UNC CHARLOTTE

Wireless Sensor Networks – Fall 2020

Final Project- Project #1

Distance Estimation Along a Line

ECGR 6189

Sumukh Raghuram Bhat
801131997

Table of Contents:

Project Objective:

The main objective of this project is to track the location of a target mote along a straight line between two anchor motes placed 100m apart, using RSSI measurements which is an indication of the power level being received by the receiving base station. Initially the path loss characteristics of the experiment set up place is found and later the distance has to be estimated based on the beacon mote 1 first and then with both beacon motes and finally to compare readings among these results so as to obtain an optimum distance of the unknown mobile mote.

Depiction:

|< --------------------------------100m --- >|

Beacon Node1 --------------------- Target Mote A ------------------------------ Beacon Node2

------------------- >   x dist known

by measuring signal strength

Requirements of the Project:

1. To Estimate the distance x using only one beacon mote first (discarding the second beacon mote signals)

2. To Determine a method to obtain the optimum x value by using both beacons.

3. To Compare the results from 1 and 2.

Stages of Execution:

1st week :

- Decided the place where the experiment should be done.
- Measured the channel characteristics of that place.
- Wrote a function to convert to distances based on RSSI.
- Understood the demo code of RSSI in the apps/tutorial of tinyos main.

2nd week:

- Started writing the program of localization using RSSI from two beacons.
- Programmed Beacon Node 1 and Beacon Node 2 to transmit periodic messages.
- Target mote of the Base Station mote is going to receive them and get the signal strength and perform measurements and find distance based on signal strength.

3rd week:

- Finished writing the algorithm and tested the program of localization.
- Estimated the distance x using only one beacon mote first.
- Determined method to obtain the optimum x value by using both beacons.

4th week:

- Compared the results from 1 beacon and by using 2 beacons.
- Made the demo ready.
- Prepared the report to submit.

The work was divided into 5 phases:

1. Study of the technologies like the MicaZ, RSSI code to be used.

2. Design of the architecture in Deployment and Component for the WSN network.

3. Implementation on WSN MicaZ nodes in nesC using TinyOS;

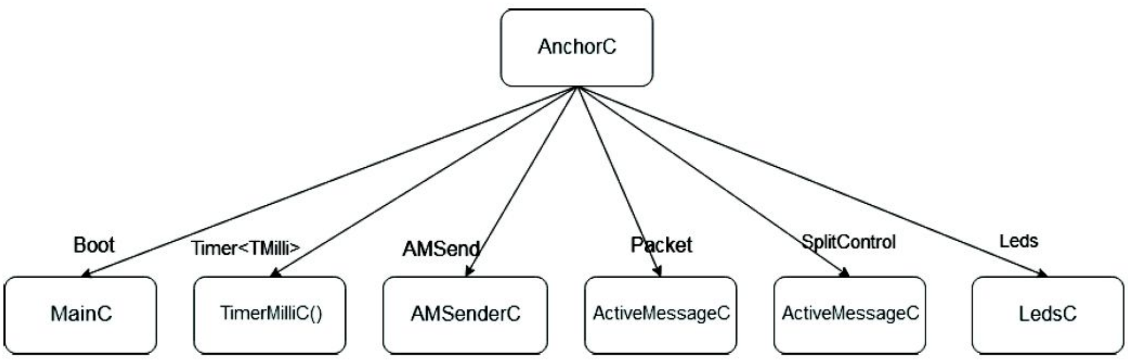
4. Test of the architecture in indoor situations with collection of the data generated by it.

5. Analysis of data collected by testing it in an empty parking lot..

Design :

Two types of nodes with distinct functionalities are identified in the project:

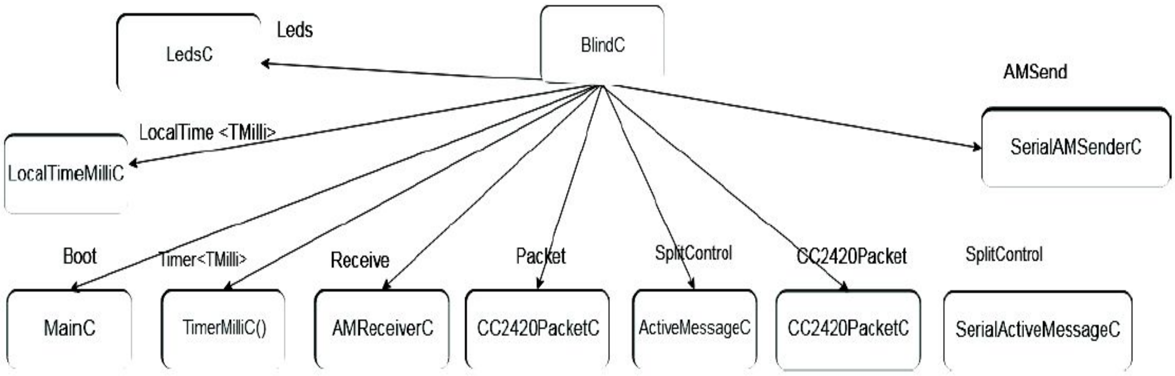
1. **Anchor /Sending part:**



-Identifies by a node ID (with values from 1 to N)

-Sends packets to BaseStation over a certain time interval via radio to provide its location based on the RSSI.

