

A Mote for Local Information Broadcasts

The Mote-Enabled Bus Stop

V1.2 – September 29, 2005

Craig Pataky

Around six months ago, it was first mentioned that the motes could play an important role in navigating public transportation systems. We envisioned the strategic placement of motes at bus stops and other stations where a user could wirelessly interact with the device to receive useful information.

Since then, we discussed many possible applications for a local information server. Ideas such as passenger counting, on-time pick-up reliability, and behavior tracking were considered.

In the end, we chose the task of enabling a Mote to broadcast informative pre-recorded messages applicable to the bus station's immediate surroundings. Such information includes bus schedules, to be sure, but also includes directions for how to reach nearby shops, café's, and other establishments that might not be immediately visible from the station.

The scenario where this information is most useful is where a person approaches a bus stop and learns that the bus is not due for at least 15 minutes. The person could then walk to a nearby establishment to get something to eat, drink. Phone numbers might also be included with the directional information, so the person could conceivably call ahead and have an order ready for pick-up.

At a minimum, our Local Information Broadcast device must wirelessly transmit all parts of a message block while also providing some means to accept information updates. Though theoretically simple, it did require us to examine the mote's nonvolatile memory resources and devise a reliable storage scheme.

Of particular interest in this project is that we introduce a multipart wireless messaging scheme to our framework. Most messages using motes are atomic entities where each packet contains all the information necessary to make the packet relevant. In our information broadcast scenario, we must assemble several messages in order to provide the user with a coherent information block.

The Radio Message Interface

The software is designed to work hand-in-hand with modes 3 & 4 of the Mote Modem. This makes it easy for people to interact with the GPS without requiring specific knowledge of TinyOS messages.

Using a Mote Modem in mode 3 or 4, the device can be pinged and queried for power. The commands and responses are shown below.

<u>Query:</u>	<u>Response:</u>	<u>Use:</u>
PING	BUSSTOP_PONG	Determine if bus stop is in area.
BUSSTOP_POWER	BUSSTOP_MV=nnnn	Determine busstop battery condition, where 3000mV is typical.
BUSSTOP_POSITION	Lat/Long Data	See below.

Query the position of the bus stop by sending the string “BUSSTOP_POSITION” and the device will respond with position information (if available) in the following manner:

```
BUSSTOP_LAT=44.04589N  
BUSSTOP_LNG=123.07052W
```

The textual responses BUSSTOP_LAT and BUSSTOP_LNG represent degrees and fractional degrees, along with N/S or E/W indicators.

If the bus stop is operating in mode 1, the position information will be automatically broadcast about every ten seconds. Otherwise, the position data will be provided only if asked for.

This action assumes that the device has previously been configured with accurate GPS data. If the device has never been configured, the bus stop will not transmit its position under any circumstance.

Status Information on LEDs

The red LED flashes when data is transmitted over the radio.

The yellow LED flashes when data is received over the radio.

The green LED flashes when data is transmitted or received on USART0.

Multipart Messages

A TinyOS message can be up to 29 characters, which is not sufficient for the type of information we want our bus stop to provide the user. To achieve our goals, we will need to expand our simple messaging protocol with a multipart type.

A multipart message must contain in a unique originator ID, a message ID, how many parts comprise a total message, and the part number of the particular segment represented by a single TinyOS message. Fortunately, all this can be represented in a mote modem Mode 3 compatible message.

Consider the following multipart message segment:

```
MP0105007002=uo motelab[cr]
```

The format of this data is described as follows:

Field	# Of Bytes	Description
MP	2	All multipart message start with the characters MP
01	2	A number (in hex) that represents the originator of the msg.
05	2	A number (in hex) that uniquely identifies the multipart message from this originator. An originator can in fact be transmitting several multipart messages simultaneously, and this is a way to associate parts with messages.
007	3	How many parts make up this message, in hex.
002	3	The number of this particular part, in hex. The part number always starts at one, not zero.
=	1	A delimiter indicating that the part content is to follow.
uo motelab	10	Up to ten bytes are sent per part.
[cr]	1	The part always ends with a carriage return.

So, using this scheme, we must be aware of some limitations.

- 1) There can be no more than 255 (ffh) originators in proximity and that each originator can be responsible for no more than 255 (ffh) uniquely identified multipart messages.
- 2) A multipart message can contain no more than 4095 (fffh) parts. This limits the multipart message size to around 40K bytes.

Assembling a Complete Message from Multipart Components

The receiver must gather each part of the multipart before processing the full message. When a receiver gets any segment of a multipart, it can know the total number of parts by examining the appropriate field. The receiver must store the part and evaluate a table to see if all parts have been gathered, and also to throw away redundant parts if present. If all parts have been gathered, then the complete message can be evaluated.

The Serial Interface on USART0

By default, the mote's USART0 has been programmed to operate at 9600 baud, N81. You can connect the device to a PC by using the Crossbow MIB510 programming board and a program such as HyperTerminal.

