP = 1

include $(SHMLIB)/ISHM/Makerules
```

Figure 7. Makefile for RemoteSensing (shm/tools/RemoteSensing/Makefile).

In order to make sure that the correct set of `FlashConstants` are installed on the motes, run the following command on the gateway's Blush shell:

```
LocalCommand RestoreFC 2
```

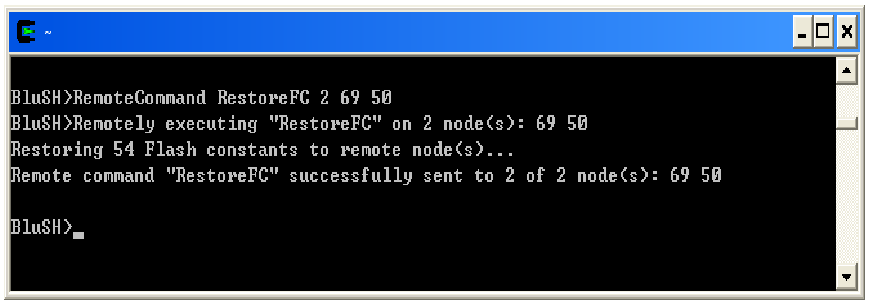
The Gateway will be reset after successfully Restoring FlashConstants.



```
BluSH>
BluSH>LocalCommand RestoreFC 2
BluSH>Restoring 54 Flash constants to this node...
Executing command "RestoreFC"...
Erasing Flash...
Writing to Flash...
Flash write complete.  Resetting...
Local command "RestoreFC" executed.
Rebooting...

BluSH>Illinois SHM Toolsuite version 2.3.0 (Rev. 1418)
Release date 2011/01/23
Charger Voltage is 0.000V...charger not enabled
Disabling Charger...Battery Voltage is 4.182V
Device Removed;

BluSH>
```

Also, run the following command for each leaf node:

```
RemoteCommand RestoreFC 2 <nodeId> [nodeId] …
```



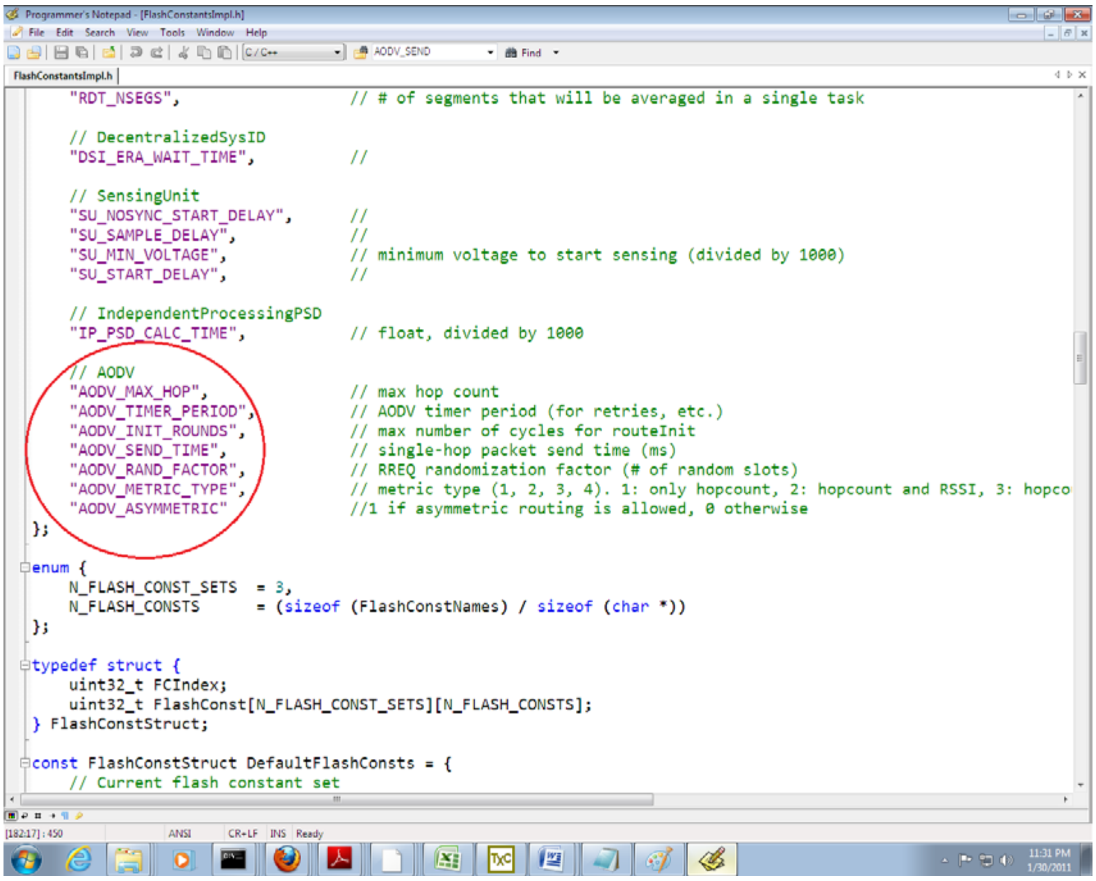
```
BluSH>RemoteCommand RestoreFC 2 69 50
BluSH>Remotely executing "RestoreFC" on 2 node(s): 69 50
Restoring 54 Flash constants to remote node(s)...
Remote command "RestoreFC" successfully sent to 2 of 2 node(s): 69 50