2. **BaseStation /Blind/Unknown mote part:**



-Reception of Beacon.
-Extraction of the contents of Beacon received so as to make the necessary data available to compute the distance
-Computation of the localization algorithm and to find the position of the unknown mote.

Implementation and Work Done:

This project is divided into two parts

Part-1: Obtain the path loss factor in a parking lot:

Experiment Set up area:

- The first mote was set as a base station for packet reception and computations.

- The second mote was set to be a transmitter that will be placed at different distances from the receiver transmitting.

The below code snippet is of the Sending moteC which sends the 10 packets for 10 seconds along with the node ID: (time interval set to 1s or 1000ms)

```
  event void SendTimer.fired(){

    //printf("[BEACON %d] Broadcasting beacon... \n", TOS_NODE_ID);

    if (count<10){
    SendPacket();
    count++;

     }

  }

event void RssiMsgSend.sendDone(message_t *m, error_t error){
   SentBlink();
}

void SendPacket(){
  RssiMsg* packet = (RssiMsg*) (call RssiMsgSend.getPayload(&msg, sizeof (RssiMsg)));
  packet->beacon_id = BEACON_ID;
  call RssiMsgSend.send(AM_BROADCAST_ADDR, &msg, sizeof(RssiMsg));
  SuccessBlink();
}
}
```

At each location, the transmitter mote transmits a certain number of packets, here it is 10 packets before being moved to the next location- 10 ft apart.

Formula to calculate path loss exponent:

$$n = \frac{\{P_L(d_i) - P_L(d_0)\}}{10 log_{10}\left(\frac{d_i}{d_0}\right)}$$

Where, PL(d0) is the reference power or 1 m RSSI which is -23 dBm in this case
PL(di) is the obtained RSSI value
di and d0 are the obtained distance and reference 1m distance respectively.

After a certain number of repeated measurements, The path loss exponent is calculated from average received signal strength (RSS) measurements from the different known distances, and also record the rate of successful packet reception at each distance.

Hence based on the formula, it is seen that in the environment of the Parking lot, the n is approximately 2.2 and inside an apartment, the path loss exponent n is 3.5.

Part-2: Determining distances by Beacon motes:

Two Anchors were placed at a certain distance sending periodic beacon messages including their nodeID.

- A Target mote placed at an unknown distance x meters from one end to receive these beacons, estimate the received RSSI values of the beacons, and transmit to the base-station.

Helper functions for SendingMoteC:

```
void SentBlink() {
  call Leds.led1Toggle();
}

void FailBlink() { // If a packet Reception over Radio fails, Led0 is toggled
  call Leds.led0Toggle();
}

void SuccessBlink() { //// If a packet Reception over Radio is successful, Led2 is toggled
  call Leds.led2Toggle();
}
```

SendingMoteC: sending packets with nodeID :

```
  event void SendTimer.fired(){

    //printf("[BEACON %d] Broadcasting beacon... \n", TOS_NODE_ID);
    SendPacket();

  }

event void RssiMsgSend.sendDone(message_t *m, error_t error){
   SentBlink();  //calls sent blink function
}

//funtion that sends packets to BS
void SendPacket(){
  RssiMsg* pkt = (RssiMsg*) (call RssiMsgSend.getPayload(&msg, sizeof (RssiMsg)));
  pkt->beacon_id = TOS_NODE_ID;
  call RssiMsgSend.send(AM_BROADCAST_ADDR, &msg, sizeof(RssiMsg));
  SuccessBlink();
}
}
```

The base station will estimate the distance x of the target mote along the line and display it on the screen.

RSSI to Distance Relation:

$$\text{RSSI}\ [\text{dBm}] = -10n\ \log_{10}(d) + A\ [\text{dBm}]$$

where , n is the path loss exponent, d is the distance from the sender and A is the RSSI at 1m distance which is -23dBm.

Function for conversion of RSSI to distance :

```
//funtion to convert rssi to dist
double distFromRSSI(int16_t rssi) {

    double p;
    double alpha = 3.52;   //indoor value
    //Formula
    p = (-23-rssi)/(10*alpha);   //-23 for 1m rssi
    res = pow(10, p);

    //printf("dist %f\n",res);
    //printfflush();
    return res;
  }
```

The above function returns the floating point value of the distance as a result.

