Time Synchronization Using RBS Protocol

KFUPM – COE Department – Wireless Sensor Networks Project

Prepared by:

Faisal Arafsha (232083) and Redha Al-Hashem (235219)

Prepared For:

Dr. Tarek Sheltami

ABSTRACT

This report describes the project done in the Wireless Sensor Networks course (COE499) offered by the Computer Engineering Department of King Fahd University of Petroleum and Minerals (KFUPM) in semester 082. The main topic of our group's project is regarding Time Synchronization in wireless sensor networks. Implementation of the whole project was done using TinyOS as the operating system for TelosB sensor motes. The project is mainly the implementation of the Reference Based Synchronization (RBS) protocol. After the RBS protocol was implemented, it was tested on 2 motes and a base station mote to ensure correct operation.

1. INTRODUCTION

Time synchronization is an essential requirement in wireless sensor networks. Some operations in a sensor network depend strictly on synchronized time amongst different sensor nodes in the network. There are different reasons why time synchronization is an important issue in wireless sensor networks, these include:

- Time stamping measurements and readings
- In-network signal processing
- Localization using *Time Difference of Arrival* protocol (TDoA)
- Co-operative communication

There are different known fine-grained protocolsfor time synchronization in a wireless sensor network including RBS (Reference Based Synchronization), TPSN (Time Synchronization Protocol for Sensor Networks), Linear Parameter-Based Synchronization, and FTSP (Flooding Time Synchronization Protocol).

For our project, we decided to implement the RBS time synchronization protocol.

2. PROBLEM DESCRIPTION

RBS (Reference Based Synchronization) protocol is a fine-grained sensor network protocol. The main concept under RBS is as follows:

> Every node (other than the base station) has its own time. The base station broadcasts a beacon that includes the reference time. Once other nodes receive the beacon from base station, each node renews its time to the reference time, and then broadcasts its local time to other nodes. Once every node receives all external time(s), it takes the average of all of them and sets its local time to the average time. This makes time synchronized amongst all nodes.

This protocol is to be implemented in the TinyOS environment to be programmed in TelosB motes.

3. LITERATURE SURVEY

Fine-grained time synchronization protocols for wireless sensor networks have been extensively studied during the last decade. Some of the importantprotocols include the following:

- RBS: explained in the previous section.
- TPSN: Time Synchronization Protocol for Sensor Networks.
- Linear parameter based synchronization
- FTSP: Flooding Time Synchronization Protocol

Other protocols (coarse-grained protocols) for time synchronization in wireless sensor networks include the following:

- Lamport's Algorithm
- Cristian's Algorithm

RBS was chosen for the implementation of this project due to its simplicity and its high accuracy in relatively small area wireless sensor networks.

4. PROJECT IMPLEMENTATION

There will mainly be two types of sensors:

- SND: the base station that sends the beacon. This node is supposed to broadcast a beacon every time the user button on the TelosB mote in pressed.

- RCV: operational motes that are to be synchronized. Once a beacon is received, immediately take the average time, set it as new local time, and broadcast it. Once all external times received, take their average and update local time again.

The two types of sensors will be programmed with TinyOS 2.1 to implement the RBS time synchronization protocol.

The project implementation was divided such that the SND (base-station) mote programming is done by Redha Al-Hashem, and the RCV (operational) motes programming is done by Faisal Arafsha.

Operation Example:

The following figures show two example scenarios, one of a sensor network with 2 motes and a base station with time in between the two motes, and another example with a sensor network with base station time lower than both motes. The two nodes are located at different distances from the base station which means that the node which is closer to the base station will receive the beacon first. Each mote in the network has its own time (not synchronized). When the base station broadcasts the beacon, each node sets its local time with the received base station time, and then broadcasts its local time and receives other external times. Received times are then summed and the average amongst all of them is taken. The average time will become the local time for all nodes.

