NXT Bluetooth

Communication

- **Elizabeth Mabrey** -

(This document is mainly derived from the RobotC online documentation, but with substantial amount of modification done by Elizabeth in order to be more fit for our young technical readers)

Content

3 Common Wireless Communication

(note: this document is partially from the robotc documentation)

- Bluetooth
- Wi-Fi
- InfraRed

Bluetooth	Wi-Fi	IR
Is the name of ancient Danish king Harold Bluetooth who unified Denmark and Norway in the 10th century	Short for Wireless Fidelity	Short for Infra-Red. *infrared* is a wave of light that in the area beyond the visible part of the color spectrum.
Is a telecommunications industry specification allows wireless connection via radio frequency (RF)	same	InfraRed. While it is invisible to human eye infrared is often used to enhance visibility when using night vision devices.
Slow	Much faster	
Various protocol. Not all Bluetooth devices are compatible.	Standardized protocol. All Wi-Fi devices must follow a particular protocol. If it says, e.g., 802.11g, it will be compatible with another 802.11g device.	All IR devices are compatible
Range: For short-range up to approx. 10 meters from peer to peer.	**Range:** For long range of up to 45 meters away from a "hot spot" (slang for a Wi-Fi networking node).	**Range** - IR communications span very short distances, usually not more than 5 feet. Unlike Wi-Fi and Bluetooth technologies, IR network signals cannot penetrate walls or other obstructions and work only in the direct "line of sight."
Performance: Operate in the 2.45 Gigahertz frequency range. Other devices currently operate in this frequency range as well, including cordless phones, baby monitors, and garage-door openers, etc. Have a maximum transmission rate of only 1 Mbps — up to 2	**Performance:** Also operates in the 2.4 GHz band The latest specification, 802.11g, offers much more powerful and capable of reaching data transmission rates approaching 54Mbps.	**Performance** - Infrared technology used in local networks exists in three different forms: IrDA-SIR (slow speed) infrared supporting data rates up to 115 Kbps IrDA-MIR (medium speed) infrared supporting data rates up to 1.15 Mbps

Mbps in the second generation of the technology —		IrDA-FIR (fast speed) infrared supporting data rates up to 4 Mbps
Bluetooth	Wi-Fi	IR
Connections can be point-to-point or point-to-multipoint	Connections can be in any combination: PTP, PTM, MTM, MTP. Allow much more elaborated network infrastructure.	Only PTP.
Provides a wireless, point-to-point, "personal area network" for PDAs, notebooks, printers, mobile phones, audio components, and other devices.	Provide wireless, high-speed access to the Internet or a local area network.	Like Bluetooth.
Bluetooth products make it easy to connect various electronic devices to each other in a relative small distance, and included in a smaller network infrastructure.	Wi-Fi products, such as 802.11-based, were designed as a replacement for — or extension of — the wired LAN - can be implemented for a much larger network infrastructure.	Not meant for network, but just one device to another directly.
Inexpensive	Can be Expensive.	Very Inexpensive
Other than facilitate pin login, there is really no security protocols.	Facilitate high security protocols	No security measure

.

The following table compares the available Bluetooth power classes:

Class	**Maximum Power**	**Operating Range**
Class 1	100mW (20dBm)	100 meters
Class 2	2.5mW (4dBm)	10 meters
Class 3	1mW (0dBm)	1 meter

The actual range for each power class may vary depending upon environmental factors where the Bluetooth device is used. Class 3 devices have a very limited range and not very common, hence, they will be ignored for the rest of this discussion

So which power class should you choose for your new Bluetooth product? The two most important question that should be asked here are: "over what distance do I need my Bluetooth devices to operate?" and "what is the power class of the other Bluetooth device I want to communicate with?" Here are the two important pieces of information that you need to understand:

- If you wish to communicate over the **100m** range, you will need a **class 1** Bluetooth device at **both** ends.
- If you wish to communicate over the **10m** range, you can have a **class 1 or class 2** device at both ends.

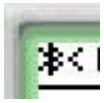
  Many people make the mistake of believing that they can extend the range of their class 2 device to 100m by purchasing a class 1 device for the other end. This is simply not true. Consider two people standing 100m apart, if person A yells loud enough for their voice to travel over 100m, person B will be able to hear what person A is saying, but if when person B replies they only yell loud enough for their voice to travel over 10m, person A will obviously not hear the response.

CONNECTING TWO NXTS VIA BLUETOOTH USING THE NXT'S USER INTERFACE

In NXT Bluetooth configuration, you can only connect up to 3 subordinate devices, we called them Worker from now on. The major one which initiates the connection will be called the “master”.