Code snippet of Base Station:

```
message_t* receive(message_t *msg, void *payload, uint8_t len, am_id_t id) {


  message_t *ret = msg;

  RssiMsg* pkt;
  if (!signal RadioIntercept.forward[id](msg,payload,len))
    return ret;
  pkt = (RssiMsg*) payload;

  //Distance value sent to dist
  pkt->dist = distFromRSSI(pkt->rssi);
  //printf("Testing dist.. %f %d \n",pkt->dist,pkt->rssi);
  //printfflush();
```

When the message is received from the Sending mote, the RSSI value is converted to distance by using the above distfromRSSI function.

```
//to calculate mse based on the beacon node id
if(pkt->beacon_id == 1)
{
    d1=res;
}
if(pkt->beacon_id == 2)
{
    d2=res;
}


errs = sqrt(pow((d1),2) + pow((d2),2));

//mean square error
mserr=(errs/2);

pkt->mse = mserr;

//printf("Testing mse  ..%d %d %d\n",(int)pkt->mse,pkt->rssi, (int)res);
//printfflush();
```

At the end of this initialization, reception and distance calculation phase, the algorithm for the MSE is implemented which accounts for minimizing the cost function and helps in reducing the mean square error between estimated and real position (also called MSE).

Formula:

$$\text{Localization Error (LE)} = \sqrt{(x^{i}_{est} - x^{i}_{a})^2 + (y^{i}_{est} - y^{i}_{a})^2}$$

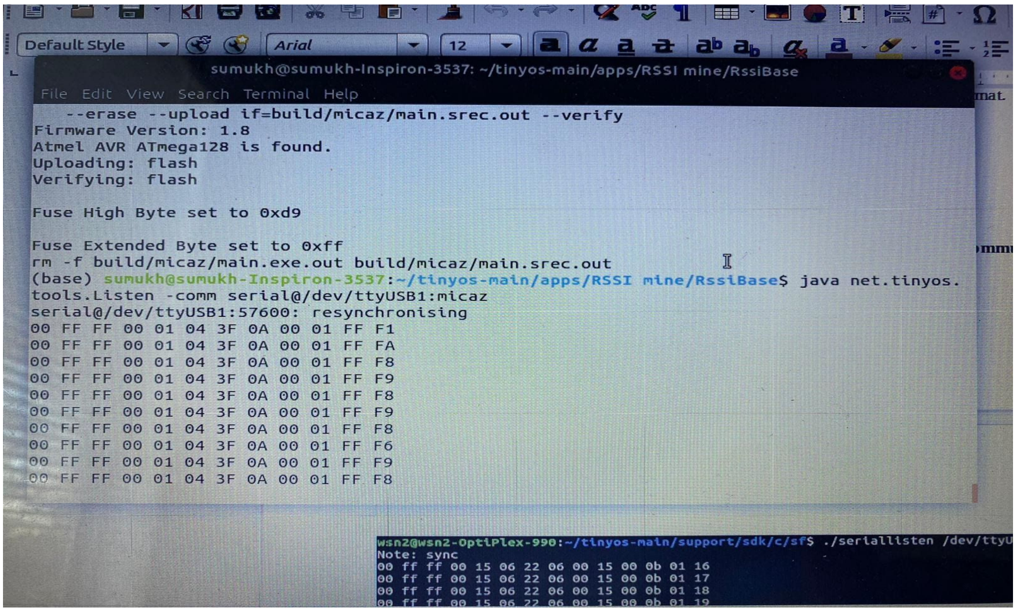
The algorithm takes the distances from both the node ids and based on them, coordinates the position of the mobile mote as the average of the distances that separate the mobile mote.

```
  atomic {
    if (!uartFull)
{
  ret = uartQueue[uartIn];
  uartQueue[uartIn] = msg;

  uartIn = (uartIn + 1) % UART_QUEUE_LEN;

  if (uartIn == uartOut)
    uartFull = TRUE;

  if (!uartBusy)
    {
      post uartSendTask();
      uartBusy = TRUE;
    }
}
    else
dropBlink();
  }

  return ret;
}
```

Thus finally an optimum distance is found by which the position of the target mote is found. This position of the Mobile mote is found which is later added to the UART Queue and is displayed in the output console.