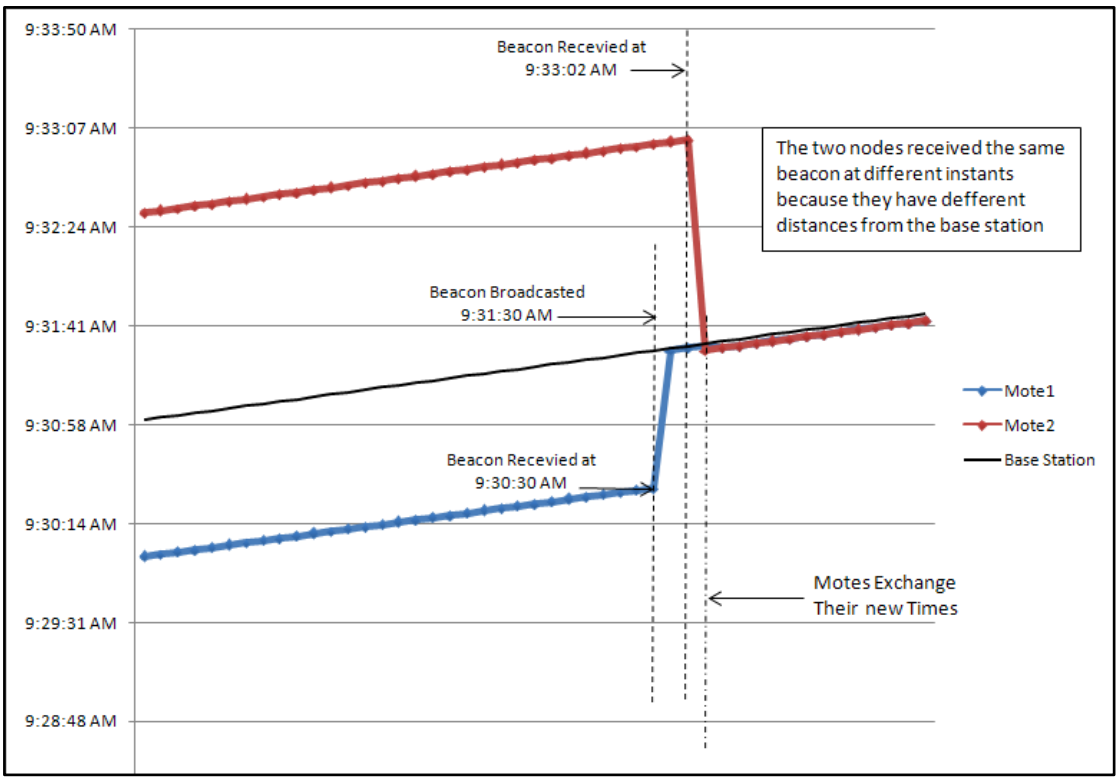


Illustration: two unsynchronized motes and their