BluSH>_
```

Note: If your application is using SnoozeAlarm, make sure SnoozeAlarm is enabled by removing the "#" at the beginning of the corresponding line. In this case, for correct multi-hop communication functionality, make sure that Charger Control is enabled by removing the "#" from the beginning of the corresponding line (`USE_CHARGER_CONTROL = 1`). Also, SnoozeAlarm must be enabled only for leafnode. For the gateway node, compile the SnoozeAlarm disabled Makefile.

```
COMPONENT = RemoteSensing

# supported sensorboards: ITS400CA, ITS400CB, SHM_A, SHM_H,
SHM_DAQ
SENSORBOARD = SHM_A

# configuration parameters, uncomment only if changing default
values
#UART_SPEED = UART_BAUD_921600
#SHMA_CUSTOM_FILTER_RATE = 1000
#RFPOWER = 31
#RFCHANNEL = 25

# optional components, uncomment to enable
USE_WATCHDOG = 1
USE_SNOOZE_ALARM = 1
USE_CHARGER_CONTROL = 1
USE_MULTIHOP = 1

include $(SHMLIB)/ISHM/Makerules
```

4. Parameter tuning

There are a number of parameters that are used for fine tuning multi-hop communication (shm/lib/ISHM/FlashConstantsImpl.h). The FlashConstantsImpl.h contains three sets of flash constants. The third set (started after the "`// Set 2 - Multi-hop`" comment) is used when multi-hop communication is enabled. Parameters specific to the multi-hop implementation start after the "// AODV" comment.



```
    "RDT_NSEGS",                // # of segments that will be averaged in a single task

    // DecentralizedSysID
    "DSI_ERA_WAIT_TIME",        //

    // SensingUnit
    "SU_NOSYNC_START_DELAY",    //
    "SU_SAMPLE_DELAY",          //
    "SU_MIN_VOLTAGE",           // minimum voltage to start sensing (divided by 1000)
    "SU_START_DELAY",           //

    // IndependentProcessingPSD
    "IP_PSD_CALC_TIME",         // float, divided by 1000

    // AODV
    "AODV_MAX_HOP",             // max hop count
    "AODV_TIMER_PERIOD",        // AODV timer period (for retries, etc.)
    "AODV_INIT_ROUNDS",         // max number of cycles for routeInit
    "AODV_SEND_TIME",           // single-hop packet send time (ms)
    "AODV_RAND_FACTOR",         // RREQ randomization factor (# of random slots)
    "AODV_METRIC_TYPE",         // metric type (1, 2, 3, 4). 1: only hopcount, 2: hopcount and RSSI, 3: hopco
    "AODV_ASYMMETRIC"           //1 if asymmetric routing is allowed, 0 otherwise
};