Manual Setup:

1. Turn on Bluetooth interface.
   - Go thru the NXT menu to select Bluetooth.
   - Turn on the visibility.

   This symbol should show up at the top left corner of the screen. Notice you may see “<”, not “<>”. “<” means this device is ready to connect, but not yet connected to any other BT devices.

2. Search connections.
   - Select the “Search”. (The NXT simply broadcasts a "who's available" message over BT and collects a list of all the devices that replied.
   - Wait until a device is found and then select to “Connect”. This may take up to 30 seconds before it determines there is none found.

- You can “Connect” up to 3 devices.
- Then it will prompt you to assign the port 1, 2, or 3. Select, e.g. 2, to connect.
- You may be prompted to enter password. This will depend on the settings on your NXT.
- Note that you need to enter the password too at the Worker device in order to complete establishing a single connection.
- Once the connection is made, the status ICON will change to "<>" on the top leftmost is shown on the connected devices.

3. To see all connections.
   a. Go the the menu to select “Connections”.
   b. A list of connected devices should show up on the screen.

4. To disconnect. Easy…. Select “Disconnect”.

PROGRAMMING WITH BLUETOOTH

All messages will be queued. Only maximum 10 58-bytes messages in the queue. This should be sufficient enough for most of your robotics challenge.

Major BT functions and variables:

Common functions:

cCmdMessageWriteToBluetooth(…): writes a message.
cCmdMessageGetSize(…): get the size of the first message in a mailbox containing a queue of received messages.
cCmdMessageRead(…): removes the first message from a mailbox queue and copies it to a user buffer.

Connection handler:

Short nBTCurrentStreamIndex = 0 to 3

TFileIOResult : Communication status - common ones to use include:

ioRsltSuccess
ioRsltInProgress
ioRsltBTConnectFail
ioRsltCommPending
ioRsltEmptyMailbox

Error checking

You should always perform check the returned value to determine the success/failure of the function.

DETAILED SPECIFICATION FOR THE COMMON FUNCTIONS:

➲➲

TFileIOResult cCmdMessageWriteToBluetooth(short nQueueID, ubyte nXmitBuffer[], int nSizeOfMessage**)**

Synopsis: to send a BT message to another connected NXT. This sends the message up to 58 bytes in length. When messages are received over BT by a NXT they are automatically added to the end of one of the 10 message or mailbox queues.

Parameters:
short nQueueID: mailbox number nQueueID on the targeted NXT.

ubyte nXmitBuffer[]: contains the message in bytes array.

int nSizeOfMessage : the length of the message.

Return:
TFileIOResult - communication status.

➲➲

Int cCmdMessageGetSize(short nQueueID**)**
Synopsis: to determine whether any messages have been received

Parameters:
int nQueueID: mailbox number nQueueID on the targeted NXT.

Return:
int size : Returns message size in bytes length. A positive return value indicates that a message was received.
.

➲➲

TFileIOResult cCmdMessageRead(ubyte nRcvBuffer[], int nSizeOfMessage, short nQueueID)
Synopsis: "to retrieve the first message from the specified mailbox

Parameters:
ubyte nRcvBuffer[ ] : received message is copied into this ubyte array.
Int nSizeOfMessage: dictates only the first nSizeOfMessage bytes of the message are copied.
Short nQueueID : mailbox number on the targeted NXT.

Return:
Int size : Returns message size in bytes length. A positive return value indicates that a message was received.

Other NXT Bluetooth Access Functions

Note that the RobotC is still in its infancy and has to work under the technical restriction from Bluecore and the NXT firmware. You will need to learn how to be resourceful in how to obtain detailed specification of all lower detailed access functions and system variables.

bBTBusy

btConnect(nPort, sFriendlyName);
Attempts to connect BT device with specified name (sFriendlyName) on port nPort. This NXT will be the master and the other the Worker.

btDisconnect(nPort);
Disconnects the BT connection on port nPort.

btDisconnectAll();
Disconnects all existing BT connections.

btFactoryReset();
Resets the NXT BT to the factory configuration. All connections will be dropped. The contacts list will be emptied.

btRemoveDevice(sFriendlyName);
Removes a device with the specified name (sFriendlyName) from the contacts lists

btSearch();
Begins a search for BT devices and adds new entries to the contacts lists. The search can take a long time (15 seconds).

btStopSearch();
Stops a search that is in progress.

getFirstDevice(nHandle);
getNextDevice(nHandle);
These two functions are used to scan through the entries in the contacts list. Get first device will initiate the search from the beginning of the list and get will find the next valid entry in the list.

getDeviceAddr(nHandle, sAddr);
getDeviceClass(nHandle, sCOD);
getDeviceName(nHandle, sName);
getDeviceStatus(nHandle, nStatus);
These functions are used to retrieve information about a particular entry in the contacts list.