synchronization point

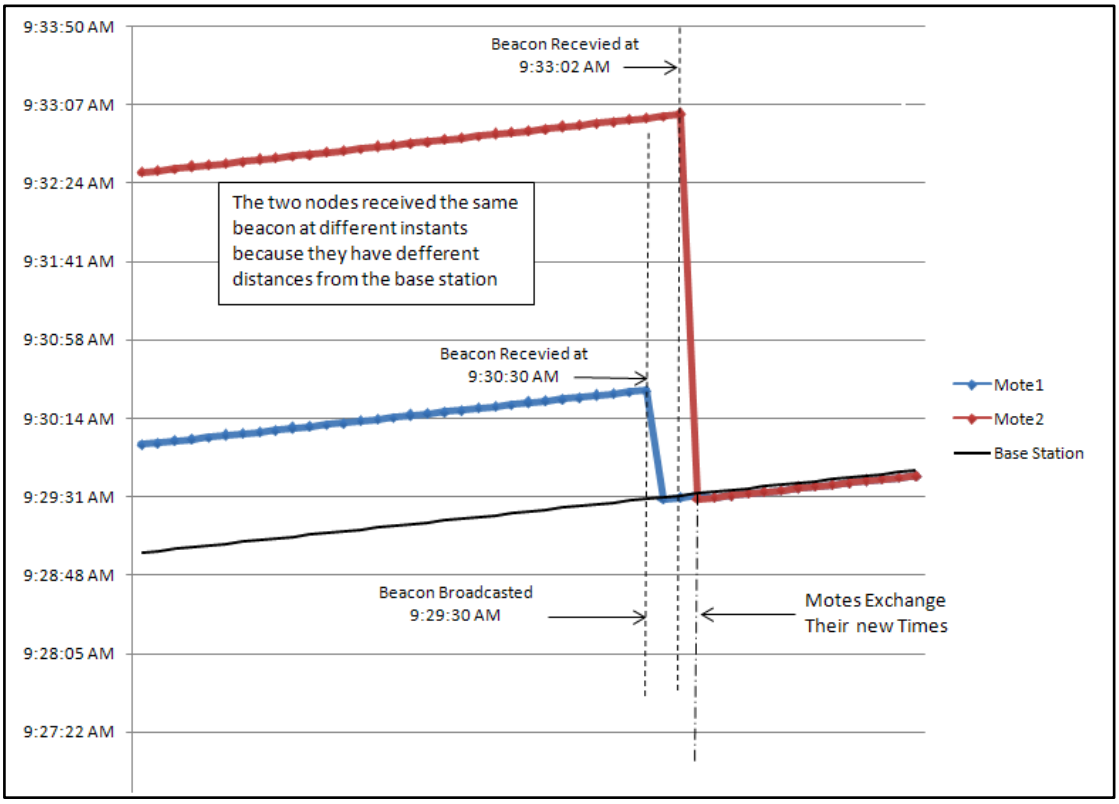
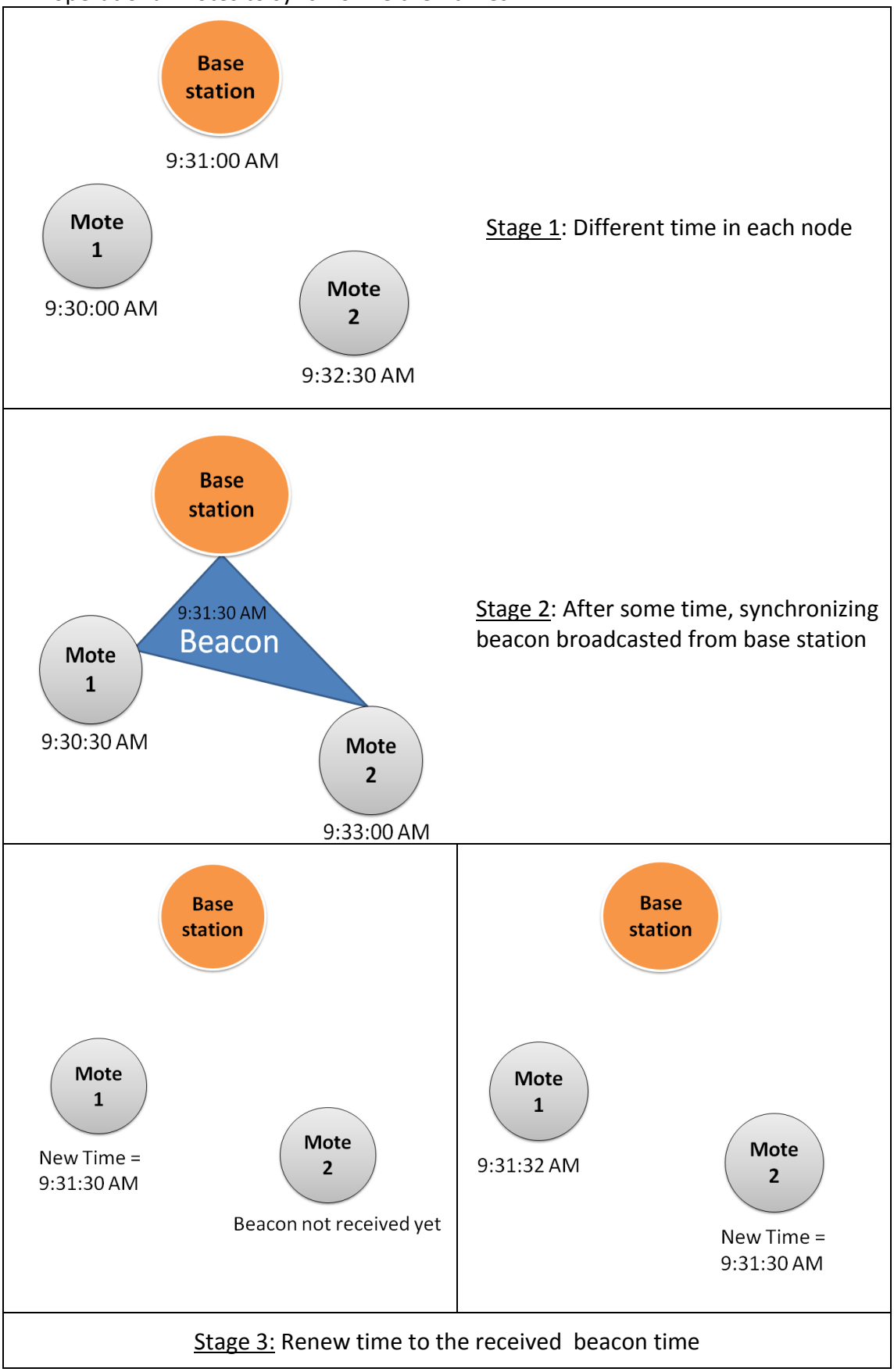