enum {
    N_FLASH_CONST_SETS  = 3,
    N_FLASH_CONSTS      = (sizeof (FlashConstNames) / sizeof (char *))
};

typedef struct {
    uint32_t FCIndex;
    uint32_t FlashConst[N_FLASH_CONST_SETS][N_FLASH_CONSTS];
} FlashConstStruct;

const FlashConstStruct DefaultFlashConsts = {
    // Current flash constant set
```

Table 4 is a list of these parameters along with their description and default values.

Table 4. Multi-hop parameters.

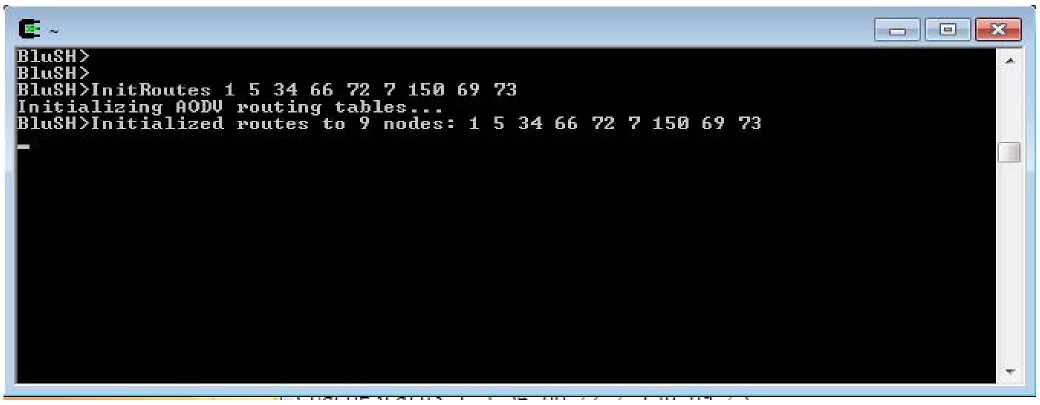
Parameter Name	Description	Default Value
AODV_MAX_HOP	Maximum allowed hops	5
AODV_TIMER_PERIOD	AODV timer period (for retries, etc.)	75
AODV_INIT_ROUNDS	Maximum number of cycles for route initiation	20
AODV_SEND_TIME	Single-hop packet send time (in (ms))	20
AODV_RAND_FACTOR	Randomization factor (number of random slots)	7
AODV_METRIC_TYPE	Metric type (1, 2, 3, 4). • 1: only hopcount, • 2: hopcount and RSSI, • 3: hopcount, RSSI, and LQI, • 4: hopcount and LQI	3
AODV_ASYMMETRIC	1 if asymmetric routing is allowed, 0 otherwise	1

The maximum allowed hops depends on the network size, topology and communication environment. The default value of 5 is for a network of 70 nodes, a linear topology, and an open area with direct line of sight between most nodes.

`AODV_SEND_TIME` is the time allocated for routing. If this parameter is too low, there won't be enough time for route establishment and the routing may fail; a too high value for `AODV_SEND_TIME` increases network delay. To check if this parameter has a proper value follow the steps below:

1. Run the `InitRoutes` command from the gateway node's BluSH shell. This command initializes AODV routing tables for the specified nodes. Include all leaf nodes for a better assessment:

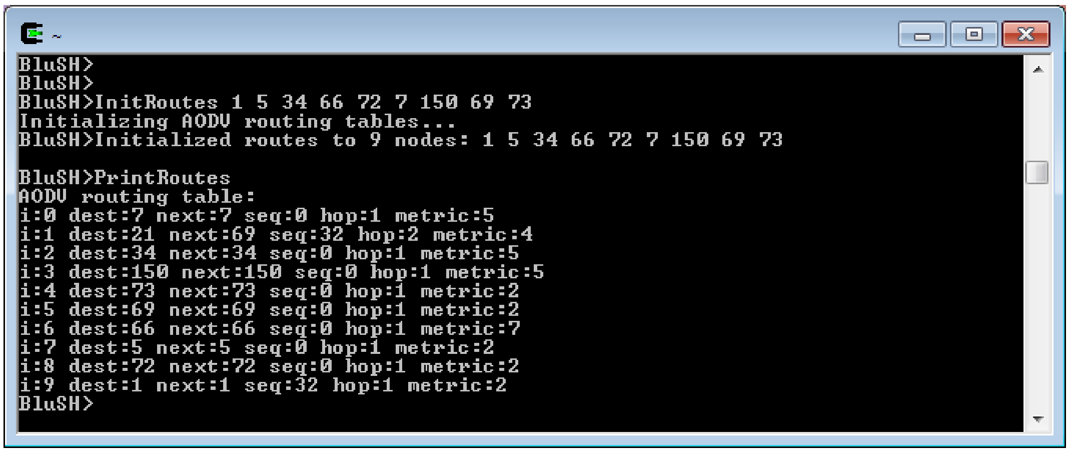
   ```
   InitRoutes 1 5 34 66 72 7 150 69 73
   ```



