BaseStation Based on GenericComm

Aly El-Osery
Electrical Engineering Dept.
New Mexico Tech
Socorro, NM
Scenario

- Have a node send a message to the basestation.
- Basestation forwards message to UART.
- Basestation sends a message to other nodes based on a timer event.
AMStandard

- The implementation module of GenericComm
tos/system/AMStandard.nc
- Message get sends with an id taken from the type field of the packet and used to identify the parametrized interface.
- Messages sent with a certain id has to be received with the same id.

```
DataToUARTM.Send -> Comm.SendMsg[id];
ReceiveDataM.ReceiveMsg ->
GenericComm.ReceiveMsg[id];
```
Sending Message

```
call Send.send(ADDR, sizeof(IntMsg), &data)
```

If ADDR=TOS_UART_ADDR send to UART
If ADDR=TOS_BCAST_ADDR send to all nodes over radio
SendMsg AMStandard.nc

// Command to accept transmission of an Active Message
command result_t SendMsg.send(uint8_t id)(uint16_t addr, uint8_t length, TOS_MsgPtr data) {
  if (!state) {
    state = TRUE;
    if (length > DATA_LENGTH) {
      dbg(DBG_AM, "AM: Send length too long: %i. Fail.\n", (int)length);
      state = FALSE;
      return FAIL;
    }
    if (!(post sendTask())) {
      dbg(DBG_AM, "AM: post sendTask failed.\n");
      state = FALSE;
      return FAIL;
    }
    else {
      buffer = data;
      data->length = length;
      data->addr = addr;
      data->type = id;
      buffer->group = TOS_AM_GROUP;
      dbg(DBG_AM, "Sending message: %hx, %hhx\n\t", addr, id);
      dbgPacket(data);
    }
    return SUCCESS;
  }
}
// This task schedules the transmission of the Active Message

void sendTask() {
    result_t ok;
    TOS_MsgPtr buf;
    buf = buffer;
    if (buf->addr == TOS_UART_ADDR)
        ok = call UARTSend.send(buf);
    else
        ok = call RadioSend.send(buf);

    if (ok == FAIL) // failed, signal completion immediately
        reportSendDone(buffer, FAIL);
}
// Handle the event of the reception of an incoming message
TOS_MsgPtr received(TOS_MsgPtr packet) __attribute__((C, spontaneous)) {
    uint16_t addr = TOS_LOCAL_ADDRESS;
    counter++;
    dbg(DBG_AM, "AM_address = %hx, %hhx; counter:%i\n", packet->addr, packet->type, (int)counter);
    if (packet->crc == 1 && // Uncomment this line to check crcs
        packet->group == TOS_AM_GROUP &&
        (packet->addr == TOS_BCAST_ADDR ||
         packet->addr == addr))
    {
        uint8_t type = packet->type;
        TOS_MsgPtr tmp;
        // Debugging output
        dbg(DBG_AM, "Received message:\n\t");
        dbgPacket(packet);
        dbg(DBG_AM, "AM_type = %d\n", type);
        // dispatch message
        tmp = signal ReceiveMsg.receive[type](packet);
        if (tmp)
            packet = tmp;
    }
    return packet;
}
BaseStation.nc
configuration BaseStation {
}
implementation {
    components Main, ReceiveData, DataToUART, MsgToRfm;

    Main.StdControl -> ReceiveData.StdControl;
    Main.StdControl -> DataToUART.StdControl;
    Main.StdControl -> MsgToRfm.StdControl;
    ReceiveData.MsgOutput -> DataToUART;
}

MsgOutput is based on IntOutput but it sends the entire data field instead of an single int value
ReceiveData.nc
ReceiveData.nc

includes IntMsg;

configuration ReceiveData {
    provides interface StdControl;
    uses interface MsgOutput;
}
implementation {
    components ReceiveDataM, GenericComm, LedsC;

    MsgOutput = ReceiveDataM;
    StdControl = ReceiveDataM;
    ReceiveDataM.ReceiveMsg -> GenericComm.ReceiveMsg[AM_INTMSG];
    ReceiveDataM.CommControl -> GenericComm;
    ReceiveDataM.Leds -> LedsC;
}
includes IntMsg;

class configuration DataToUART
{
    provides {
      interface MsgOutput;
      interface StdControl;
    }
}

class implementation
{
  components DataToUARTM, GenericComm as Comm, LedsC;

  MsgOutput = DataToUARTM;
  StdControl = DataToUARTM;

  DataToUARTM.Send -> Comm.SendMsg[AM_INTMSG];
  DataToUARTM.SubControl -> Comm;
  DataToUARTM.Leds -> LedsC;
}
DataToUART.nc
command result_t MsgOutput.output(IntMsg msg)
{
    IntMsg *message = (IntMsg *)data.data;
    if (!pending)
    {
        pending = TRUE;
        call Leds.yellowOn();
        atomic{
            *message = msg;
        }
        if (call Send.send(TOS_UART_ADDR, sizeof(IntMsg), &data))
            return SUCCESS;

        pending = FALSE;
    }
    return FAIL;
}
MsgToRfm.nc
configuration MsgToRfm {
    provides interface StdControl;
}
implementation {
    components Main, Counter, IntToRfm, TimerC;

    StdControl = Counter.StdControl;
    StdControl = IntToRfm.StdControl;
    StdControl = TimerC.StdControl;
    Counter.Timer -> TimerC.Timer[unique("Timer")];
    Counter.IntOutput -> IntToRfm;
}
Output

- Node 1 is running CntToLedsAndRfm, the leds blink to represent the counter at a high rate.
- Node 2 is running BaseStation and forwards the incoming msg from CntToLedsAndRfm to the UART. Based on a 1sec timer it will also send a counter to Node 3.
- Node 3 running RfmToLeds displays the counter at a slower rate than Node 1.
- Because of packet id the Node 3 will not process data received from Node 1 and if BaseStation is not running the Leds will not run.
Known Problems

- While running `xlisten` to monitor the data being sent by BaseStation to UART, there is an extra byte that is so far unexplained.
- Also the `addr` field seems to be incorrect.