Illustration: two unsynchronized motes and their synchronization point

The following illustration shows the steps taken by the base station and operational motes to synchronize their times:



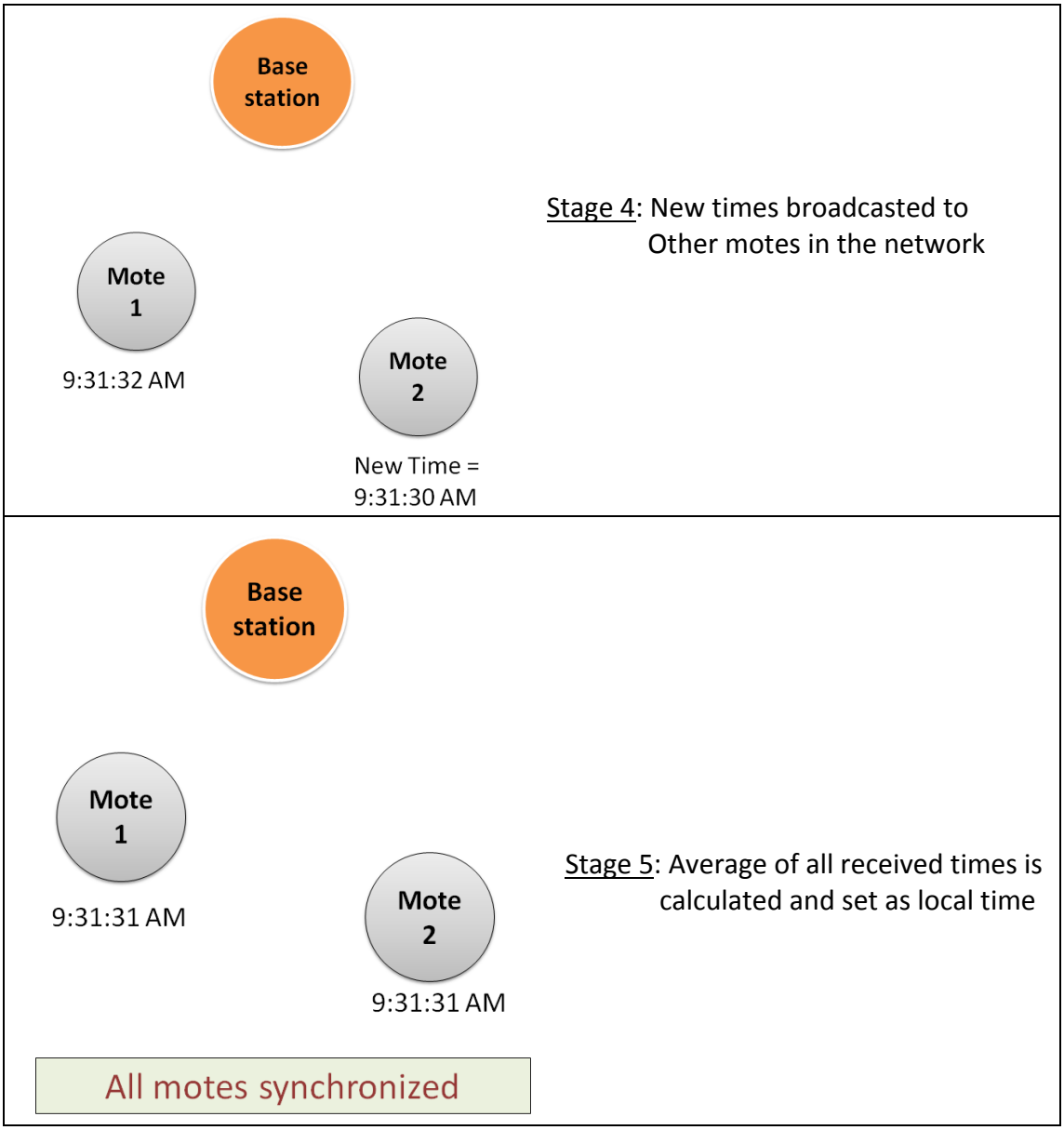


Illustration: time synchronization using RBS protocol

Code Description:

Four main files were used for the implementation of the protocol on TelosB motes. *SNDC.nc* and *SNDAppC.nc*files for the *SND* module, and *RCVC.nc* and *RCVAppC.nc* for the *RCV* modules. The following are pseudo-codes for the important files: *SNDC.nc* and *RCVC.nc*. Actual codes can be found in the appendix of this report.

SNDC.nc

- Create beacon as unsigned integer with value -1 (FF…FF)
// No matter what the size of the integer is in the sending or receiving end,
// all F's will be considered as a beacon.
- Keep watching the *User Button*.
  If pressed:
  - Broadcast the beacon with base time
  - Turn on blue LED

  If released:
    Turn off blue LED
  // Blue LED s used as an indicator that the button is pushed correctly
- After sending:
  If sending successful:
    Turn on green LED
  If sending is not successful:
    Turn on red LED
  Start a timer for 1 sec
- Timer fired:
    Close green and red LEDs

RCVC.nc

- Create an unsigned integer of size 32 bits for counting the seconds (time)
- Create a task for printing the time:
    // Time is received in seconds, needs conversion to HR:MN:SS
    hrs = timeInSec/3600 // decimals ignored
    min = (timeInSec/6) – (hrs*60)
    sec = timeInSec – ((hrs*3600)+(min*60))
    print "Time = hrs:min:sec"
- When mote starts:
    Start radio transceiver
    Start incrementing the counter every sec
      If counter = 86400 (24 hours)
        reset the counter
- When a message is received:
    If msg received has most significant byte == FF
      It's a beacon ➔ renew counter then broadcast new counter
    Else
    It's an external timer ➔ wait for all timers then take the average
- After sending:

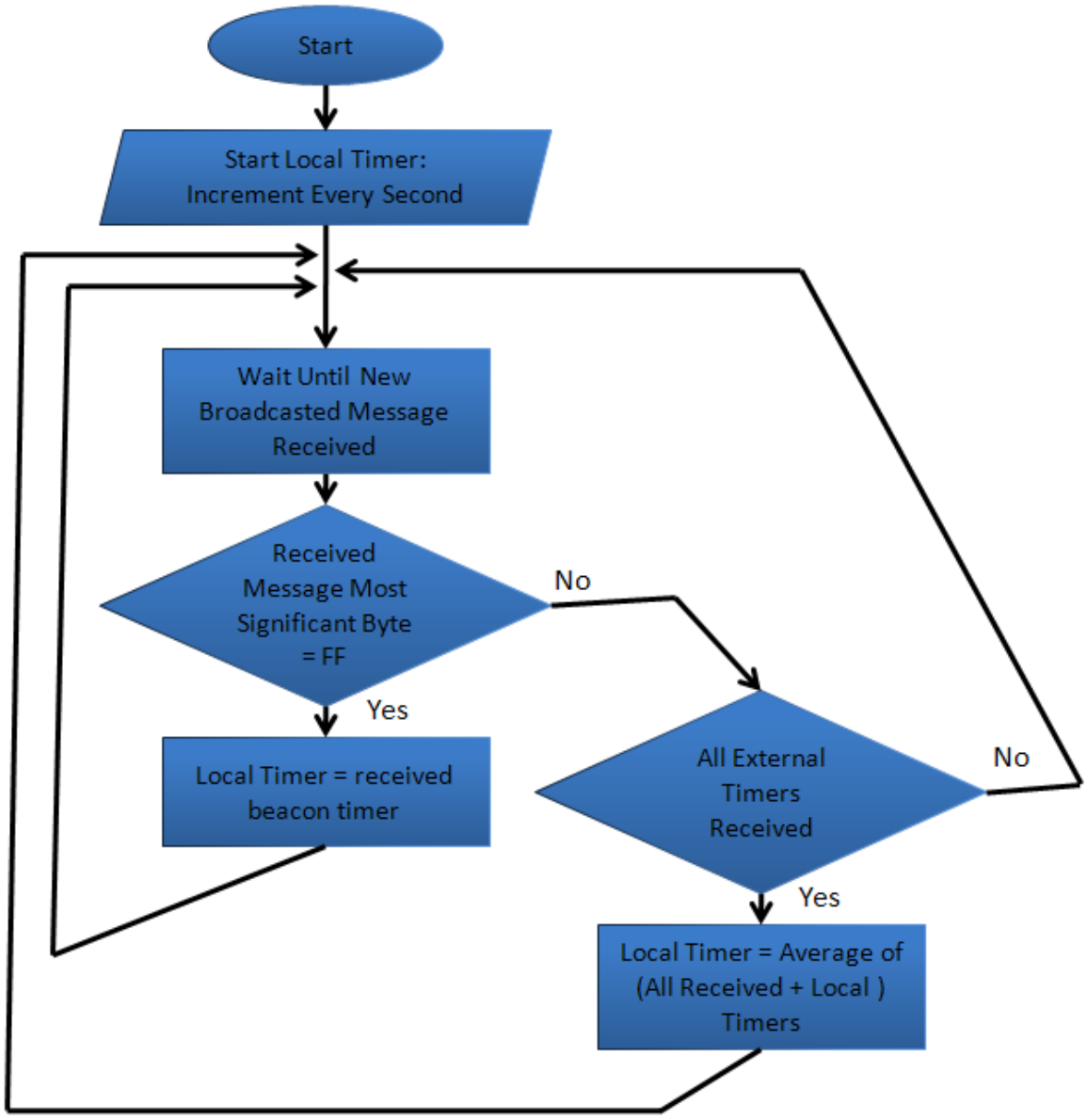
If sending (broadcasting local timer) successful:

Turn on green LED for 1 sec

