Project Spartan’s Bluetooth Guide

Getting the BlueZ Bluetooth Stack for the BBB

- First make sure you have internet connection for your BeagleBone Black, whether it be a WiFi dongle or ethernet

- SSH into your BeagleBone Black

- Now apt-get the BlueZ stack: apt-get install bluez

- The utility tools are also useful for debugging bluetooth: apt-get install bluez-utils

- Cross compiling is a pain for bluetooth on the BBB and I would recommend native compiling your code before trying to go down this road
  - here is a good place to start if you want to try though: http://wiki.beyondlogic.org/index.php?title=Cross_Compiling_BlueZ_Bluetooth_tools_for_ARM#Installing
    - the guide is a little out of date so you might want to try and install newer versions of the dependencies
    - the important parts to this guide are the flags given to the make commands and configure files
    - good luck!

Connecting Bluetooth Dongle

- Connect your Bluetooth dongle via USB

- Typing “hciconfig” will let you know if the device is recognized by the beaglebone
  - If not then you may have to reboot with it plugged in
- If the above fails check that you have the right firmware for your device by “dmesg | grep Bluetooth”

- There should be a line stating firmware failed to load followed by the name of a particular file with a .fw extension

- Typing “lsusb” provides information about the manufacturer adding the ‘-v’ tag makes it more verbose

- This a help resource to find the firmware files you are looking for “http://wireless.kernel.org/en/users/Drivers/”

Compiling with GCC

- make sure when you build your program that you put -lbluetooth at the end of your command it won’t work if it’s in middle

A Basic RFCOMM Server in BlueZ in C

Bluetooth communication using BlueZ is very similar to socket programming in C since many of the same functions are used. Figure 1, is a reference for the flow of Bluetooth from device searching, connecting and closing the connection.

- The first thing that must be done is to create a socket.
  - create a integer value to hold the file descriptor for the socket
  - call the socket function “int sock = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
    - the first parameter indicates that we’re using the bluetooth domain
    - the second makes the socket a two way connection oriented socket
    - the last parameter sets the bluetooth protocol we are using which is RFCOMM
  - the socket function will return the file descriptor for the newly created socket which will be stored in the created integer value.

- Next the socket address must be set and bound to the socket
  - create an instance of “struct sockaddr_rc”
  - set the rc_family value to AF_BLUETOOTH which set it the bluetooth addressing family
  - set the rc_bdaddr to *BDADDR_ANY which specifies that you will use any local bluetooth adapter
  - set the rc_channel to any number between 1 and 30 (note: if the port number is already in use it will not bind

    - call the bind function with the parameters: your socket, a reference to the socket address struct (sockaddr_rc) and the length of your socket address structure which can be obtained using sizeof.
        - this will bind your socket to the specified port number

- Next call the listen function with your socket and the number of queued connections which you can set to 1
    - example "listen(socket, 1);"
    - this will put the server into listening mode and will make it sleep until a connection is attempted by another device

- After a connection has been made with an external device the program should call accept
    - the function takes the parameters: the server socket, a socket address structure and the socket address size
    - it will return file descriptor for the client's socket

- Finally the server can begin reading bytes from the client using the read function
    - this takes the client socket file descriptor a character buffer of set size and the size of the character buffer
    - the buffer can be a character array of "reasonable" size, ours was only a kilobyte but I'm sure it can be larger than that

- For sending information use the write / send function
    - write(client, char_buffer, buffer_length)
    - send(client, char_buffer, buffer_length, flags)

- When done transmitting calling close with the server file descriptor will close the socket and end the connection

- Depending on what devices that the program is connecting to you may want to allow for connections just in case the device loses the connection

