

Localization Software Setup Manual

Jeffrey Vander Stoep

June 12, 2009

1 Sensor Network Implementation

The sensor network implementation has been achieved via the Crossbow Imote2 platform [1]. The Imote2 is designed around the low-power PXA271 XScale microcontroller with 802.15.4 radio and surface mount 2.4 GHz antenna. We have installed the Linux Kernel and the drivers necessary for radio communication. In this section we discuss the details of implementing localization onto a network of Imote2 wireless sensors.

1.1 Communicating with the network

In order to study the localizing sensor network, communication from the network to a destination outside of the network is required both to issue commands to the network and to receive location estimates from localizing sensor nodes. This section focuses on connecting to the sensor network, issuing commands, and receiving location information from the network.

A special node referred to as the *base* node will be the communication link between the sensor network and a personal computer. The base node will be connected to a PC using USB and communicating through USB via a socket. The base node's duties include connecting to the GUI, keeping track of routing information, and forwarding network commands from the GUI to the sensor network.

1.2 Routing

In large network deployments, nodes will likely be out of radio range of the base node. Thus message routing through other nodes is essential for control and data acquisition. This particular network requires that each node be able to route a message to the base, and the base be able to route commands to each node. We have tried to accomplish two things with our routing protocols. First the protocol needs to be light weight and simple. Second the routing messages should never trigger other messages as this could cause a cascade of nodes trying to transmit simultaneously.

1.2.1 Routing information to the base node

Routing of information to the base node is accomplished through a distributed Bellman-Ford algorithm [2]. At set, unsynchronized, intervals each node broadcasts their fewest number of hops to the base. In

this setup, we are assuming that all nodes have the same radio range. Thus if node a receives a message from node b , then node b is within node a 's range.

Table 1 illustrates the Bellman-Ford routing table. The *Last Heard* column is incremented at each timer interval. If the last heard column goes above a certain threshold, then the address is considered dead and removed from the routing table. Each time that a particular address is heard broadcasting its routing message, its last heard count is reset to zero.

Table 1: Bellman-Ford routing table.

Address	Hops to base	Next hop	Last Heard
1001	2	1003	2
1002	3	1001	4
1005	4	1002	1
.	.	.	.
.	.	.	.
.	.	.	.

1.2.2 Routing commands out from the base node

Routing messages from the base node out to members of the network makes use of the Bellman-Ford routing protocol described in Section 1.2.1. Once a node has a valid path to the base node, it sends a packet to the base node via that path. As the message traverses the path, each intermitent node appends its address to the packet. The resulting message can be seen in Table 2. Thus when the message reaches the base node, it contains the list of hops taken. This process is repeated periodically to ensure that the base node has an up to date path to each member of the network. Messages from the base node out to members of the network follow a similar pattern, but with the hop sequence reversed. As the message makes its way through the network, each node removes its address from the end of the message. This message structure can be seen in Table 3.

Table 2: Message sent to base node to establish route

Source node addr	First hop addr	Second hop addr	...
------------------	----------------	-----------------	-----

Table 3: Routing commands from the base node to members of the network

Message data	Source node addr	Last hop addr	...	First Hop
--------------	------------------	---------------	-----	-----------

2 Linux Installation and Setup of the Imote2 network.

Configuring the Imote2 network for localization is a complex process and requires five major steps. Loading the Linux kernel, configuring the radio, setting up the cross compiler, loading the localization program, and defining node IDs and the base node. The installation requires

1. Windows PC with a parallel port
2. Linux PC
3. Intel JTAG cable
4. Parallel port cable
5. IIB2400 debugger board
6. Imote2.

To facilitate a more manageable installation and setup process, the following directions have been divided into numbered steps. Figure 1 exhibits the Imote2 programming setup.