Results:

Part 1:

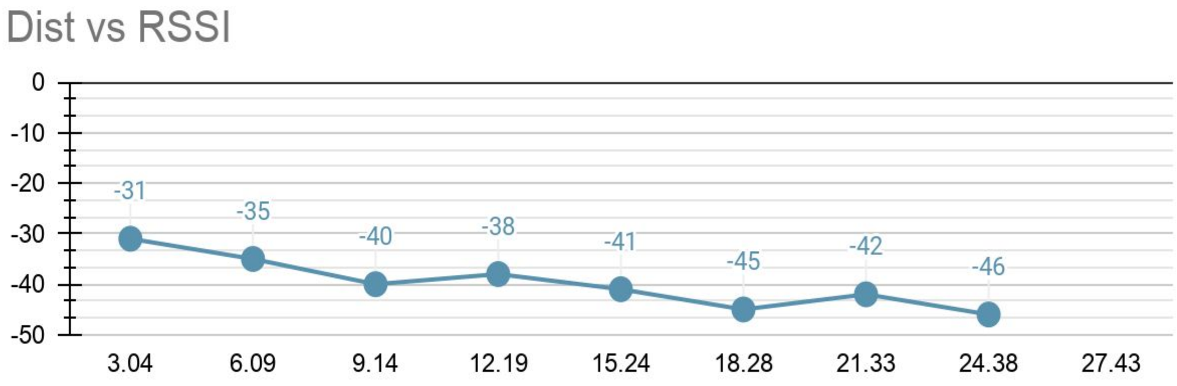


Output of Base Station receiving 10 packets at every 10 ft distances.

Table 1 : Distance vs RSSI:

Distance (m)	Distance (ft)	RSSI [dBm]	Rate of success(in%)
3.04	10	-31	100
6.09	20	-35	100
9.14	30	-40	100
12.19	40	-38	100
15.24	50	-41	100
18.28	60	-45	100
21.33	70	-42	90
24.38	80	-46	60
27.43	90	-	00

Graph 1 : Dist in m vs RSSI in dBm:



Part 2 Results:

Distance calculation based on RSSI by single Beacon node:

Actual Distance (m)	RSSI r1 [dBm]	Position of target mote (m)
1.5	-25	1.29
2.5	-26	2.57
3.5	-31	3.88
4.5	-33	4.74
5.5	-33	5.21
6.5	-36	6.12

Table 2: Distances with single Beacon Node 1

Results with 2 Beacon motes: (sending motes are separated by 1m distance)

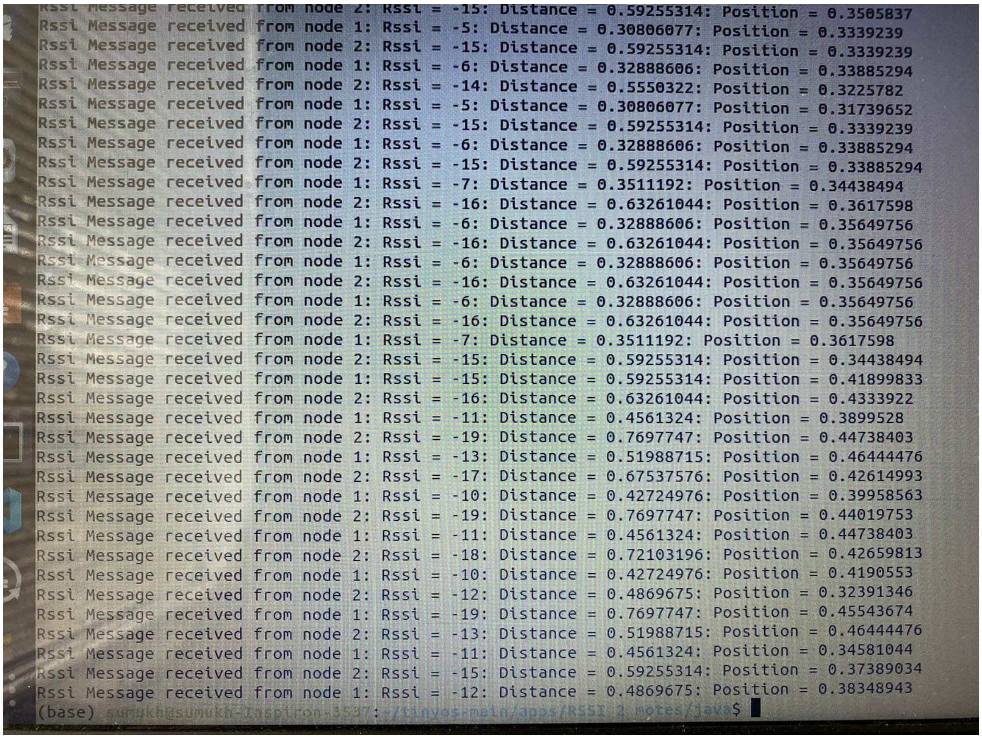


Table 2: Distance estimated with 2 Beacon Nodes (1 and 2):

RSSI 1 [dBm]	Distance 1 (m approx)	RSSI 2 [dBm]	Distance 2 (m approx)	Position of target mote (m)
-25	1.29	-31	3.04	1.52
-26	2.57	-33	4.59	2.63
-31	3.88	-35	5.74	3.46
-33	4.74	-38	7.69	4.51
-33	5.21	-41	9.17	5.27
-36	6.12	-44	10.43	6.04

Comparison between the results :

Position of target mote (m) With single mote -approx	Position of target mote(m) With 2 motes-approx	Actual Distance (m)
1.29	1.52	1.5
2.57	2.63	2.5
3.88	3.46	3.5
4.74	4.51	4.5
5.21	5.27	5.5
6.12	6.04	6.5

It can be seen that the optimum distance found by using 2 motes provides a better location of the unknown mobile mote than that of the position provided by a single mote.

