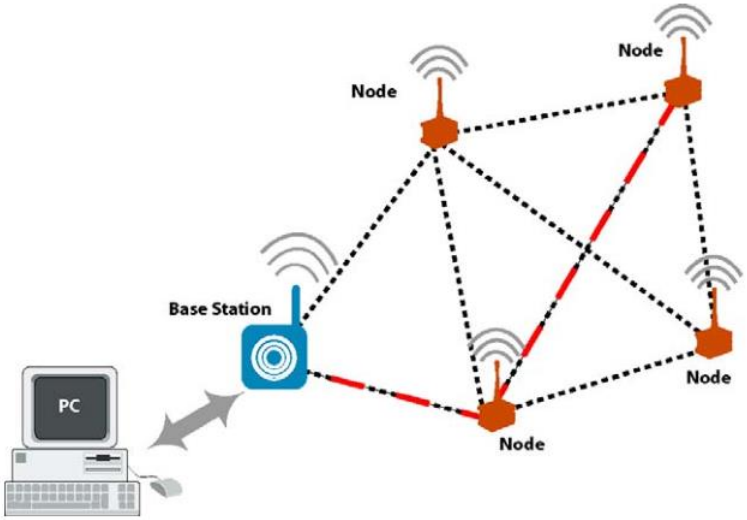
XMesh

Multi-hop mesh networking service

Contents

1 Introduction

XMesh is a full featured multi-hop, ad-hoc, mesh networking protocol developed by Memsic for wireless networks. An XMesh network consists of nodes (motes) that wirelessly communicate to each other and are capable of hopping radio messages to a base station where they are passed to a PC or other client. The hopping effectively extends radio communication range and reduces the power required to transmit message. Two nodes do not need to be within direct radio range of each other to communicate. A message can be delivered to one or more nodes in-between which will route the data. Likewise, if there is a bad radio link between two nodes, that obstacle can be overcome by rerouting around the area of bad service. Typically the nodes run in a low power mode, spending most of their time in a sleep state, in order to achieve multi-year battery life.



The entire XMesh network consists of

- Motes (nodes) participating in the network.
- A base station mote. This is a mote programmed with *XMeshBase* application. This mote manages the network and forwards data messages to and out of the mesh. The base station mote does not have to be permanently attached to a gateway.
- A PC which receives data and sends commands to the network.

XMesh can route data from nodes to a base station (*upstream*) or to individual nodes (*downstream*). It can also broadcast within a single area of coverage or arbitrarily between any two nodes in a cluster. QOS (Quality of Service) is provided by either a best effort (link level acknowledgement) and guaranteed delivery (end-to-end acknowledgement). Also, XMesh can be configured into various power modes including HP (high power), LP (low power), and ELP (extended low power).

More details can be found in XMesh user manual.

2 Example XMesh application

In this example you will learn:

- How to enhance the light sensing application with the XMesh multi-hop networking service
- Using XSniffer to display XMesh sensor data message on a PC

Hardware requirements:

- Three motes
- One sensor board (MDA100, MTS300 or MTS310)
- One gateway/programming board MIB520
- A PC with MoteWorks & MoteView installed

Exercise 1

To start, create a new folder named **Ex1_Light_Xmesh**. Copy there the files `Photo.nc`, `PhotoM.nc`, and `sensorboardApp.h` available in **/MoteWorks/apps/tutorials/lesson_4**, for example.

a) MakeMemslocal

The *MakeMemslocal* file contains global parameters which are applicable across all applications built for a particular installation. The file is located in **/MoteWorks/apps**.