```
BluSH>
BluSH>
BluSH>InitRoutes 1 5 34 66 72 7 150 69 73
Initializing AODV routing tables...
BluSH>Initialized routes to 9 nodes: 1 5 34 66 72 7 150 69 73
```

2. Run the `PrintRoute` command from the gateway node's BluSH shell. This command prints the routing table on the node.

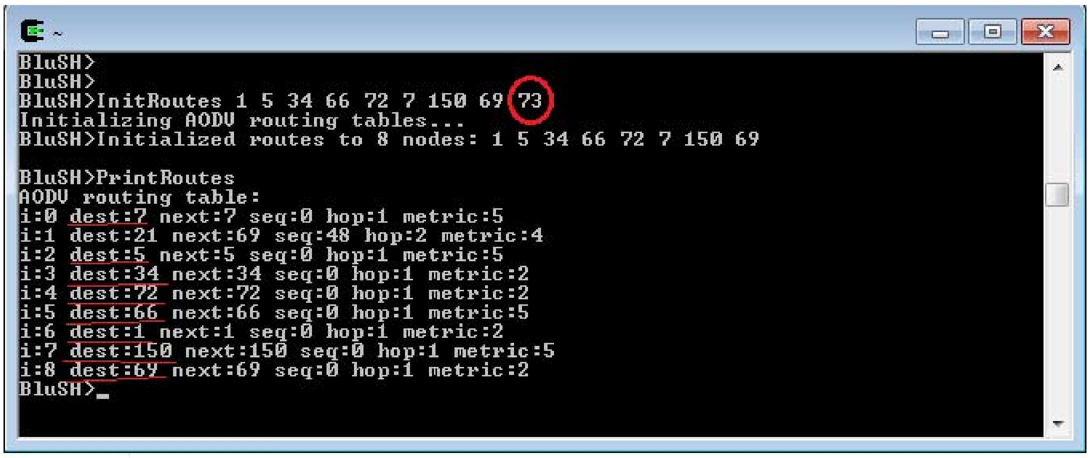
   ```
   PrintRoutes
   ```



```
BluSH>
BluSH>
BluSH>InitRoutes 1 5 34 66 72 7 150 69 73
Initializing AODV routing tables...
BluSH>Initialized routes to 9 nodes: 1 5 34 66 72 7 150 69 73