Discussion:

Difficulties faced:

1. The setting up of sending motes and receiving motes in the parking lot was a little complicated since the mote wasn't sending messages properly when kept on ground and hence needed a little bit of elevation.
2. The selection of proper parking lot spaces with direct line of sight was tough since the parking lots had uneven surfaces and had slopes.
3. In the implementation part of the project, one of the initial difficulty was to understand how the RSSI in the tinyos works i.e., to understand the RSSIDemo code and how the code is formed through various components.
4. The conversion of the RSSI to distance and later to have the localization error implemented by giving proper thoughts for logic.

What should have been done:

1. Even though the results were shown in the demo, the inclusion of the distances of the sending motes ex: (0-1m) , (0-3m) or (0-5m) etc was not shown in the results above and hadn't been properly explained in the demo which is a mistake from my side.
2. Implementation of the localization error calculation and finally the mean square error had to be correlated with that of the distances and calibrated. Since I am calculating only the Localization error and not calibrating it with distances in the right way, I believe this should have been properly implemented by me.

CODE:

SendingMoteAppC.nc :

```
#include "RssiDemoMessages.h"

configuration SendingMoteAppC {
} implementation {
 components ActiveMessageC, MainC, LedsC;
 components new AMSenderC(AM_RSSIMSG) as RssiMsgSender;
 components new TimerMilliC() as SendTimer;
 components SendingMoteC as App;


 App.Boot -> MainC;
 App.SendTimer -> SendTimer;
 App.RssiMsgSend -> RssiMsgSender;
 App.RadioControl -> ActiveMessageC;
 App.Leds -> LedsC;
}
```

SendingMoteC.nc:

```
#include "ApplicationDefinitions.h"
#include "RssiDemoMessages.h"
#include "Timer.h"

module SendingMoteC {
 uses interface Boot;
 uses interface Leds;
 uses interface Timer<TMilli> as SendTimer;
 uses interface AMSend as RssiMsgSend;
 uses interface SplitControl as RadioControl;
}

 implementation {
 message_t msg;

 void SendPacket();

 void SentBlink() {
   call Leds.led1Toggle();
```

```
 }

 void FailBlink() { // If a packet Reception over Radio fails, Led0 is
toggled
   call Leds.led0Toggle();
 }

 void SuccessBlink() { // If a packet Reception over Radio is
successful, Led2 is toggled
   call Leds.led2Toggle();
 }

 event void Boot.booted(){
   //printf("[BEACON %d] Mote booted...\n", TOS_NODE_ID);
   call RadioControl.start();
 }

 event void RadioControl.startDone(error_t result){
   call SendTimer.startPeriodic(SEND_INTERVAL_MS);
 }

 event void RadioControl.stopDone(error_t result){
}


 event void SendTimer.fired(){
   //printf("[BEACON %d] Broadcasting beacon... \n", TOS_NODE_ID);
   SendPacket();
 }

 event void RssiMsgSend.sendDone(message_t *m, error_t error){
  SentBlink();  //calls sent blink function
}

//function that sends packets to BS
void SendPacket(){
 RssiMsg* pkt = (RssiMsg*) (call RssiMsgSend.getPayload(&msg, sizeof
(RssiMsg)));
 pkt->beacon_id = TOS_NODE_ID;
 call RssiMsgSend.send(AM_BROADCAST_ADDR, &msg, sizeof(RssiMsg));
 SuccessBlink();
}
}
```

RssiDemoMessages.h:

```
#ifndef RSSIDEMOMESSAGES_H__
#define RSSIDEMOMESSAGES_H__

enum {
 AM_RSSIMSG = 10
};

typedef nx_struct RssiMsg{
 nx_uint16_t beacon_id;
 nx_int16_t rssi;
 nx_float dist;
 nx_float mse;
} RssiMsg;

#endif //RSSIDEMOMESSAGES_H__
```

ApplicationDefinitions.h :

```
#ifndef APPLICATIONDEFINITIONS_H__
#define APPLICATIONDEFINITIONS_H__

enum {
 SEND_INTERVAL_MS = 1000, //to send packets for every 1 second
};

#endif //APPLICATIONDEFINITIONS_H__
```