getPortName(nPort, sPortName);
Gets the name (sPortName) of the device connected on port nPort

readLinkQuality(nQuality);
Reads the BT link quality (future)

requestLinkQuality();
Requests the current BT link quality

setBluetoothOff();
setBluetoothOn();
Turns Bluetooth ON or OFF on the NXT.

setBluetoothVisibility(true | false);

Sets whether the NXT BT is visible or invisible to searches from other Bluetooth devices. Invisible status means that the NXT will not respond to “search for devices” requests from other BT devices. However, if a device already knows the name/address of this NXT, then it can make a connection to this NXT.

setDefaultPIN(sPIN);

Sets a default PIN to use for Bluetooth connections (future)

setFriendlyName(sFriendlyName);

Sets the sFriendlyName that a NXT will be known by. The friendly names is normally displayed on the top status line of the NXT LCD.

transferFile(nPort, sFileName);

Transfers the file sFileName from this NXT to the NXT connected to port nPort.

nBluetoothCmdStatus

Gets the status/progress of the last BT 'command'

nBluetoothStatus

Gets the current BT status. See the enum for more details. Contains a bit map that indicates whether BT is on/off, visible/invisible and connected/not connected. These status bits are used to update the BT icon on the top line status display on the NXT.

nLastBTCommand

Gets the last BT command issued.

BLUETOOTH OVERVIEW AND PROFILES

Bluetooth (BT) is an industry standard short-distance (up to 10 meters) wireless communications protocol operating at 2.4 GHz. It can optionally use a higher power transmission to achieve distances up to 100 meters. The NXT utilizes the 10 meter option.

BT includes not only the low-level radio transmission (at 2.5 GHz) but also several higher layer message protocols (or profiles) designed for different applications. There are over 25 different BT profiles currently defined. Some of the more popular profiles include:

- The **Serial Port Profile (SPP**) is used to provide wireless emulation of a conventional RS-232 serial communications cable. This is the only protocol supported on the NXT.
- The **Human Interface Device (HID)** protocol is used for communication on wireless keyboards and mice. It is also used on some game controllers like the Nintendo Wii or SONY Playstation 3.
- The **HeadSet Profile** (HSP) is used to connect wireless headsets to devices like cellphones.

In order to connect two devices via BT, the devices must not only support BT but also support the type of profile that will be used for the connection. The NXT only supports the SPP so that it cannot, for example, directly connect to a Nintendo or Sony game controller.

Connecting via Bluetooth, Pairing

Each BT device is unique identified by a unique 12 hexadecimal digit address. It's a little awkward to refer to devices with this address so the BT protocols include a "friendly name" that is a more conventional 15-character string. The default friendly name for a NXT brick is "NXT" but you can modify this using the NXT's on-brick user interface, or from the ROBOTC IDE, or directly from within a ROBOTC program.

Before you can connect two devices via BT they must be "paired". The pairing process exchanges messages between the two devices where they share their BT address, their friendly names and their supported profiles. The devices confirm that they support the same profiles and that the (optional) password matches. Once two devices have been paired then you can make subsequent connections using the friendly name only and the devices remember the passwords so that they don't have to be re-entered.

There are a lot of low-level messages used in establishing a paired connection. Fortunately, this is hidden from the user on the NXT with a simple user interface via the user interface on the NXT.

After the devices are paired, i.e. connection established, the found device will be added to the NXT's internal table of paired devices, i.e. the "My Contacts" list on the NXT. Next time you want to make a connection you can select the device from the "My Contacts" and avoid the 30-second search.

Possible problems for two previously paired devices not to connect:

1. one device is no longer powered on.
2. the devices have got out of sync on the pairing status where one device has "lost" the paired status. If this happens, try removing both devices from the "My Contacts" list on each NXT and restart the connection using the "Search" function.

Restrictions (or you may call them features) for using Bluetooth

- A device can be either a subordinate or a master, but not both.
- A single master can (optionally) make connections to multiple Workers.
- A subordinate can only connect to a single master.
- The BT protocol is asymmetric. At its lowest level:
  - A master can transmit over radio at any time.
  - A Worker only transmits in response to a request from a master.

Master vs Worker Device

(*in technical world, it is called Slave. But, to be political correct, we call it the slave device as the "Worker" device*)

One end of a BT connection is the master device and the other end is the worker device. The master device generates the clocking signal used for the BT connection and the Worker device derives its clock from the received radio signal. That why a worker cannot have connections to other masters as it would then need to support multiple clock sources!

During the initial connection setup, the two devices negotiate the maximum bandwidth that they will use and how often they are listening for traffic. This allows the devices to use a low-power mode (i.e. the radios are only enabled part of the time) at the expense of lower bandwidth.

Bluecore Implementation

BT implementation on the NXT use a "Bluecore" chip from CSR.