***MakeMemslocal* Parameters**

Parameter	Description
RADIO_CLASS	This parameter defines the radio band in which the network communicates for MICA2/MICA2DOT radios. The operating band is defined by the mote's radio hardware. This should correspond to the label on the board. The available classes for Mica2 and Mica2Dot are 916 MHz, 433 MHz and 315 MHz.
RADIO_CHANNEL	This parameter defines the radio channel the network is operating on. Each band has multiple channels upon which it can operate. The user should choose a channel which is not being used by other wireless devices in the network (including other sensor networks). See table below for MICAz settings.
RADIO_POWER	This parameter defines the power level for the radio.
DEFAULT_LOCAL_GROUP	The local group is the group id upon which each node in your network will communicate on. The group id is a way for multiple networks to operate on the same radio band and channel yet filter communication by group id.

b) Makefile

The first step in creating an application is to type in the *Makefile*. To do it, enter the following text into a new document and save it as *Makefile*.

```
include Makefile.component
include $(TOSROOT)/apps/MakeMemslocal
GOALS += basic freq binlink,all
GOALS += group,125
include $(MAKERULES)
```

c) Makefile.component

The next step is to create the Makefile.component file. This file describes the top level application component, MyApp and the name of the sensor board we are going to use. To create it, enter the following text into a new document and save it as Makefile.component:

```
COMPONENT=MyApp
SENSORBOARD=mda100
INCLUDES += -I${TOSROOT}/tos/lib/RadioSync
```

d) Create the top-level configuration

The application's configuration is located in the `MyApp.nc` file. To implement XMesh, this configuration file needs to have two components that were not present in the previous examples: MULTIHOPROUTER and GenericCommPromiscuous. The MULTIHOPROUTER component is the XMesh multi-hop routing service. The GenericCommPromiscuous component is used by the MULTIHOPROUTER service to provide the basic radio communications functions; it's included here for initialization only.

You may remember that MyApp.nc file in **/Memsic/Cygwin/opt /MoteWorks/apps/tutorials/lesson_2** had a component named **GenericComm**. That component provides single hop, point-to-pint communication service. **GenericCommPromiscuous** is similar in functionality but adds special radio *snooping* capabilities, required by XMesh to multi-hop messages back to the base station.

To create the application's configuration, enter the following text into a new document and save it as `MyApp.nc`.

```
#include "appFeatures.h"
includes sensorboardApp;

configuration MyApp {
}
implementation {
  components Main, GenericCommPromiscuous as Comm, MULTIHOPROUTER, MyAppM, TimerC,
  LedsC, Photo;
```

```
  Main.StdControl -> TimerC.StdControl;
  Main.StdControl -> MyAppM.StdControl;
  Main.StdControl -> Comm.Control;
  Main.StdControl -> MULTIHOPROUTER.StdControl;

  MyAppM.Timer -> TimerC.Timer[unique("Timer")];
  MyAppM.Leds -> LedsC.Leds;
  MyAppM.PhotoControl -> Photo.PhotoStdControl;
  MyAppM.Light -> Photo.ExternalPhotoADC;

  MyAppM.RouteControl -> MULTIHOPROUTER;
  MyAppM.Send -> MULTIHOPROUTER.MhopSend[AM_XMULTIHOP_MSG];
  MULTIHOPROUTER.ReceiveMsg[AM_XMULTIHOP_MSG] -> Comm.ReceiveMsg[AM_XMULTIHOP_MSG];
}
```

Notice that a component named `MULTIHOPROUTER` is added. This is the actual component that implements XMesh. The module MyAppM now uses `MhopSend` instead of `SendMsg` interface. XMesh implements the `MhopSend` interface as follows:

```
interface MhopSend {
  command result_t send(uint16_t dest, uint8_t mode, TOS_MsgPtr msg,
                        uint16_t length);
  command void* getBuffer(TOS_MsgPtr msg, uint16_t* length);
  event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

e) Create the module

The application's module is located in the `MyAppM.nc` file. This module sends the sensor message over the Mote radio using the XMesh routing service. To create the application's module, enter the following text into a new document and save it as `MyAppM.nc`.