BaseStationC.nc :

```
#include "printf.h"

configuration BaseStationC {
 provides interface Intercept as RadioIntercept[am_id_t amid];
 provides interface Intercept as SerialIntercept[am_id_t amid];
}
implementation {
 components MainC, BaseStationP, LedsC;
 components ActiveMessageC as Radio, SerialActiveMessageC as Serial;
```

```
 RadioIntercept = BaseStationP.RadioIntercept;
 SerialIntercept = BaseStationP.SerialIntercept;

 components PrintfC;
 components SerialStartC;
  MainC.Boot <- BaseStationP;

 BaseStationP.RadioControl -> Radio;
 BaseStationP.SerialControl -> Serial;
  BaseStationP.UartSend -> Serial;
 BaseStationP.UartReceive -> Serial;
 BaseStationP.UartPacket -> Serial;
 BaseStationP.UartAMPacket -> Serial;
  BaseStationP.RadioSend -> Radio;
 BaseStationP.RadioReceive -> Radio.Receive;
 BaseStationP.RadioSnoop -> Radio.Snoop;
 BaseStationP.RadioPacket -> Radio;
 BaseStationP.RadioAMPacket -> Radio;
  BaseStationP.Leds -> LedsC;
}
```

BasteSationP.nc:

```
#include "AM.h"
#include "Serial.h"
#include "math.h"
#include "printf.h"
#include "float.h"

module BaseStationP @safe() {
 uses {
   interface Boot;
   interface SplitControl as SerialControl;
   interface SplitControl as RadioControl;

   interface AMSend as UartSend[am_id_t id];
   interface Receive as UartReceive[am_id_t id];
   interface Packet as UartPacket;
   interface AMPacket as UartAMPacket;

   interface AMSend as RadioSend[am_id_t id];
   interface Receive as RadioReceive[am_id_t id];
   interface Receive as RadioSnoop[am_id_t id];
```

```
    interface Packet as RadioPacket;
    interface AMPacket as RadioAMPacket;

    interface Leds;
  }
   provides interface Intercept as RadioIntercept[am_id_t amid];
  provides interface Intercept as SerialIntercept[am_id_t amid];
}

implementation
{
  enum {
    UART_QUEUE_LEN = 12,
    RADIO_QUEUE_LEN = 12,
  };

  message_t  uartQueueBufs[UART_QUEUE_LEN];
  message_t  *uartQueue[UART_QUEUE_LEN];
  uint8_t    uartIn, uartOut;
  bool       uartBusy, uartFull;

  message_t  radioQueueBufs[RADIO_QUEUE_LEN];
  message_t  *radioQueue[RADIO_QUEUE_LEN];
  uint8_t    radioIn, radioOut;
  bool       radioBusy, radioFull;

  task void uartSendTask();
  task void radioSendTask();
  double distFromRSSI(int16_t RSSI);

  void dropBlink() {
    call Leds.led2Toggle();
  }

  void failBlink() {
    call Leds.led2Toggle();
  }

  event void Boot.booted() {
    uint8_t i;

    for (i = 0; i < UART_QUEUE_LEN; i++)
      uartQueue[i] = &uartQueueBufs[i];
```

```
  uartIn = uartOut = 0;
  uartBusy = FALSE;
  uartFull = TRUE;

  for (i = 0; i < RADIO_QUEUE_LEN; i++)
    radioQueue[i] = &radioQueueBufs[i];
  radioIn = radioOut = 0;
  radioBusy = FALSE;
  radioFull = TRUE;

  call RadioControl.start();
  call SerialControl.start();
}

event void RadioControl.startDone(error_t error) {
  if (error == SUCCESS) {
    radioFull = FALSE;
  }
}

event void SerialControl.startDone(error_t error) {
  if (error == SUCCESS) {
    uartFull = FALSE;
  }
}

event void SerialControl.stopDone(error_t error) {}
event void RadioControl.stopDone(error_t error) {}

//globally declared
uint8_t count = 0;
float d1;    //distance1
float d2;    //distance2
float errs;  //error
float mserr;  //mean square erro
double res;   //rssi to distance converted.

message_t* receive(message_t* msg, void* payload,
       uint8_t len, am_id_t id);
 event message_t *RadioSnoop.receive[am_id_t id](message_t *msg,
             void *payload,
             uint8_t len) {
  return receive(msg, payload, len, id);
```

```
}
 event message_t *RadioReceive.receive[am_id_t id](message_t *msg,
              void *payload,
              uint8_t len) {
   return receive(msg, payload, len, id);
 }

 message_t* receive(message_t *msg, void *payload, uint8_t len, am_id_t
id) {


   message_t *ret = msg;

   RssiMsg* pkt;
   if (!signal RadioIntercept.forward[id](msg,payload,len))
     return ret;
   pkt = (RssiMsg*) payload;

