

Mote as Network Modem

Version 1.0 - July 14, 2005

We have developed a foundation module for RF communication that is fairly independent of host platform, operating system, programming language, and application.

Background in Brief

During the course of our research, we have found many situations where wireless peer-to-peer communications are advantageous. Situations specific to our project include, but are not limited to:

<u>Project</u>	<u>Wireless Application(s)</u>
Cognitive Bag	Bag to Base Station PC Bag to GPS Mote Bag to YMCA Bag to Bus Bag to Bus Station
Bus / Bus	Station Bus to Bus Station
Van Announcer	Van to Care Center PC
RFID Doorway	Doorway to Base Station PC

Early on, we conceptualized using motes as both wireless bridge and computational engine. Lately, however, it has been made clear that relying exclusively on motes in this dual-role may not be advantageous. Indeed, it is become clear that a mote alone does not have the horsepower for the more demanding tasks such as route navigation and trip scheduling. Further, the motes must be programmed using a language and methods not familiar to most students and this causes delays in idea implementation and can compromise system reliability.

We have arrived on a new hybrid solution where the mote is effectively married to a device with greater capability, such as an iPaq or in some cases a full-blown PC. This allows for the bulk of software development and innovation to take place on a more capable and familiar platform, where the mote itself is used as a communications peripheral more than a computing device unto itself.

Care has been taken to develop the mote software to serve as a system independent bridge from a wired connection (serial) to wireless (RF) and work with a number of platforms relevant to the foreseeable future. Considering the variety of peripherals and situations we expect in the coming years, it was important to design with an eye toward interoperable modes and clear paths for evolution.

Approaching an Ideal

The Mote as Network Modem satisfies the requirements outlined in our background discussion. The device allows for easy wireless connectivity between devices as divergent as Java applications on Windows, Terminal applications in Unix, and almost any other device with a serial interface.

Modes of Operation

The device offers several operational modes, each having been designed to meet specific demands. What follows is a brief overview of these modes.

Mode 0: Special "listen only" diagnostic mode.

This mode listens for TinyOS packets coming in over the radio and dumps their content in a human-readable form out USART0.

Mode 1: Packet forwarder, no CRC verification.

TinyOS packets received on USART0 are forwarded to the radio. Similarly, TinyOS packets received over the radio are forwarded to USART0. No CRC validations are performed on the data. That is to say if a bad packet is received from the radio, it will be forwarded to USART0 regardless.

Mode 2: Packet forwarder, CRC verification required.

As with mode 1, TinyOS packets received on USART0 are forwarded to the radio. Similarly, TinyOS packets received over the radio are forwarded to USART0. Unlike mode 1, mode 2 will not pass radio packets that fail CRC validation.

Mode 3: Message Mode

USART0 receives text-only serial data, and forwards the data to the radio only when a carriage-return is received. Messages can be a maximum of 200 characters in length, including the carriage-return. Keep in mind that messages longer than 24 bytes (including the carriage return) are broken into multiple packets.

In mode 3, radio messages received are stripped of non-printable characters, headers and footers, and forwarded to USART0.

This mode will discard any message that has not been terminated with a [cr] within three seconds. This helps ensure that transmissions are deliberate.

Also, CRC validation of radio packets is automatically enforced.

Mode 4: Virtual party-line serial link.

The serial stream received on USART0 is packetized and sent out the radio. Similarly, the payload of packets received from the radio are forwarded to USART0. This causes the radio to act as a half-duplex "party line" for all connected serial devices.

As with mode 3, CRC validation of radio packets is automatically enforced.

Command and Control

The mote can be controlled using familiar AT-style command strings, similar in spirit to a Hayes smart modem. From any operating mode, the device can be put into a "command" mode by sending it the string "ATCOMMAND" (without the quotes) on USART0, followed by a carriage return [cr] character. The device will enter command mode only if additional characters are not received on USART0 within three seconds. This period of silence enables the device to discriminate between a bonafide command and situations where the command string is coincidentally part of a larger transmission.

It is envisioned that a systems integrator will configure the device using a terminal program such as HyperTerminal. When the integrator enters the command string "ATCOMMAND" (without the quotes) followed by a carriage return, the following is displayed:

```
COMMAND MODE BEGIN
OK
```

The integrator could now get the menu of commands available. This is accomplished by typing "ATH" followed by a carriage return. The following is displayed:

```
COMMANDS:
ATBAUD4800 .. SETS 4800 BAUD
ATBAUD9600 .. SETS 9600 BAUD
ATBAUD19200 .. SETS 19200 BAUD
ATBAUD38400 .. SETS 38400 BAUD
ATBAUD57600 .. SETS 57600 BAUD
ATMODEn .. SETS OPERATING MODE 0-4
ATECHOn .. SETS MODE 3 ECHO 1=YES 0=NO
ATLEDn .. SETS LED MODE 1=ENABLED 0=DISABLE
AT2TXn .. TRANSMIT M3&4 RADIO PKTS TWICE 1=YES 0=NO
ATDEFAULT .. RESTORES SYSTEM DEFAULTS
ATSHOW .. SHOW CURRENT SETTINGS
ATSAVE .. EXIT COMMAND MODE & SAVE
ATEXIT .. EXIT COMMAND MODE WITHOUT SAVING
ATH OR AT? .. THIS SCREEN
OK
```

Many commands are self explanatory, such as baud rate and mode selection. The others

we'll discuss here.

The setting `ATECHO` causes mode 3 to echo characters locally. If echo is enabled, then everything that the device receives on `USART0` is automatically sent on `USART0`. For those using a terminal program to interact with the device, they will be able to see the characters as they type them--much like using an instant messenger application.