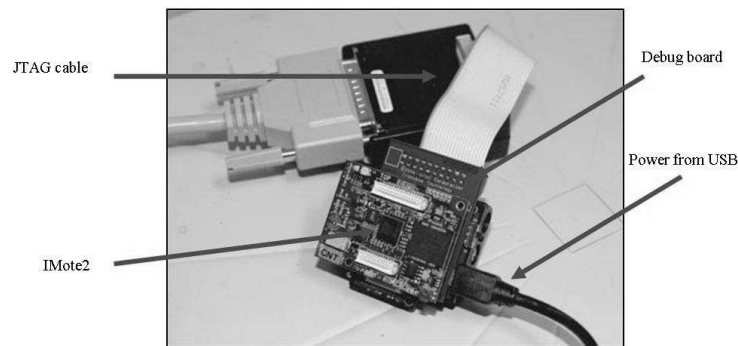


Figure 1: Imote2, JTAG cable, and IIB2400 debugger board connected to the USB cable

2.1 Installing the Linux Kernel

This section requires a Windows PC with parallel port.

1. Download and install the XFlash C++ compiler from the Intel web page. A 30 day evaluation is available at <http://software.intel.com/en-us/intel-sdp-home/>.
2. In BIOS, change the parallel port settings to enable ECP and set the range to 378.
3. In Windows' device manager, open the LPT1 port and check *Enable legacy Plug and Play detection*.
4. Before plugging the Imote2 into the Windows PC, go to <http://www.ftdichip.com/Drivers/D2XX.htm> and download the appropriate driver. Unzipping this file should result in the folder *CDM 2.04.16 WHQL Certified*. Create a folder *C:\Program Files\FTDI Drivers* and copy the unzipped folder into this directory.
5. Connect the IIB2400 debugger board to the Windows PC via the parallel port cable and USB cable. Wait for the Windows PC to install the necessary software and then restart when indicated.

6. Download the bootloader *blob*, the Linux Kernel *zImage*, and system file *fs.jffs2* from http://ubi.cs.washington.edu/files/imote2/images/backup_7_29_2008/Console.winGadget/images/. Place them in the xflash directory C:\Program Files\Intel\SDT2.0\xflash\
7. Connect the IIB2400 to the parallel port of the Windows PC through the JTAG cable and directly to the Windows machine through the USB cable.
8. Power on the Imote2 by pressing the reset button.
9. In the file C:\Program Files\Intel\SDT2.0\xflash\boards.ini append the following:

```
[INTELMOTE2]
standardflash=GP_J3_K3_K18_8
monaddr=0x01000000
scanchain=PXA27X
algoaddr=0x5c000000
baseaddr=0x00000000
```

10. Create C:\Program Files\Intel\SDT2.0\xflash\intelmote2.xdb.
11. Open the Window's command prompt. Navigate to C:\Program Files\Intel\SDT2.0\xflash\ and execute the following commands:

```
xflash.exe -p intelmote2 -tt "Intel(R) JTAG Cable" blob -offset 0x00000000
xflash.exe -p intelmote2 -tt "Intel(R) JTAG Cable" zImage -offset 0x00040000
xflash.exe -p intelmote2 -tt "Intel(R) JTAG Cable" fs.jffs2 -offset 0x00240000
```

You should see something similar to

```
Memory Successfully Burned.
File successfully burned.
```

12. Log into the Imote2 via Windows' Hyperterminal. The proper comm port can be decided by testing the available options. Set the following properties for the connection:

```
Bits per second: 115200
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None
```

13. Login ID is *root*, password *rootme*.
14. Type *ifconfig* to obtain the correct IP address.
15. An alternative method for connecting is to use SSH. Remove the Imote2 from the debugger board, and connect it directly to the USB cable. Windows will automatically detect the *Linux USB Ethernet/RNDIS Gadget*. Choose to install from a specific location. Include C:\Program Files\FTDI Drivers\CDM 2.04.06 WHQL Certified\ in the search. Complete the installation.

16. In Windows' Network connections, right click on the *Linux USB Ethernet/RNDIS Gadget* connection and select properties. In the *General* tab, select *Internet Protocol (TCP/IP)* and select *properties*. Select *Use the following IP address:* and specify the IP as *192.168.99.100*.
17. Using Cygwin or SSH Secure Shell Client or another unix emulator, login using the command *ssh root@192.168.99.101* (or the appropriate address). The Password is *rootme*.