```
#include "appFeatures.h"
includes MultiHop;

module MyAppM {
  provides {
    interface StdControl;
  }
  uses {
    interface Timer;
    interface Leds;
    interface StdControl as PhotoControl;
    interface ADC as Light;
    interface MhopSend as Send;
    interface RouteControl;
  }
}
```

```
implementation {
  bool sending_packet = FALSE;
  TOS_Msg msg_buffer;
  XDataMsg *pack;
```

```
  command result_t StdControl.init() {
    uint16_t len;
    call Leds.init();
    call PhotoControl.init();

    // Initialize the message packet with default values
    atomic {
      pack = (XDataMsg*)call Send.getBuffer(&msg_buffer, &len);
      pack->board_id = SENSOR_BOARD_ID;
      pack->packet_id = 4;
    }

    return SUCCESS;
  }
```

```
  /**
   * Start things up.  This just sets the rate for the clock component.
   **/
  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000);
  }

  /**
   * Halt execution of the application. This just disables the clock component.
   **/
  command result_t StdControl.stop() {
    return call Timer.stop();
  }
```

```
  /**
   * Toggle the red LED in response to the Timer.fired event.
   * Start the Light sensor control and sample the data
   **/
  event result_t Timer.fired()
  {
    call Leds.redToggle();
    call PhotoControl.start();
    call Light.getData();
    return SUCCESS;
  }
```

```
  /**
    * Stop the Light sensor control, build the message packet and send
    **/
  void task SendData()
  {
    call PhotoControl.stop();

    if (sending_packet) return;
    atomic sending_packet = TRUE;

    // send message to XMesh multi-hop networking layer
    pack->parent = call RouteControl.getParent();
    if (call Send.send(BASE_STATION_ADDRESS,MODE_UPSTREAM,&msg_buffer,sizeof(XDataMsg))
      != SUCCESS)
      sending_packet = FALSE;

    return;
  }
```

Notice that the packet must include the current routing parent; this is obtained by making a call to the XMesh `RouteControl.getParent` command.

The message is sent using the `Send.send` command specifying the base station as the destination and the transport mode as `MODE_UPSTREAM`.

```
  /**
   * Light ADC data ready
   * Toggle yellow LED to signal Light sensor data sampled
   **/
  async event result_t Light.dataReady(uint16_t data) {
    atomic pack->light = data;
    atomic pack->vref = 417; // a dummy 3V reference voltage
    post SendData();
    call Leds.yellowToggle();

    return SUCCESS;
  }
```

```
  /**
    * Sensor data has been successfully sent through XMesh
    * Toggle green LED to signal message sent.
    **/
  event result_t Send.sendDone(TOS_MsgPtr msg, result_t success) {
    call Leds.greenToggle();
    atomic sending_packet = FALSE;