Else

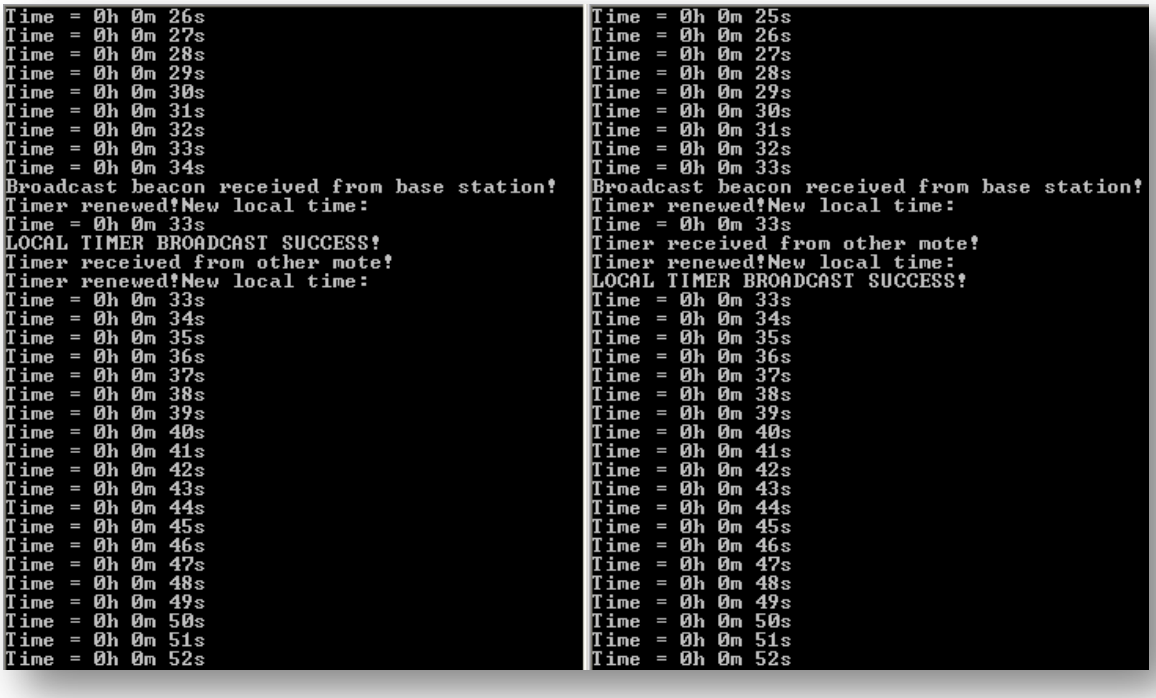
Turn on red LED for 1 sec

Flow Chart of RCV Modules:



5. RESULTS

The following illustration shows a snapshot of 2 operational, un-synchronized motes' outputs (as an example of operation) and the moment of their time synchronization.



MOTE 1 **MOTE 2**

The above illustration shows the reaction of both motes when a beacon is received. The beacon was received at time (0:0:34) for MOTE 1 and at (0:0:33) for MOTE 2. After that, both timers were renewed (by taking the time of the received beacon timer). Each mote, then, broadcasts its renewed timer. Once an external timer is received, the average between the local time and the received time is taken again and set as the local time. Afterwards, both timers are synchronized.

6. CONCLUSION

Time synchronization is very essential in wireless sensor networks. This project's main goal was to implement one of the fine-grained time synchronization protocols: the Reference Based Synchronization (RBS). Implementation of this protocol was done in the TinyOS environment as the operating system programmed in TelosB sensor motes. After the project was done and the motes were programmed, it was tested and results showed successful operation.

APPENDIX:

Codes used:

SNDC.nc

```
#include "printf.h"
#include <UserButton.h>
module SNDC{
    uses interface Boot;
    uses interface SplitControl as radioControl;
    uses interface AMSend;
    uses interface Leds;
    uses interface Packet;
    uses interface Notify<button_state_t>;
    uses interface Timer<TMilli> as timer;
    uses interface Timer<TMilli> as timer2;
    }
implementation{
    error_t ee;
    uint32_t beacon = 0xFF000000;
    message_t msg;
    uint32_t timeInSec;
    uint32_t hrs;
    uint32_t min;
    uint32_t sec;
    task void printTime();

    event void Boot.booted(){
        call radioControl.start();  //Starting the radio controller
        call Notify.enable();
        call timer.startPeriodic(1000);
    }
    event void radioControl.startDone(error_t e){
                //Do Nothing
    }
    event void radioControl.stopDone (error_t e){
                //Do nothing
    }
    event void Notify.notify( button_state_t state ) {
        if ( state == BUTTON_PRESSED ) {
                //get a 3-byte payload pointer in msg
                nx_uint32_t * myData = call AMSend.getPayload(&msg,4);
                * myData = beacon;     //edit the payload of "msg"
                call AMSend.send(AM_BROADCAST_ADDR, &msg, sizeof(*myData)); //send "msg"

                call Leds.led2On();
                }
        else if ( state == BUTTON_RELEASED ) {
                        call Leds.led2Off();
                }
        }
    event void AMSend.sendDone(message_t * msg, error_t e){
                if (e == SUCCESS)
                        call Leds.led1On();
                else
                        call Leds.led0On();
                        call timer2.startPeriodic(1000);      // count 1 sec then turn off LED
        }
    event void timer.fired(){
                printf("Beacon = %lx\n",beacon);
                timeInSec = beacon-0xFF000000;
                post printTime();
                printf("Beacon ");
                printfflush();
                if(beacon>=0xFF015180)// 0x15180 = 86400 sec/day
                        beacon=0xFF000000;//reset
                else
                        beacon++;
        }
    event void timer2.fired(){
                call Leds.led0Off();
                call Leds.led1Off();
        }
        task void printTime(){
                hrs = timeInSec/3600;
                min = (timeInSec/60) - (hrs*60);
                sec = timeInSec - ((hrs*3600) +(min*60));
                printf("Time = %dh ",hrs);
                printf("%dm ",min);
                printf("%ds\n",sec);
        }
    }
```