   //Distance value sent to dist
   pkt->dist = distFromRSSI(pkt->rssi);
   //printf("Testing dist.. %f %d \n",pkt->dist,pkt->rssi);
   //printfflush();

   //to calculate mse based on the beacon node id
   if(pkt->beacon_id == 1)
   {
       d1=res;
   }
   if(pkt->beacon_id == 2)
   {
       d2=res;
   }


   errs = sqrt(pow((d1),2) + pow((d2),2));

   //mean square error
   mserr=(errs/2);

   pkt->mse = mserr;

   //printf("Testing mse  ..%d %d %d\n",(int)pkt->mse,pkt->rssi,
(int)res);
```

```
  //printfflush();

  atomic {
    if (!uartFull)
{
  ret = uartQueue[uartIn];
  uartQueue[uartIn] = msg;

  uartIn = (uartIn + 1) % UART_QUEUE_LEN;
   if (uartIn == uartOut)
    uartFull = TRUE;

  if (!uartBusy)
    {
      post uartSendTask();
      uartBusy = TRUE;
    }
}
    else
dropBlink();
  }

  return ret;
}

uint8_t tmpLen;

//funtion to convert rssi to dist
double distFromRSSI(int16_t rssi) {

      double p;
      double alpha = 3.52;   //indoor value
      //Formula
      p = (-23-rssi)/(10*alpha);   //-23 for 1m rssi
      res = pow(10, p);

      //printf("dist %f\n",res);
      //printfflush();
      return res;
  }


task void uartSendTask() {
```

```
    uint8_t len;
    am_id_t id;
    am_addr_t addr, src;
    message_t* msg;
    atomic
      if (uartIn == uartOut && !uartFull)
  {
    uartBusy = FALSE;
    return;
  }

    msg = uartQueue[uartOut];
    tmpLen = len = call RadioPacket.payloadLength(msg);
    id = call RadioAMPacket.type(msg);
    addr = call RadioAMPacket.destination(msg);
    src = call RadioAMPacket.source(msg);
    call UartAMPacket.setSource(msg, src);

    if (call UartSend.send[id](addr, uartQueue[uartOut], len) ==
SUCCESS)
      call Leds.led1Toggle();
    else
      {
  failBlink();
  post uartSendTask();
      }
  }

  event void UartSend.sendDone[am_id_t id](message_t* msg, error_t
error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
  if (msg == uartQueue[uartOut])
    {
      if (++uartOut >= UART_QUEUE_LEN)
        uartOut = 0;
      if (uartFull)
        uartFull = FALSE;
    }
    post uartSendTask();
  }
```

```
event message_t *UartReceive.receive[am_id_t id](message_t *msg,
             void *payload,
             uint8_t len) {
  message_t *ret = msg;
  bool reflectToken = FALSE;

  if (!signal SerialIntercept.forward[id](msg,payload,len))
    return ret;

  atomic
    if (!radioFull)
{
  reflectToken = TRUE;
  ret = radioQueue[radioIn];
  radioQueue[radioIn] = msg;
  if (++radioIn >= RADIO_QUEUE_LEN)
    radioIn = 0;
  if (radioIn == radioOut)
    radioFull = TRUE;

  if (!radioBusy)
    {
      post radioSendTask();
      radioBusy = TRUE;
    }
}
    else
dropBlink();

  if (reflectToken) {
    //call UartTokenReceive.ReflectToken(Token);
  }

  return ret;

}

task void radioSendTask() {
  uint8_t len;
  am_id_t id;
  am_addr_t addr;
  message_t* msg;
```

```
    atomic
      if (radioIn == radioOut && !radioFull)
  {
    radioBusy = FALSE;
    return;
  }

    msg = radioQueue[radioOut];
    len = call UartPacket.payloadLength(msg);
    addr = call UartAMPacket.destination(msg);
    id = call UartAMPacket.type(msg);
    if (call RadioSend.send[id](addr, msg, len) == SUCCESS)
      call Leds.led0Toggle();
    else
      {
  failBlink();
  post radioSendTask();
      }
  }

  event void RadioSend.sendDone[am_id_t id](message_t* msg, error_t
error) {
    if (error != SUCCESS)
      failBlink();
    else
      atomic
  if (msg == radioQueue[radioOut])
    {
      if (++radioOut >= RADIO_QUEUE_LEN)
        radioOut = 0;
      if (radioFull)
        radioFull = FALSE;
    }

    post radioSendTask();
  }

  default event bool
  RadioIntercept.forward[am_id_t amid](message_t* msg,
                void* payload,
                uint8_t len) {
    return TRUE;
```

```
 }

 default event bool
 SerialIntercept.forward[am_id_t amid](message_t* msg,
         void* payload,
         uint8_t len) {
   return TRUE;
 }
}
```