    return SUCCESS;
  }
}
```

Plug the mote that will function as the sensor node into the programming board. Compile and install `MyAPP` application into that sensor node. Set the note ID to 1. Once programmed, you should see the LEDs flashing every second.

f) Creating a base mote

Plug the mote that will function as the base station into the programming board. This Mote should be programmed with a special application named XMeshBase (v2.4) located in **/MoteWorks/apps/xmesh** folder. Set the node ID to 0.

Remove the base station mote from the programming board. Keep it switched off for the moment.

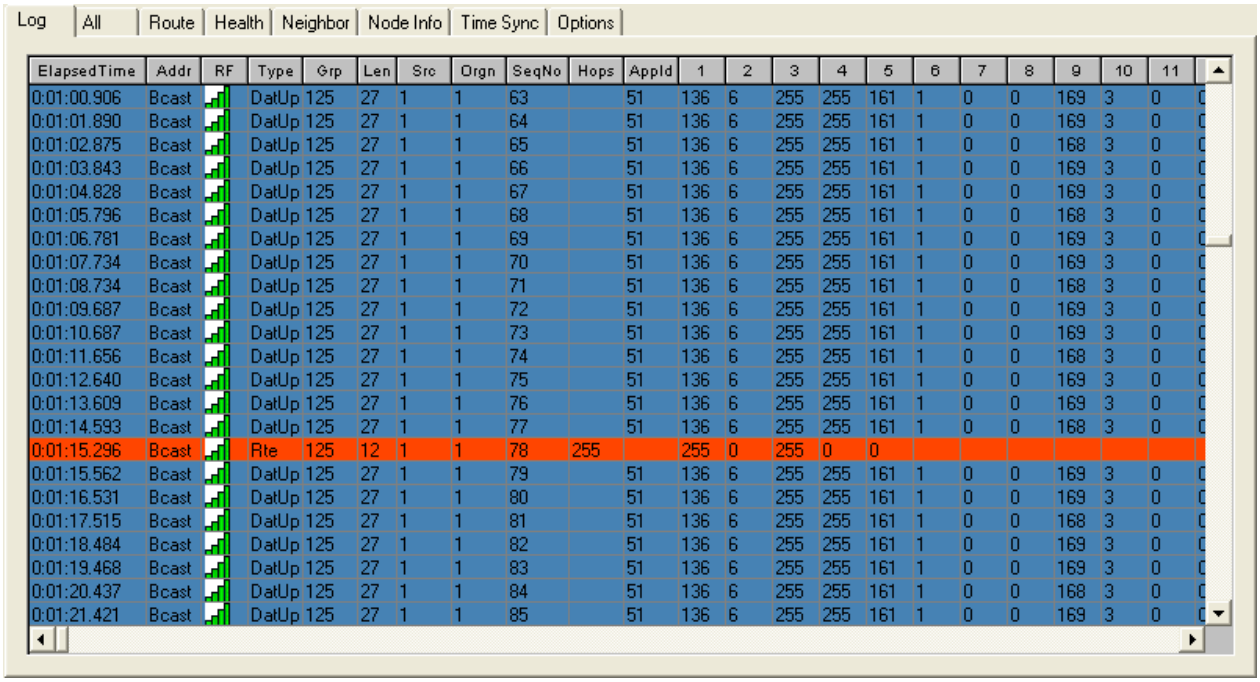
Note: The mote that functions as the base station is always programmed with the node ID of 0.

g) Using XSniffer to view sensor data sent through the network

We will now use the XSniffer tool to monitor the messages being sent from the sensor node. Install the XSniffer application onto one of motes, and leave it connected to the gateway/PC. This application is located in **/MoteWorks/apps/general/XSniffer** folder. Install this application with a node ID of 2.

Keep the mote you just programmed plugged into the programming board. Launch the XSniffer software application to start *sniffing* the radio traffic. Switch ON the sensor mote.

After a short time you should see message packets displayed in Xsniffer, similar to figure below.

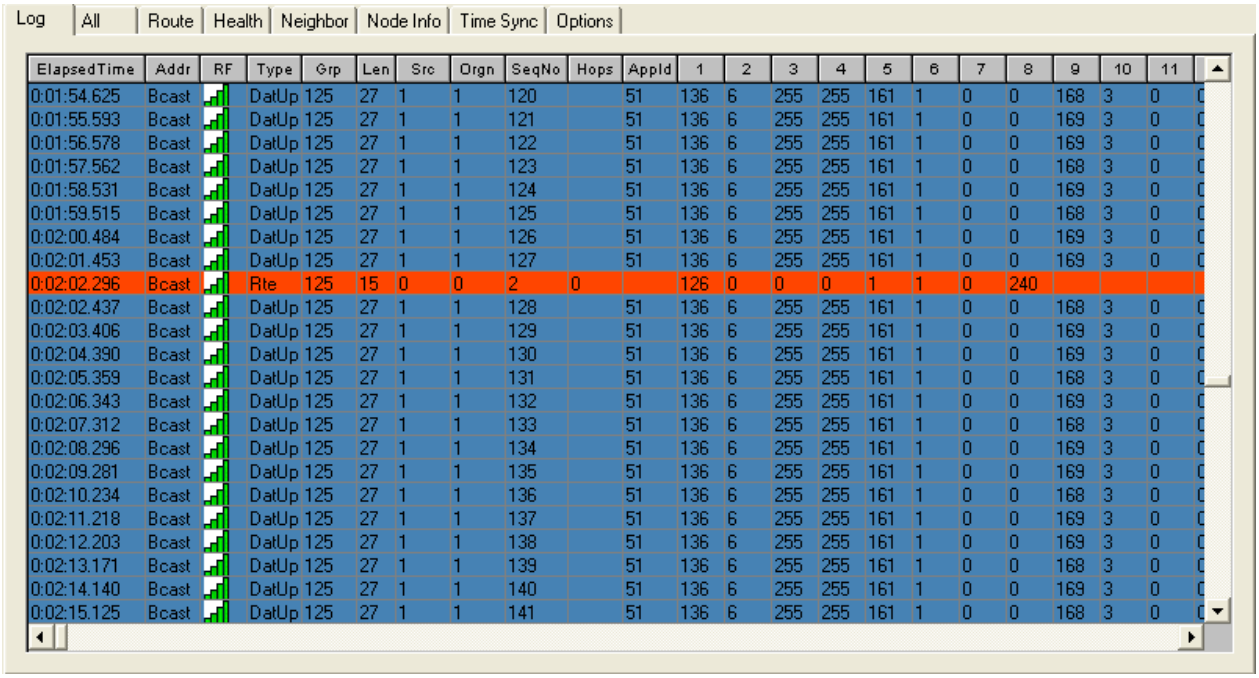


ElapsedTime	Addr	RF	Type	Grp	Len	Src	Orgn	SeqNo	Hops	AppId	1	2	3	4	5	6	7	8	9	10	11
0:01:00.906	Bcast		DatUp	125	27	1	1	63		51	136	6	255	255	161	1	0	0	169	3	0
0:01:01.890	Bcast		DatUp	125	27	1	1	64		51	136	6	255	255	161	1	0	0	169	3	0
0:01:02.875	Bcast		DatUp	125	27	1	1	65		51	136	6	255	255	161	1	0	0	168	3	0
0:01:03.843	Bcast		DatUp	125	27	1	1	66		51	136	6	255	255	161	1	0	0	169	3	0
0:01:04.828	Bcast		DatUp	125	27	1	1	67		51	136	6	255	255	161	1	0	0	169	3	0
