Lab 2

- Goal 1: Program the critical part of an application
- Goal 2: Use the radio component to have motes communicate
A wireless application

• Problem:
  – Create an application called **Broadcast** that
    • makes a gateway broadcast packets every second
    • makes the red LED of a mote toggle each time it receives a packet (from the gateway)

• Solution:
  – Raise a timer every second at the gateway node
  – Each time the timer expires, we send a packet
  – Each time a packet is received, we toggle the red LED
Architecture of Broadcast
Broadcast application

- the gateway (id==0) keeps broadcasting packets
- when receiving a packet, a node (id!=0) toggles the red LED

• Create a directory Broadcast in your home directory (say muc/Broadcast)

• Copy Makefile, BroadcastC.nc, BroadcastM.nc and Broadcast.h to this directory (they are in a shared directory)

• Edit the files and fill in the blanks

• Compile, run and check the application
Hints

• If your module provides interface Intf
  - open /opt/tinyos-1.x/interfaces/Intf.nc
  - ensure that you implement all the commands

• If your module requires interface Intf
  - open /opt/tinyos-1.x/interfaces/Intf.nc
  - ensure that you implement all the events

• To test the application, a good filter might be
  - export DBG=am,led

• http://www.tinyos.net/api/tinyos-1.x/doc/tutorial/lesson5.html
Makefile

COMPONENT = BroadcastC

include /opt/tinyos-1.x/apps/Makerules
#ifndef BROADCAST_COUNT_H
#define BROADCAST_COUNT_H

enum {
    AM_COUNT_MSG = 100,
};

typedef struct Count_Msg {
    uint16_t value;  // we do not use this field currently
} Count_Msg;

#endif
BroadcastC.nc

includes Broadcast;

configuration BroadcastC {
}

implementation {
    components BroadcastM, LedsC, TimerC,
           GenericComm as Comm, Main;

    Main.StdControl -> BroadcastM;
    Main.StdControl -> TimerC;
    Main.StdControl -> Comm.Control;

    BroadcastM.Leds -> LedsC;
    BroadcastM.CountTimer ->
               TimerC.Timer[unique(“Timer”)];
    BroadcastM.SendCountMsg ->
               Comm.SendMsg[AM_COUNT_MSG];
    Broadcast.ReceiveCountMsg ->
               Comm.ReceiveMsg[AM_COUNT_MSG];
}
module BroadcastM { 
    provides { 
        interface StdControl;
    }
    uses { 
        interface Leds;
        interface Timer as BlinkTimer;
        interface SendMsg as SendCountMsg;
        interface ReceiveMsg as ReceiveCountMsg;
    }
}

implementation { 
    ... // see next slides
}
implementation {

    TOS_Msg message; // “message” has to be global

    task void SendCountTask() {
        Count_Msg * payload;
        payload = (Count_Msg *)message.data;
        payload->value = 0x1234;
        call SendCountMsg.send(TOS_BCAST_ADDR,
                               sizeof(Count_Msg), &message);
    }

    command result_t StdControl.init() { 
        call Leds.init();
        return SUCCESS;
    }

    command result_t StdControl.stop() { 
        return SUCCESS;
    }

    // ...
}
command result_t StdControl.start() {
    if (TOS_LOCAL_ADDRESS==0) {
        call CountTimer.start(TIMER_REPEAT, 1024);
    }
    return SUCCESS;
}

event result_t CountTimer.fired() {
    post SendCountTask();
}

event result_t SendCountMsg.sendDone(
    TOS_MsgPtr message, result_t result) {
    return SUCCESS;
}

// ...
event TOS_MsgPtr ReceiveCountMsg.receive(TOS_MsgPtr receivedMessage) {
    Count_Msg * payload;
    uint16_t value;
    if (TOS_LOCAL_ADDRESS!=0) {
        payload = (Count_Msg *)receivedMessage->data;
        value = payload->value;
        call Leds.redToggle();
    }
    return receivedMessage;
}