2.2 Configuring the radio

This section covers the steps needed to configure the Imote2's radio.

1. Go to <https://sites.google.com/site/imote2linux/setting-up-radio> and download the attachment *tos_mac.ko*. Copy to the *imote2* in directory */lib/modules/2.6.14_r1.1/kernel/drivers/tosmac/* overwriting the previous version.
2. On the Imote2, add *tos_mac* to the */etc/modules* file. Reboot the Imote2.
3. Create a character device by executing *mknod /dev/tosmac c 240 0* on the Imote2.

2.3 Seting up the cross compiler

This section requires a Linux PC.

1. On a Linux machine, download the ARM GCC compiler from http://ubi.cs.washington.edu/files/imote2/toolchain/imote2_toolchain_3.4.3_binutils.tgz.
2. Uncompress to the root folder of the Linux PC.
3. Add `export PATH=$PATH:/usr/local/arm/3.4.3/bin/` to `~/.bashrc`.

2.4 Compiling and loading the localization program onto the Imote2

The section covers the steps required to download the localization code, compile it, and load it onto the imote2. Do the following steps on the Linux PC with where the cross-compiler was set up in Section 2.3.

1. First the code must be downloaded. It is currently available at <http://code.google.com/p/imote2-localization/source/browse/trunk>.
2. In the terminal, navigate to the directory with the localization code. Type *make*. This will result in the building of two files, *handler* and *handler_base*. The following directions are split up into directions for the base node and directions for anchors/localizing nodes
3. Base Node

- (a) Copy *local_base.sh* and *handler_base* to */root/* on the imote. This can be accomplished as follows. Transfer the file to a Windows PC that is connected to the imote. In a Unix like environment (Cygwin, SSH Shell, etc.) navigate to the directory on the pc with *local_base.sh* and *handler_base* and enter the following command:

```
scp local_base.sh root@192.168.99.101
scp handler_base root@192.168.99.101/
```

- (b) Allow each of these files to be executable. SSH onto the imote and in the */root/* directory, type:

```
chmod +x local_base.sh
chmod +x handler_base
```

- (c) OPTIONAL but recommended to allow the base node run its software upon bootup. Add the following symbolic link to */etc/init.d/*.

```
cd /etc/init.d
ln -s /root/local_base.sh
```

This will allow the base node and routing programs to restart if it should crash.

4. Anchor/Localizing node

- (a) Copy *local.sh* and *handler* to */root/* on the imote. This can be accomplished as follows. Transfer the file to a Windows PC that is connected to the imote. In a Unix like environment (Cygwin, SSH Shell, etc.) navigate to the directory on the pc with *local.sh* and *handler* and enter the following command:

```
scp local.sh root@192.168.99.101
scp handler root@192.168.99.101/
```

- (b) Allow each of these files to be executable. SSH onto the imote and in the */root/* directory, type:

```
chmod +x local.sh
chmod +x handler
```

- (c) Add the following symbolic link to */etc/init.d/*.

```
cd /etc/init.d
ln -s /root/local.sh
```

This will allow the node's localization and routing programs to restart if it should crash.

5. Transfer *handler* to the Windows PC and in a Unix like environment (Cygwin, SSH Shell, etc.) navigate to the directory with *handler* and enter the following command.

```
scp handler root@192.168.99.101/
```

6. SSH into the imote and make sure that the file is executable by entering.

```
chmod +x handler
```

2.5 Setting up the Network and defining node IDs

In this section we cover giving the node its 16 bit address and defining the base node. This will enable the systems routing tables to be set up.

1. Create of file in */root* called *router.conf*.
2. In *router.conf* the first line defines the nodes address and should be a four character hexadecimal number (e.g 111A). Any value from 0000 to *FFFF* may be selected. The second line is 0 if the node is the base node and 255 otherwise. There can only be one node defined as the base node for the network to function properly.

References

- [1] *Crossbow Technology: Imote2*, 2008. <http://www.xbow.com/Products/iMote2.aspx>.
- [2] D. Bertsekas and R. Gallager. *Data networks*. 1992. Prentice Hall, 1992.