0:01:05.796	Bcast		DatUp	125	27	1	1	68		51	136	6	255	255	161	1	0	0	168	3	0
0:01:06.781	Bcast		DatUp	125	27	1	1	69		51	136	6	255	255	161	1	0	0	169	3	0
0:01:07.734	Bcast		DatUp	125	27	1	1	70		51	136	6	255	255	161	1	0	0	169	3	0
0:01:08.734	Bcast		DatUp	125	27	1	1	71		51	136	6	255	255	161	1	0	0	168	3	0
0:01:09.687	Bcast		DatUp	125	27	1	1	72		51	136	6	255	255	161	1	0	0	169	3	0
0:01:10.687	Bcast		DatUp	125	27	1	1	73		51	136	6	255	255	161	1	0	0	169	3	0
0:01:11.656	Bcast		DatUp	125	27	1	1	74		51	136	6	255	255	161	1	0	0	168	3	0
0:01:12.640	Bcast		DatUp	125	27	1	1	75		51	136	6	255	255	161	1	0	0	169	3	0
0:01:13.609	Bcast		DatUp	125	27	1	1	76		51	136	6	255	255	161	1	0	0	169	3	0
0:01:14.593	Bcast		DatUp	125	27	1	1	77		51	136	6	255	255	161	1	0	0	168	3	0
0:01:15.296	Bcast		Rte	125	12	1	1	78	255		255	0	255	0	0						
0:01:15.562	Bcast		DatUp	125	27	1	1	79		51	136	6	255	255	161	1	0	0	169	3	0
0:01:16.531	Bcast		DatUp	125	27	1	1	80		51	136	6	255	255	161	1	0	0	169	3	0
0:01:17.515	Bcast		DatUp	125	27	1	1	81		51	136	6	255	255	161	1	0	0	168	3	0
0:01:18.484	Bcast		DatUp	125	27	1	1	82		51	136	6	255	255	161	1	0	0	169	3	0
0:01:19.468	Bcast		DatUp	125	27	1	1	83		51	136	6	255	255	161	1	0	0	169	3	0
0:01:20.437	Bcast		DatUp	125	27	1	1	84		51	136	6	255	255	161	1	0	0	168	3	0
0:01:21.421	Bcast		DatUp	125	27	1	1	85		51	136	6	255	255	161	1	0	0	169	3	0

You can see from the elapsed time that the messages are being sent about one second apart – each time the LEDs blink, a new message should be captured by XSniffer.

There are a couple of interesting things to note. First, look at the destination address field *Addr*. The value for this field is **Bcast**, which means the sensor node ID 1 (*Src* column) is broadcasting its packet to all nodes. This is the initial state of XMesh until the multi-hop network has formed efficient routes. You can see there are two types of messages being sent by the sensor