BluSH>PrintRoutes
AODV routing table:
i:0 dest:7 next:7 seq:0 hop:1 metric:5
i:1 dest:21 next:69 seq:32 hop:2 metric:4
i:2 dest:34 next:34 seq:0 hop:1 metric:5
i:3 dest:150 next:150 seq:0 hop:1 metric:5
i:4 dest:73 next:73 seq:0 hop:1 metric:2
i:5 dest:69 next:69 seq:0 hop:1 metric:2
i:6 dest:66 next:66 seq:0 hop:1 metric:7
i:7 dest:5 next:5 seq:0 hop:1 metric:2
i:8 dest:72 next:72 seq:0 hop:1 metric:2
i:9 dest:1 next:1 seq:32 hop:1 metric:2
BluSH>
```

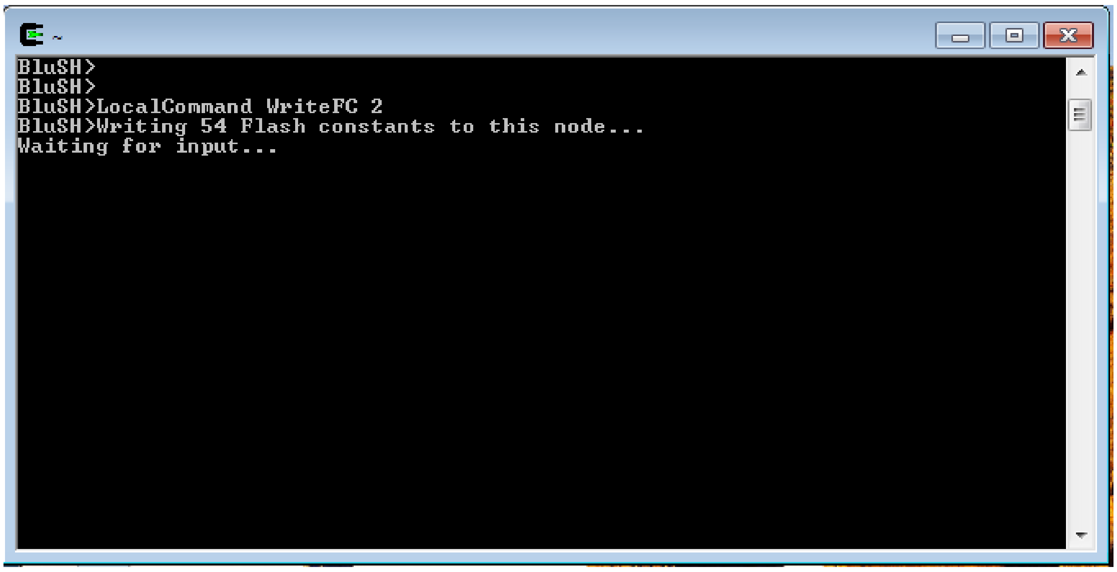
3. If some of the nodes are not in the `dest` field of the printed table it means that the gateway node was not able to find a route to that specific node. If the nodes that do not appear in the routing table are up and running, and have line-of-site to some node(s) in your network, you should increase the `AODV_SEND_TIME` by a small amount (typically by a value less than 5). In

order to make sure this increase is necessary you can run the `FlushRoutes` command from the node's Blush shell and repeat steps 1 to 3.


```
BluSH>
BluSH>
BluSH>InitRoutes 1 5 34 66 72 7 150 69 73
Initializing AODV routing tables...
BluSH>Initialized routes to 8 nodes: 1 5 34 66 72 7 150 69

BluSH>PrintRoutes
AODV routing table:
i:0 dest:7 next:7 seq:0 hop:1 metric:5
i:1 dest:21 next:69 seq:48 hop:2 metric:4
i:2 dest:5 next:5 seq:0 hop:1 metric:5
i:3 dest:34 next:34 seq:0 hop:1 metric:2
i:4 dest:72 next:72 seq:0 hop:1 metric:2
i:5 dest:66 next:66 seq:0 hop:1 metric:5
i:6 dest:1 next:1 seq:0 hop:1 metric:2
i:7 dest:150 next:150 seq:0 hop:1 metric:5
i:8 dest:69 next:69 seq:0 hop:1 metric:2
BluSH>_
```

In order to change the AODV_SEND_TIME flash constant follow the steps bellow:

I. In your Cygwin shell type:
`LocalCommand WriteFC 2`