SNDAppC.nc

```
configuration SNDAppC{
}
implementation {
    components MainC, LedsC, SNDC;
    components ActiveMessageC;
    components new AMSenderC(20);
    components UserButtonC;
    components new TimerMilliC() as Timer;
    SNDC.Leds      ->      LedsC;
    SNDC.Boot      ->      MainC;
    SNDC.AMSend    ->      AMSenderC;
    SNDC.radioControl-> ActiveMessageC;
    SNDC.Packet    ->      AMSenderC;
    SNDC.Notify -> UserButtonC;
    SNDC.timer     ->      Timer;
    SNDC.timer2    ->      Timer;
}
```

RCVC.nc

```
#include "printf.h"
module RCVC{
    uses interface SplitControl as radio;
    uses interface Receive;
    uses interface AMSend;
    uses interface Timer<TMilli> as timer;// for count incrementing every sec
    uses interface Timer<TMilli> as timer2;// for LEDs off only
    uses interface Leds;
    uses interface Boot;
}
implementation{
    uint32_t count=0; //86400 sec/day
    uint32_t timeInSec;
    uint32_t hrs;
    uint32_t min;
    uint32_t sec;
    uint32_t received;
    nx_uint32_t* receivedP;

    task void printTime();

    event void Boot.booted(){
            call radio.start();    //Starting the radio controller
            call timer.startPeriodic(1000);       // to increment count
    }
    event void timer.fired(){
            if(count>=86400) // seconds per day (24x60x60)
                    count = 0;
            else
                    count++;
            timeInSec = count;
            post printTime();
            printfflush();
    }
    event void timer2.fired(){
            call Leds.led0Off();
            call Leds.led1Off();
    }

    event message_t* Receive.receive(message_t* msg, void* payload, uint8_t len){
            message_t* dummyMsg;
            message_t* myMsg;      //new message to send
            nx_uint32_t * myTimer;
            receivedP = (nx_uint32_t*)payload;
            received = (uint32_t) *receivedP;
            if(received >= 0xFF000000)
            //message received is a broadcasted beacon ==> broadcast my timer
            {
            printf("Broadcast beacon received from base station!\n");
            received = received - 0xFF000000;
            count = received;
            printf("Timer renewed!New local time:\n");
            timeInSec = count;
            post printTime();
            printfflush();
```

```
            //create new msg and get a 4-byte payload pointer in it
            myTimer = call AMSend.getPayload(myMsg,4);

            * myTimer = count;      //edit the payload of "newMsg"

            call AMSend.send(AM_BROADCAST_ADDR, myMsg, sizeof(*myTimer));  //broadcast "myMsg"

        }

        else
        // message received is "timer" from another node ==> average all received times
            {
                    printf("Timer received from other mote!\n");
                    printfflush();
                    count = (count+(received))/2;
                    printf("Timer renewed!New local time:\n");
                    timeInSec = count;
                    post printTime();
                    printfflush();

            }
            return dummyMsg;
    }
    task void printTime(){
            hrs = timeInSec/3600;
            min = (timeInSec/60) - (hrs*60);
            sec = timeInSec - ((hrs*3600) +(min*60));

            printf("Time = %dh ",hrs);
            printf("%dm ",min);
            printf("%ds\n",sec);
    }
    event void AMSend.sendDone(message_t * msg, error_t e){

            if (e == SUCCESS){
                    call Leds.led1On();
                    printf("LOCAL TIMER BROADCAST SUCCESS!\n");
            }
            else{
                    call Leds.led0On();
                    printf("LOCAL TIMER BROADCAST NOT SUCCESS!\n");
            }
            printfflush();
            call timer2.startPeriodic(1000);
    }
    event void radio.startDone(error_t e){

    //Do Nothing
    }

    event void radio.stopDone (error_t e){
    //Do nothing
    }
}
```

RCVAppC.nc

```
configuration RCVAppC{
}
implementation{
    components MainC;
    components new AMReceiverC(20);
    components new AMSenderC(20);
    components new TimerMilliC() as Timer;
    components LedsC;
    components RCVC;
    components ActiveMessageC;
    RCVC.Leds      ->      LedsC;
    RCVC.Boot      ->      MainC;
    RCVC.Receive-> AMReceiverC;
    RCVC.AMSend    ->      AMSenderC;
    RCVC.timer     ->      Timer;
    RCVC.timer2    ->      Timer;
    RCVC.radio     ->      ActiveMessageC;
}
```