The setting `ATLED` either enables or disables the LEDs. Though it is usually most desirable to have LEDs enabled, it might be advantageous to disable them in some situations to save power.

The `AT2TX` setting applies to modes 3 and 4. When enabled, all packets are actually sent twice. This increases the odds of a packet getting through in some situations, but comes at the expense of increased network congestion. Note that modes 3 and 4 are intelligent enough to reject redundant packets, so that if the same packet is received twice within a small time window, the extra packet is dropped.

Use `ATSHOW` to display current settings. An example show is as follows:

```
SETTINGS:
MODE=3
BAUD=9600
ENABLE_LEDS=1
TRANSMIT M3&4 PACKETS TWICE=1
MODE 3 ECHO=1
OK
```

Command `ATSAVE` is used to save the settings into flash and exit command mode.

Command `ATEXIT` is used to exit command mode without saving the settings into flash. Note that any changes in settings take effect, but they are just not saved in persistent storage. This allows the operator to experiment with settings before committing them to flash.

System Defaults

The default settings for the device are as follows:

```
Mode=0
Baud=9600
LEDS=ENABLED
TRANSMIT MODE 3 & 4 PACKETS TWICE=NO
MODE 3 ECHO=NO
```

LEDs

When enabled, the LEDs communicate state information to the operator.

Green LED status: Normally on. (Red on Mica2Dot)

The green LED is on when the unit is idle or in command mode.

The green LED blinks when the packets are received from the radio.

The green LED blinks when packets are transmitted out the radio.

Red LED status: Normally off. (Not applicable to Mica2Dot)

The red LED is on when unit is in command mode.

The red LED flashes when a packet with invalid CRC is received on the radio.

Yellow LED status: Normally off. (Not applicable to Mica2Dot)

The yellow LED is on when unit is in command mode.

Yellow LED blinks as data is being received on USART0.

Yellow LED blinks as data is being transmitted on USART0.

Appendix A: Making the most of Mode 0

Mode 0 is designed to aid development of new devices and to troubleshoot existing systems. When the device is in this mode, it will listen for and print the essential details of any TinyOS packet received. Typical output is shown below.

```
Rx Pkt#110
  addr=FFFF
  type=80
  group=7D
  length=11
  crc=0001
  strngth=00B7
  data=0C.00.00.00.48.65.6C.6C.6F.20.77.6F.72.6C.64.21.0D
  .....H .e .l .l .o . .w .o .r .l .d .! .
```

The above example shows a case where a mode 3 packet was received with the effective payload of “Hello world!” When examining the sample, you’ll first notice the somewhat cryptic “Rx Pkt#110” which means the device has received 110 packets since being powered on. The number will increment to 111 for the next received packet, 112 for the next, and so on.

The rest of the output has 1-to-1 correlation with the TinyOS packet structure. Note that mode 3 and 4 packets are always of type 80h.

A nice feature of this display is the human-readable translation printed below the hexadecimal representation of packet data. This makes it easier to read and diagnose packets that have such human-readable content.

Appendix B: Examining modes 3 and 4 in further detail.

Modes 3 and 4 were designed to hide the gory details of packet structures from applications developers. For the developer who uses the device in either of these modes, wireless networking is as easy as communicating with the serial port. This means that for Java programmers, no knowledge beyond *javax.comm* is required.

Still, programmers must keep in mind that the device can only *approximate* the behavior of a true-blue comm port. Small details should be kept in mind when writing an application.

Consider mode 3, where data is buffered by the device and sent out the radio only when a [cr] is received, and recall that the maximum message length in this mode is 200 characters. All mode 3 packets can carry up to a maximum of 24 bytes of payload data, so it really takes 9 packets to send a 200 character message.

Mode 4 works in a similar way, but instead of expecting a [cr] to signal the end of a message, it looks for pauses in the data stream or, if no pauses are present, sends the buffered data over the radio when its USART0 receive buffer is approaching full.

Transmit Twice Mode

Radio communications are not always reliable, so any single packet may be dropped at any time. This can essentially delete large sections of our long message, so programmers should write code to expect this behavior. The problem is not new to this device, as even a regular phone modem regularly drops data. Serial transfer protocols such as XModem and even PPP have been designed to address this issue.

To increase transmission reliability, modes 3 and 4 can be set to transmit every packet twice. This potentially doubles the reliability of the system, but at the expense of throughput since every packet is sent twice. Both modes 3 and 4 will reject an identical packet if received within a one second window. This was done so that, in double-transmit mode, redundant data is discarded.

The Transmit Twice mode is further aided by extending the TinyOS message structure so that true redundancies are easily detected. Consider the data of our mode 3 packet again:

```
data=0C.00.00.00.48.65.6C.6C.6F.20.77.6F.72.6C.64.21.0D
.....H .e .l .l .o . .w .o .r .l .d .! .
```

The first four bytes have been reserved by modes 3 and 4. The first byte is always incremented for every original message sent. The next three bytes are (for now) always zero. It follows that redundant message area easily detected by examining the message data and ensuring that it does not perfectly match the previous message. It really is that simple.

Appendix C: Ideas for the future

The Mote as Network Modem can later be enhanced with more complex operating modes.

Consider a case where multiple channels are desired, such as where many systems share the same space but where system subgroups have no need to communicate with each other. Instead of all mode 3 & 4 messages being of type 80h, perhaps it would be beneficial to create a setting where the type could be anywhere from 0 to 255.

Another interesting idea is to create a setting where each module can be given a unique 7-digit identification number, similar to a telephone. If this number occupied the reserved bytes 1-3 of mode 3&4 packets, it would be possible for modules to “call” each other, “pick up” and establish a private point-to-point link for the duration of a conversation before disconnecting.