```
BluSH>
BluSH>
BluSH>LocalCommand WriteFC 2
BluSH>Writing 54 Flash constants to this node...
Waiting for input...
```

II. Open another Cygwin shell to enter the input file. You can find a template for the input file in the "`lib/ISHM`" directory, under the name: "`fconst2.txt`". "`fconst2.txt`" is a sample input file that can be used for multi-hop communication, and "`fconst1.txt`", and "`fconst0.txt`" are for single-hop communication. Copy "`fconst2.txt`" to your Cygwin home directory and increase the value for: "`AODV_SEND_TIME`" and save the file.

III. In your Cygwin shell type:
```
autocomm –i fconst2.txt COMx
```
IV. Repeat steps I to III for all the leaf nodes by typing the following command on the first step:
```
LocalCommand WriteFC 2 nodeId
```

In the proposed multi-hop implementation, each route request packet carries a routing metric, which is calculated on each intermediate node. The implemented routing metric for multi-hop communication allows for 4 different options. The first option is similar to the routing metric used in the original AODV algorithm and chooses the route with the minimum number of intermediate hops. The major drawback of the hop-count routing metric is that it may lead to the selection of long links, which in turn results in an increase in the loss ratio and power consumption, as well as a decrease in signal strength indicator for the received packets. The other options allow for the use of RSSI, and LQI which are included in every received packet on CC2420. RSSI is the estimate of the signal power and is calculated over 8 symbol periods and stored in the RSSI VAL register. RSSI is highly correlated with packet reception rate except when operating at the edge of receiver sensitivity. The LQI value characterizes the quality of link over which the packet was transmitted. LQI is calculated for each received packet by the CC2420 radio, and measures the received energy level and/or SNR. LQI is expected to have a higher correlation with link reliability when the network topology is sparse, and poor-quality links are prevalent. Both RSSI and LQI are included in the header of each received packet. Based on the selected routing metric option, the route metric is a linear combination of the hop count, RSSI, and the LQI value for the received packet. The proposed metric provides a sufficiently reliable assessment of path quality, while adding close to zero overhead to the protocol. Moreover, the simplicity of using this metric along with the built- in metric of AODV is attractive.

In this implementation, non-identical routes are allowed between source and destination nodes. Asymmetric routing is optional and can be disabled by setting the FlashConstant: `AODV_ASYMMETRIC` to 0.

Note. The current metric calculation assumes the links are symmetric. This means that the quality of the link on the forward and backward direction is very close. While this is a valid assumption for indoor and short-scale tests, it might not hold for outdoor testbeds. In environments with asymmetric links, the current implementation of metric calculation may not provide a good assessment of link quality.

5. Testing multi-hop communication Using RemoteSensing Application

In order to test multi-hop communication, you should either have long distances between nodes, or have week (or no) antennas on the motes. Multi-hop communication can be used with any application, but in this section `RemoteSensing` is selected as an example. In order to test this application, prepare at least two leaf nodes stacked with appropriate sensor boards (ISM 400 are recommended) and a gateway node.

Start Guide for Multi-hop Communication

In the following steps for testing multi-hop, we will first test if the software works well with nodes at short distances and with antennas attached, and will then test with either long distance between nodes, or no antennas. If you are testing in an indoor environment in which all nodes can be reached in a single hop you can detach the antennas for testing. In an outdoor environment the same effect is provided by increased distances between nodes.

Setp1. The aim of this step is to verify if the software is working as expected in a single-hop environment.

I. Compile and program the motes
   Compile `WriteFlashConstants` and `RemoteSensing` applications to all of the motes you prepared. (Refer to Step 3.3 Compiling and programming the motes) Make sure you select `Multihop` in Makefile before you compile.

II. Set the nodes at the same place (single-hop range), with their antennas attached.

III. Run `RemoteSensing` application and verify that the application is working correctly.
   - Open two Cygwin windows.
   - In the first Cygwin shell, run:
     - `imote2comm –n -o out.txt COMx`
   - In the second window, run:
     - `imote2comm -d COMy`
     - Press <enter> a couple of times until you receive the "`BluSH>`" prompt.
   - Run `SetRSNodes <nodeId>` to initialize the `RemoteSensing` application.
   - Run `SetRSParameters 123 1000 100 1`. This will set up the program to:
     - Acquire data from channels 1, 2, and 3 (the accelerometer in the *x*, *y*, and *z* directions respectively). You can acquire any combination of channels by inputting the desired combination instead of `123`. For example, to get data from only channels 1 and 3, input `SetRSParameters 13 1000 100 1`.