RssiBaseAppC.nc:

```
#include "RssiDemoMessages.h"
#include "message.h"

configuration RssiBaseAppC {
} implementation {
 components BaseStationC;
 components RssiBaseC as App;

#ifdef __CC2420_H__
 components CC2420ActiveMessageC;
 App -> CC2420ActiveMessageC.CC2420Packet;
#elif  defined(PLATFORM_IRIS)
 components  RF230ActiveMessageC;
 App -> RF230ActiveMessageC.PacketRSSI;
#elif defined(PLATFORM_UCMINI)
 components  RFA1ActiveMessageC;
 App -> RFA1ActiveMessageC.PacketRSSI;
#elif defined(TDA5250_MESSAGE_H)
 components Tda5250ActiveMessageC;
 App -> Tda5250ActiveMessageC.Tda5250Packet;
#endif
  App-> BaseStationC.RadioIntercept[AM_RSSIMSG];
}
```

RssiBaseC.nc:

```
#include "ApplicationDefinitions.h"
#include "RssiDemoMessages.h"


module RssiBaseC {
 uses interface Intercept as RssiMsgIntercept;
```

```
#ifdef __CC2420_H__
 uses interface CC2420Packet;
#elif defined(TDA5250_MESSAGE_H)
 uses interface Tda5250Packet;
#else
 uses interface PacketField<uint8_t> as PacketRSSI;
#endif
}

implementation {

 /*//queue buffer to cache serial messages
 message_t serialQueueBuf[SERAIL_QUEUE_LEN];
 message_t* serialQueue[SERAIL_QUEUE_LEN];
 uint8_t queueIn, queueOut;
 bool queueBusy, queueFull;

 */
 uint16_t getRssi(message_t *msg);
  event bool RssiMsgIntercept.forward(message_t *msg,
             void *payload,
             uint8_t len) {
   RssiMsg *rssiMsg = (RssiMsg*) payload;
   rssiMsg->rssi = getRssi(msg);

   return TRUE;
 }

#ifdef __CC2420_H__
 uint16_t getRssi(message_t *msg){
   return (uint16_t) call CC2420Packet.getRssi(msg);
 }
#elif defined(CC1K_RADIO_MSG_H)
   uint16_t getRssi(message_t *msg){
   cc1000_metadata_t *md =(cc1000_metadata_t*) msg->metadata;
   return md->strength_or_preamble;
 }
#elif defined(PLATFORM_IRIS) || defined(PLATFORM_UCMINI)
 uint16_t getRssi(message_t *msg){
   if(call PacketRSSI.isSet(msg))
     return (uint16_t) call PacketRSSI.get(msg);
   else
```

```
     return 0xFFFF;
 }
#elif defined(TDA5250_MESSAGE_H)
  uint16_t getRssi(message_t *msg){
      return call Tda5250Packet.getSnr(msg);
  }
#else
 #error Radio chip not supported! This demo currently works only \
        for motes with CC1000, CC2420, RF230, RFA1 or TDA5250 radios.
#endif
}
```

RssiDemo.java:

```
import java.io.IOException;

import net.tinyos.message.*;
import net.tinyos.packet.*;
import net.tinyos.util.*;

public class RssiDemo implements MessageListener {

 private MoteIF moteIF;
  public RssiDemo(MoteIF moteIF) {
   this.moteIF = moteIF;
   this.moteIF.registerListener(new RssiMsg(), this);
 }

 public void messageReceived(int to, Message message) {
   RssiMsg msg = (RssiMsg) message;
   int source = message.getSerialPacket().get_header_src();
   System.out.println("Rssi Message received from node " +
msg.get_beacon_id() +
          ": Rssi = " +  msg.get_rssi() + ": Distance = " +
msg.get_dist() + ":
          Position = " +  msg.get_mse());
 }
  private static void usage() {
   System.err.println("usage: RssiDemo [-comm <source>]");
 }
  public static void main(String[] args) throws Exception {
   String source = null;
   if (args.length == 2) {
```

```java
    if (!args[0].equals("-comm")) {
 usage();
 System.exit(1);
    }
    source = args[1];
  }
  else if (args.length != 0) {
    usage();
    System.exit(1);
  }

  PhoenixSource phoenix;

  if (source == null) {
    phoenix = BuildSource.makePhoenix(PrintStreamMessenger.err);
  }
  else {
    phoenix = BuildSource.makePhoenix(source,
PrintStreamMessenger.err);
  }

  MoteIF mif = new MoteIF(phoenix);
  RssiDemo serial = new RssiDemo(mif);
 }
}
```