node (*Type* column). Message type *DatUp* identifies a message as a data message sent upstream from the sensor node to the base station. Message type *Rte* designates a route update message. Route update messages are periodically sent by all nodes in a mesh network for the purpose of updating each other's routing tables.

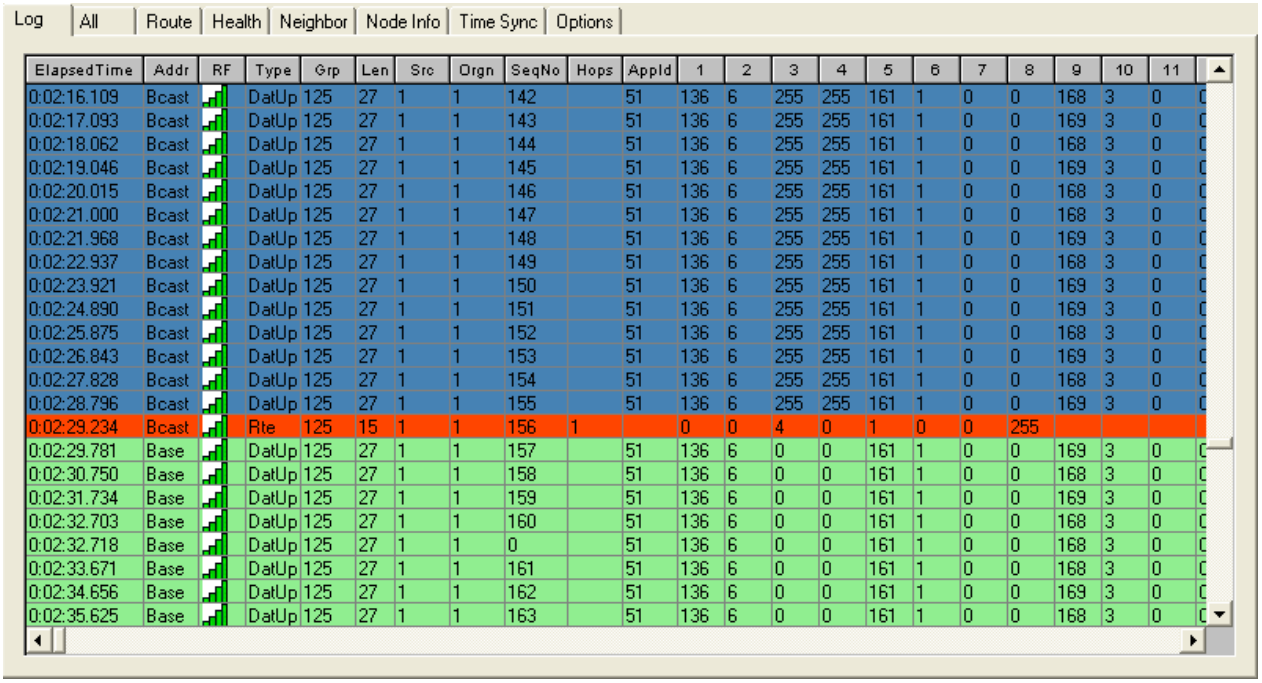
So far, we see the messages from one sensor node. Switch on the base mote. With a base mote, XMesh can now create a multi-hop network. As the messages flow into XSniffer, you should begin to see something interesting:


Log | All | Route | Health | Neighbor | Node Info | Time Sync | Options

ElapsedTime	Addr	RF	Type	Grp	Len	Src	Orgn	SeqNo	Hops	AppId	1	2	3	4	5	6	7	8	9	10	11
0:01:54.625	Bcast		DatUp	125	27	1	1	120		51	136	6	255	255	161	1	0	0	168	3	0
0:01:55.593	Bcast		DatUp	125	27	1	1	121		51	136	6	255	255	161	1	0	0	169	3	0
0:01:56.578	Bcast		DatUp	125	27	1	1	122		51	136	6	255	255	161	1	0	0	169	3	0
0:01:57.562	Bcast		DatUp	125	27	1	1	123		51	136	6	255	255	161	1	0	0	168	3	0
0:01:58.531	Bcast		DatUp	125	27	1	1	124		51	136	6	255	255	161	1	0	0	169	3	0
0:01:59.515	Bcast		DatUp	125	27	1	1	125		51	136	6	255	255	161	1	0	0	168	3	0
0:02:00.484	Bcast		DatUp	125	27	1	1	126		51	136	6	255	255	161	1	0	0	169	3	0
0:02:01.453	Bcast		DatUp	125	27	1	1	127		51	136	6	255	255	161	1	0	0	169	3	0
0:02:02.296	Bcast		Rte	125	15	0	0	2	0		126	0	0	0	1	1	0	240			
0:02:02.437	Bcast		DatUp	125	27	1	1	128		51	136	6	255	255	161	1	0	0	168	3	0
0:02:03.406	Bcast		DatUp	125	27	1	1	129		51	136	6	255	255	161	1	0	0	169	3	0
0:02:04.390	Bcast		DatUp	125	27	1	1	130		51	136	6	255	255	161	1	0	0	169	3	0
0:02:05.359	Bcast		DatUp	125	27	1	1	131		51	136	6	255	255	161	1	0	0	168	3	0
0:02:06.343	Bcast		DatUp	125	27	1	1	132		51	136	6	255	255	161	1	0	0	169	3	0
0:02:07.312	Bcast		DatUp	125	27	1	1	133		51	136	6	255	255	161	1	0	0	168	3	0