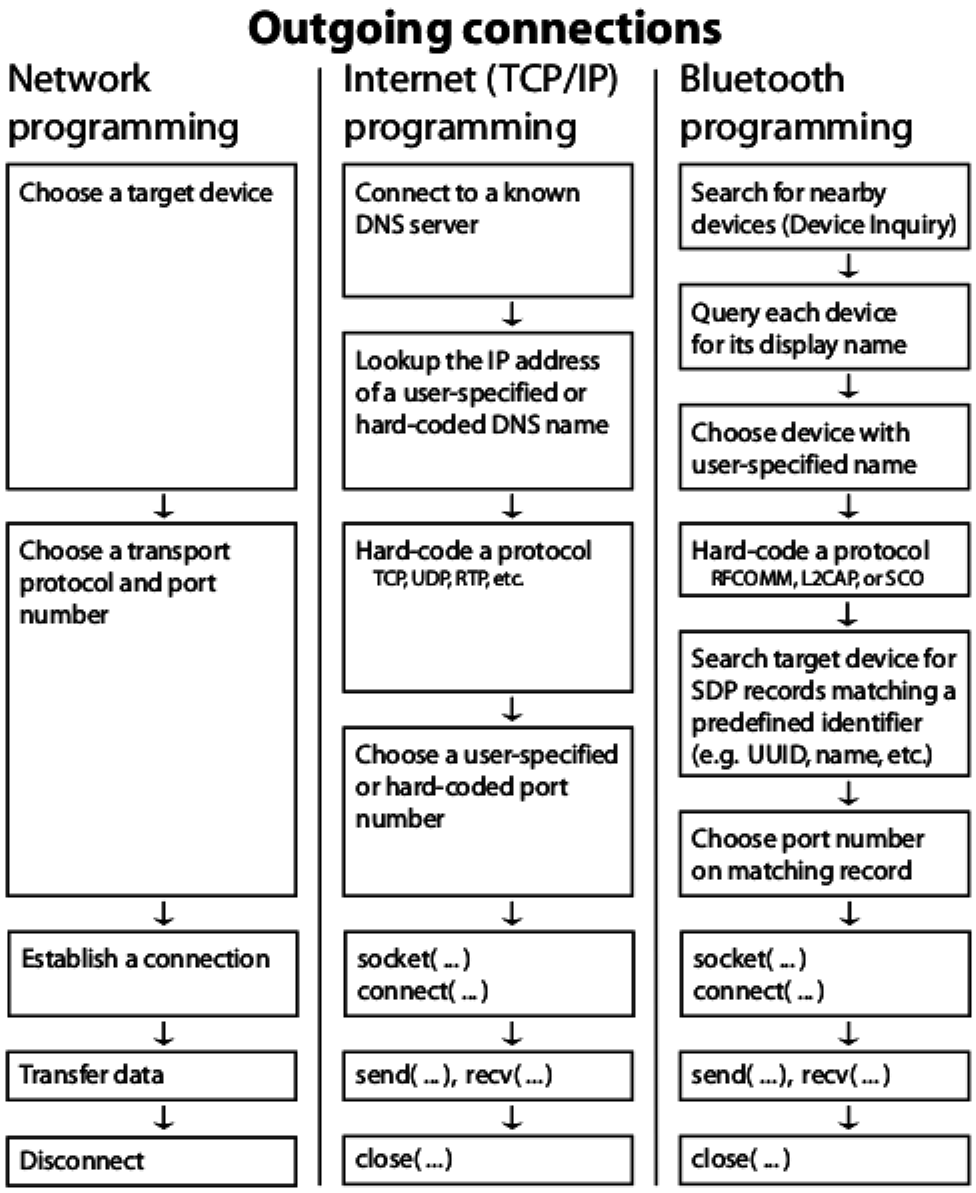
Outgoing connections
Network programming
Internet (TCP/IP) programming
Bluetooth programming
Choose a target device
Choose a transport protocol and port number
Establish a connection
Transfer data
Disconnect
Connect to a known DNS server
Lookup the IP address of a user-specified or hard-coded DNS name
Hard-code a protocol TCP, UDP, RTP, etc.
Choose a user-specified or hard-coded port number
socket(...) connect(...)
send(...), recv(...)
close(...)
Search for nearby devices (Device Inquiry)
Query each device for its display name
Choose device with user-specified name
Hard-code a protocol RFCOMM, L2CAP, or SCO
Search target device for SDP records matching a predefined identifier (e.g. UUID, name, etc.)
Choose port number on matching record
socket(...) connect(...)
send(...), recv(...)
close(...)

Figure 1: Steps required in TCP and RFCOMM socket communication [1]

<u>Sources</u>

[1] Albert S. Huang and Larry Rudolph, “Introduction” in *Bluetooth Essentials for Programmers,* 1st ed, New York, USA: Cambridge, 2007, ch 1, pp 1-39

[2] Albert S. Huang and Larry Rudolph, “C Programming with GNU/Linux” in *Bluetooth Essentials for Programmers,* 1st ed, New York, USA: Cambridge, 2007, ch 3, pp 67-109

[3] Albert S. Huang, 2005-2008, Bluetooth programming in C with BlueZ [Online], Available: http://people.csail.mit.edu/albert/bluez-intro/c404.html