Exercise in Embedded Computing: Wireless Networking (under Contiki)

Sivan Toledo, Tel-Aviv University

December 5, 2009

This exercise focuses on communication protocols for low-power radio networking. We will experiment with radio networking under Rime, a stack of communication protocols that is part of Contiki. You will need two or more tmote sky boards for this exercise.

1. From the Contiki sources (at `/users/courses/embedded/contiki-2.3`), copy `examples/rime/example-unicast.c` into your working directory. Modify it so that it prints the 16-bit identifier of the node, which is stored in `rimeaddr_node_addr`. The type of this variable is defined in `core/net/rime/rimeaddr.h`, and it is normally printed as $x.y$ where both x and y are 8-bit numbers). Modify the program so that it prints the node's own address when on every timer event. Upload the code onto your onto one of your boards and connect to the board (using `make login`) and record the node's ID.
2. Modify the code so that it sends messages to your first node, recompile, and upload to *your second node*. Modify the code again to send messages to the second node, and upload the code to the *first node*. Power both, one through a USB port and the other through batteries or a USB port (including a USB port of a laptop; it does not need to run any driver for the node). Connect to one of the nodes and verify that it receives messages from the other. Do not proceed until you get the two nodes to communicate.
3. (Optional but useful) Modify the code to blink the green LED for 1/4 second when the program sends a message, and the red LED when it receives a message.
4. Test the range of the tmote sky's radio. Keep one node connected to a computer and move with the other node. Try to determine the range both through air with a line of sight and through various materials. Send me an email with a summary of your findings.
5. Compile and upload the code from `examples/rime/example-collection.c` to both your nodes. Connect to one of the nodes and press the user button on it. It should print "I am sink". Being a sink in the collection protocol causes the node to notify its neighbors that all messages should be routed to it. This information propagates and causes the creation of a spanning tree of the reachable nodes, with the sink as the root of the tree. The tree gets updated periodically. In this program, all nodes send messages periodically, and the sink prints all the messages it gets. In principle, your sink should see messages from your other node, but because your nodes communicate with all the other nodes in the room, it is possible that your nodes became non-root nodes in somebody else's tree when that somebody pressed the button.
6. This part is for the entire class, not for individual teams. The nodes of all the teams communicate with each other, so when one team's board sends a message, it is received by boards belonging to other teams. In this part we exploit this to explore the

collection protocol a bit more. One node in the entire class should be designated a sink, and it's user button should be pressed. From this point on, do not press the user button on the other nodes. Keep track of which node IDs the sink sees. Do you see all the nodes (that is, all the powered ones)? If not, try to move the nodes that you can't receive closer to the sink.

7. Try to increase the maximum hop count as much as you can. That is, try to get some nodes in range of the sink, but not very close. Then try to get other nodes far enough from the sink so that it does not see their messages, but close enough that messages can get to the sink in two hops. Use your range estimates from part 4.

8. The source code specifies a virtual communication channel to use for the protocol. Up to now all the teams used the same channel, so the boards of all the teams communicate with each other. Coordinate the use of channel numbers so that different teams are on different channels. Make sure you study the documentation of the collection module (in `core/net/rime/collect.h`) before you assign the channel numbers. Test your code and make sure that your sink remains a sink even if another team assigns one of as nodes as a sink.

9. The channel numbers in Rime are virtual; they are simply numbers that are part of each packet's header. But the 802.15.4 radio on the tmote sky can use 10 different channels. Modify the code to use a different radio channel (different frequency) using `cc2420_set_channel`.

10. (Optional). Communicating over a heavily used channel reduces the communication range and increases the power consumption (because many of the packets that the receiver hears turn out to be irrelevant). If the channel that is used is heavily used, it may be better to try a different channel. Design and implement a protocol that allows a node to notify another node that it wishes to switch to another frequency. You can assume that there are only two nodes in your network and that they know each other's Rime addresses. The protocol should prevent one node from switching to the new frequency while the other node stays on the old frequency.

11. (Optional) Rime is implemented on top of a MAC layer; the default MAC layer in Contiki is XMAC. Should the frequency switching be implemented as part of the MAC layer, or as part of Rime? Consider the case where an entire sensor network switches frequency, not just one node. Which layer should decide that it's a good idea to switch frequencies?