0:02:08.296	Bcast		DatUp	125	27	1	1	134		51	136	6	255	255	161	1	0	0	169	3	0
0:02:09.281	Bcast		DatUp	125	27	1	1	135		51	136	6	255	255	161	1	0	0	169	3	0
0:02:10.234	Bcast		DatUp	125	27	1	1	136		51	136	6	255	255	161	1	0	0	168	3	0
0:02:11.218	Bcast		DatUp	125	27	1	1	137		51	136	6	255	255	161	1	0	0	169	3	0
0:02:12.203	Bcast		DatUp	125	27	1	1	138		51	136	6	255	255	161	1	0	0	169	3	0
0:02:13.171	Bcast		DatUp	125	27	1	1	139		51	136	6	255	255	161	1	0	0	168	3	0
0:02:14.140	Bcast		DatUp	125	27	1	1	140		51	136	6	255	255	161	1	0	0	169	3	0
0:02:15.125	Bcast		DatUp	125	27	1	1	141		51	136	6	255	255	161	1	0	0	168	3	0

After switching on the base mote, you will start to see *Rte* messages (route update) being generated for node 0 – the base station. The sensor mote will see these messages and will eventually add the base station as its parent. Once this happens you will see the *DatUp* messages from mote 1 being sent directly to the base station node 0 instead of being broadcast. The base station id 0 is denoted **Base** in *Addr* field.

You have just witnessed a two node multi-hop mesh network!



Log | All | Route | Health | Neighbor | Node Info | Time Sync | Options

ElapsedTime	Addr	RF	Type	Grp	Len	Src	Orgn	SeqNo	Hops	AppId	1	2	3	4	5	6	7	8	9	10	11
0:02:16.109	Bcast		DatUp	125	27	1	1	142		51	136	6	255	255	161	1	0	0	168	3	0
0:02:17.093	Bcast		DatUp	125	27	1	1	143		51	136	6	255	255	161	1	0	0	169	3	0
0:02:18.062	Bcast		DatUp	125	27	1	1	144		51	136	6	255	255	161	1	0	0	168	3	0
0:02:19.046	Bcast		DatUp	125	27	1	1	145		51	136	6	255	255	161	1	0	0	169	3	0
0:02:20.015	Bcast		DatUp	125	27	1	1	146		51	136	6	255	255	161	1	0	0	168	3	0
0:02:21.000	Bcast		DatUp	125	27	1	1	147		51	136	6	255	255	161	1	0	0	168	3	0
0:02:21.968	Bcast		DatUp	125	27	1	1	148		51	136	6	255	255	161	1	0	0	169	3	0
0:02:22.937	Bcast		DatUp	125	27	1	1	149		51	136	6	255	255	161	1	0	0	168	3	0
0:02:23.921	Bcast		DatUp	125	27	1	1	150		51	136	6	255	255	161	1	0	0	169	3	0