     - Requesting 1000 samples.
     - Input `1` for network time synchronization. You can initiate network without time synchronization by replacing the fourth input `1` with `0`. For example, `SetRSParameters 123 1000 100 0`.
   - Run `StartRemoteSensing 1` to actually start acquiring data.
     - Input `StartRemoteSensing 1` will erase the first memory block and store. Input `‘0’` will store data in the next available block.
     - Some debug output will be printed in the first Cygwin window.
     - Actual sensing will start after you see "`Starting data acquisition in 21 seconds…` " line in the second Cygwin window.
     - Sensing is done when you see “`Responsive nodes are <nodeId>`”line in the second Cygwin window.
   - Run `RetrieveData -1 <nodeId> [nodeId] [nodeId] …`
     - Input `-1` to retrieve the most recent data set.
     - Subsequent debug output may be delayed until all of the data is printed in the other Cygwin window. You may have to push `Enter` once in debug Cygwin window to get the `BluSH` prompt again.

You can verify that the application is running correctly, if you can retrieve data from all the nodes without any problem. (For detailed explanation of the parameter options and commands, refer to the Getting Started Advanced Users Guide)

Step2. The aim of this step is to determine a stable maximum distance between nodes.

I. If you are working in an indoor environment, detach the antennas from the nodes.
II. Set a node (leaf node) at a certain distance from the gateway node.
III. Run `TestRadio <count> <nodeId> [nodeId] …` from the gateway's Blush shell.
IV. Determine a distance at which at least 85% round trip reception rate is achieved using `TestRadio`. The distance should be the maximum distance that can still achieve 85% round trip reception rate (% should be more than 85). You can start with a small distance between the two nodes, and increase it until the reception rate falls below 85%.

Note. Multi-hop communication is very sensitive to the reception rate, too large a distance or too low radio power can highly degrade multi-hop performance.

Step3. The aim of this step is to determine a suitable set-up for multi-hop test.

I. Set the nodes in a line. The distance between every two nodes should be equal to the distance determined in Step2.
II. Run the application in Step1 once again. If successful go to III, otherwise decrease the distance and try again.
III. Run the `PrintRoutes` command. This is to verify that we have multiple hops in the routes. In the printed routes, if the hop is 1 (or equivalently if the "`next`" for each node is equal to itself), it means that all routes are single hop. If this is the case, increase the distances between nodes and goe to II, otherwise, the test was successful.

6. Troubleshooting

The following list gives some guidelines that will help improve the success of your Imote2 testing and deployment.

- If a node is exhibiting unexpected behavior or does not participate in routing, check the battery voltage. Generally, multi-hop communication is more sensitive to battery voltage compared to single-hop communication. This issue is easy to forget and can lead to frustration if overlooked. In this case run the following commands on the gateway's Blush shell:
  ```
  RemoteCommand Vbat [NodeId]
  ```
- If a node is repeatedly exhibiting unexpected behavior and the battery voltage is adequate, it could simply be a hardware problem. Imote2s, battery boards, sensor boards and antennas have finite lives and can simply be "bad". Try to isolate the source of the hardware problem by switching the battery board/sensor board/antenna.
- If a node is not participating in the routing or if inconsistent behavior is observed, it means that the node is not operating with the correct set of flash constants. In this case run the following commands on the gateway's Blush shell:

```
LocalCommand RestoreFC 2
RemoteCommand RestoreFC 2 <nodeId> [nodeId] …
```

- If after uploading the application image the mote continuously reboots (the LED flashes briefly about once every 2 seconds), it means that incorrect Flash constants are loaded on the mote. It is necessary to reprogram the mote with a fresh set of Flash constants. See the README file in *tools/WriteFlashConstants* for detailed instructions.
- If you experience consistent difficulty with communication (network wakeup times out, sending RemoteSensing channel parameters takes too long, etc) check the communication environment using TestRadio. This will reveal if a particular node has communication problems or if the test environment/network topology in general is not conducive to successful communication.
- If you plan to deploy your network in cold temperatures, be aware that battery performance degrades as the temperature decreases and this may cause problems with the operation of the Imote2s.

7. Conclusions

This guide has provided instructions on how to compile and program the motes with multi-hop communication capability. Fine-tuning network settings for efficient multi-hop communication and guidelines for testing multi-hop communication were also discussed. While the troubleshooting tips are not comprehensive, they address many commonly encountered difficulties.

Please check back often with the ISHMP website (http://shm.cs.uiuc.edu) for software and documentation updates.