Alternatively, a TTL connection is available on J21 or J22. Attach to pin 28 as transmit data (data from the mote to an external device). This connection is meant for embedding the unit into a larger system, as you cannot connect directly to a PC using this method—a TTL to RS-232 adapter is required. If you really want to connect to a PC and not use the MIB510, adapters are readily available from HVW Technologies (www.HVWtech.com) for only \$10.

When the mote first powers-up, the following sign-on message is displayed:

```
APP: BUS STOP MODULE
September 27 2005
13:36:51
```

The sign-on message is the best opportunity to verify that the serial connection has been made correctly, and that the module has been programmed with the correct version of software.

For every message sent over the radio, the device will send the same information over USART0. This makes it easy to connect devices that do not have wireless capability, and also allows for diagnosing the system should a need arise.

Command and Control

The bus stop 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

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
ATRECORD .. INITIATES MESSAGE RECORD MODE
ATSTOP .. STOPS RECORDING MODE
ATDELETE .. DELETES THE RECORDED MESSAGE
ATPRINT .. PRINTS THE RECORDED MESSAGE
ATLAT=nnnnnnn .. SETS LATITUDE
ATLONG=nnnnnnn .. SETS LONGITUDE
ATMODEn .. SETS OPERATING MODE 0 or 1
ATLEDn .. SETS LED MODE 1=ENABLED 0=DISABLE
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
```

Baud rate commands are self-explanatory. The others we’ll discuss here.

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.

Use ATMODE to set the operating mode. Mode 0 is the default mode, where the device will not send position messages out the radio unless queried. In Mode 1, the bus stop will regularly transmit its position information, if available. This message provides an easy way for passive listening devices to determine their proximity to the bus stop.

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

```
SETTINGS:
MODE=1
BAUD=9600
ENABLE LEDS=1
OK
```

Use ATRECORD and press return to initiate recording mode. When record mode begins, the operator can type or send an ASCII message to the mote for recording. In this mode, the mote will use XON/XOFF flow control to prevent the loss of data. To exit recording mode, type ATSTOP followed by a carriage return. Maximum message size is currently set at about 1.4K

Command ATDELETE erases the recorded message from memory.

View the recorded message by using the ATPRINT command. This is a good time to review the message and check for errors.

Set position information using ATLAT= and ATLONG= to enter the exact strings to be transmitted when the mote provides positioning data. No format checks are performed, so it is up to the operator to ensure that the information is valid and accurate. One advantage of this is that lat and long needn't be GPS coordinates if some other representation is more useful in some conditions, such as indoors.

Command ATSAVE is used to save the settings into flash and exit command mode. The unit will automatically reset and the new settings will take effect.

Command ATEXTIT is used to exit command mode without saving the settings into flash. The unit will automatically reset and the old settings will be restored. Note that this does not apply to a newly recorded message. Recorded messages take effect the moment they are entered.

Appendix A: Common Considerations of Multipart Messaging

Lost packets

A lost packet represents a hole in the data.

Our array will have one or more "holes" represented by a part number of zero. The receiver then must either dump the entire table, or perhaps could wait for holes to be filled. This is context-specific behavior. Consider a transmitting mote, such as the bus stop, that provides the same data over and over again. In this case, it would be useful to keep the old table and just wait for the holes to eventually be filled. On the other hand, if the same multipart message is not likely to be repeated, then the best option is to dump the entire table.

A packet of a different multipart slipping in while receiving another

This should be handled in a context specific manner. It may be desirable to ignore the interleaved multipart message, or it might be worthwhile to dump the one the receiver was working on and start with this new one, or it might be desirable to begin constructing another table in addition to the original. It all depends on the aim of the device.

To listen to more than one speaker at a time is possible, but possibly confusing and definitely inelegant.

A good analogy for this is trying to listen to someone in a crowded restaurant. Most of us will tune out the surrounding conversations and focus only on the people we're with. If the conversation is boring, we'll listen to what is going on around us and perhaps focus on what someone else is saying. Also consider if someone shouts "FIRE" then we're bound to drop everything and listen only to that person. The point is that we're selective about our source, and we pay attention to only one speaker at a time. The same should be true for our motes.

Ok, so I'm waiting to fill a hole created by a lost packet. How long should I wait for this

Well, as a rule of thumb, if you don't see another packet with the same originator and message ID within about ten seconds, then I would say dump the whole multipart message.

Otherwise, it is up to you. I'd say that as long as you keep getting indicators that the hole will eventually be filled (and you're still interested in the data) then I'd keep waiting.

Using this method, how long would it take to assemble a 2k message

Assuming that 2k is 2048 bytes, and our multipart message payload is 10 bytes per, then

it would take at least 205 messages to represent the data we want. TinyOS will transmit a message about once every 50ms. So $205 * 50 = 10250$ ms, or 10.25 seconds. This is a best case scenario, where no packets were dropped. Because radio communications are never 100% reliable, we can easily assume at least two passes at the data are needed to fill holes. This brings our time up to 20.5 seconds. So, being conservative, it would be wise to budget 30 seconds to fully communicate a 2k multipart message.