0:02:24.890	Bcast		DatUp	125	27	1	1	151		51	136	6	255	255	161	1	0	0	169	3	0
0:02:25.875	Bcast		DatUp	125	27	1	1	152		51	136	6	255	255	161	1	0	0	168	3	0
0:02:26.843	Bcast		DatUp	125	27	1	1	153		51	136	6	255	255	161	1	0	0	169	3	0
0:02:27.828	Bcast		DatUp	125	27	1	1	154		51	136	6	255	255	161	1	0	0	168	3	0
0:02:28.796	Bcast		DatUp	125	27	1	1	155		51	136	6	255	255	161	1	0	0	169	3	0
0:02:29.234	Bcast		Rte	125	15	1	1	156	1		0	0	4	0	1	0	0	255			
0:02:29.781	Base		DatUp	125	27	1	1	157		51	136	6	0	0	161	1	0	0	169	3	0
0:02:30.750	Base		DatUp	125	27	1	1	158		51	136	6	0	0	161	1	0	0	168	3	0
0:02:31.734	Base		DatUp	125	27	1	1	159		51	136	6	0	0	161	1	0	0	169	3	0
0:02:32.703	Base		DatUp	125	27	1	1	160		51	136	6	0	0	161	1	0	0	168	3	0
0:02:32.718	Base		DatUp	125	27	1	1	0		51	136	6	0	0	161	1	0	0	168	3	0
0:02:33.671	Base		DatUp	125	27	1	1	161		51	136	6	0	0	161	1	0	0	168	3	0
0:02:34.656	Base		DatUp	125	27	1	1	162		51	136	6	0	0	161	1	0	0	169	3	0
0:02:35.625	Base		DatUp	125	27	1	1	163		51	136	6	0	0	161	1	0	0	168	3	0

Exercise 2

Create a new folder called **Ex1_Light_Xmesh_ACK** and copy there the files located in **Ex1_Light_Xmesh** folder. Then, do the following modifications in the code:

1) Set different group ID for each network (for example 9, 77, or 100).
2) Set the period between two sensor readings to `T_sample` = 4 s.
3) Modify the code to use the `MOTE_UPSTREAM_ACK` transport mode to request an acknowledgment back from the base station when the mote message is received.
4) Make the yellow LED blink (switch 500ms ON and then OFF) only when the acknowledgment message is received.
5) Introduce the maximum waiting time for the acknowledgment message `T_ack` = 1 s. If the acknowledgment message is not received within `T_ack` time, resend the last message with the sensor readings. After two resend attempts, stop resending and continue sampling light sensor values.