Bluecore is a self-contained implementation of BT that manages the BT hardware and protocols. It has a few limitations that are common to many other BT hardware implementations:

- The "search" function requires 100% of the BT resources. When a search is in progress, then no other BT activity can take place on the NXT.
- The module can be in either "command" or "data" mode. They are mutually exclusive states.
  - In command mode, housekeeping functions related to the BT protocol are being performed -- this includes things like searching, pairing, making connections and disconnection.
  - Data mode is used to transfer data between two devices over the BTX wireless link.
- Bluecore supports a single "master" device connecting to up to three 'worker' devices at one time. However, it can only communicate data with one device at a time; it could not simultaneously receive data from all three worker!

On the radio side, Bluecore can have three connections (or "streams") to different Worker devices. However, on the other side -- i.e. the connection from Bluecore to the NXT CPU, only a single connection is possible. It is this implementation that leads to the above restrictions. So, for example:

- If you wanted to add a second Worker connection to a NXT, the Bluecore module would be switched into "command" mode. This would interrupt "data" traffic from the original connection. The appropriate commands would be performed to set up the connection and then Bluecore would be switched back to "data" mode.
- In data mode, Bluecore is only connection to one of the two streams/connections at a time. If data from the disconnected stream is received, it is discarded.
- Bluecore can be switched from "data mode on stream 1" to "data mode on stream 2". This is done by switching to "command" mode, sending the "switch stream" command and then switching back to "data" mode; this takes over 100 milliseconds.

MORE DETAILS ON BLUETOOTH WITH LEGO MINDSTORMS

NXT-G Bluetooth Messaging

The NXT-G firmware has built a message passing system on top of the NXT Bluectore implementation. It works as follows:

- Messages can contain up to 58 bytes of data. The restriction is that firmware has a fixed 64-byte format for message buffers and there is six bytes of overhead in the structure.
- Single master can connect to up to three Workers.
- Messages can be sent to one of ten queues. The queue number is one of the overhead bytes.
- When a message is sent from the master, it is put on a queue of outgoing BT messages.
    - Messages are transmitted from the queue in a FIFO basis whenever the previous queue item is completed.
    - The firmware looks at the stream ID -- ie. which of the three possible connections -- before transmitting a message.
    - If the stream ID does not match the currently "active" stream, then the active stream is switched to this stream by the 100+ millisecond process described above.
- Worker devices cannot use the above process! The Worker does not know whether it's stream is the master's "active" stream. The implementation for messageing from the Worker is as follows:
    - An outoging message from the Worker is always added to one of the ten outgoing message queues. One queue for each of the ten possible "mailboxes".
    - The message sits in the outoging queue until the Worker receives a "poll for message from mailbox N" message from the master.
    - In response to the "poll for message" the Worker responds with either "the selected mailbox is empty" or it sends the first message found in the queue.

If there is only a single Worker, then the above process does not incur the 100+ msec delays of switching 'active' streams on the master. It still incurs a delay while the Worker waits for a polling request from the master.

ROBOTC Bluetooth Messaging

The ROBOTC BT messaging has been optimized for a single Worker connecton on the master. This allows for significantly less latency on the BT message link. Workers can immediately transmit messages without having to wait for a polling request from the master. ROBOTC also allows for multiple Worker support, but a description of this is beyond the scope of this tutorial. This optimization gives a performance improvement of three to 20 times over other architecture that support – at a significantly reduced performance level – multiple simultaneous Workers.

ROBOTC measured performance to send a BT message, receive it at far end, process it, generate a reply and receive it at original device is about 36 such transactions per second. Other implementations (e.g. NXT-G) typically support about 13 transactions per second.

ROBOTC allows full duplex operation over a single BT stream. Measured performance indicates each end of the stream can autonomously send 250 messages per second. The half-duplex implementation is limited to 13.

NXT Bluetooth Overview

The NXT contains an integrated Bluetooth (BT) communications link. BT is an industry standard short-distance (up to 10 meters) wireless communications protocol.

- BT communications can be used to connect to the PC instead of the USB link. You can do all functions available over the USB link via BT. Except that NXT firmware can only be upgraded via USB. The USB communications is far faster than BT but requires a tethered link. BT, because it is wireless, is un-tethered.
- BT communications can be used to connect multiple NXTs together.
- BT communications can also be used to connect the NXT to another BT device – like a cell phone – as long as the other device communicates with the protocol specifications in the NXT BT Development Kit.

BT communications can operate in either a master or Worker mode. The NXT BT device can only be in one mode or the other at any one time.

- When connected to the PC via RobotC, the NXT is in Worker mode and the PC is the master.
- When two NXTs are connected via BT, one is the master device and one is the Worker device. A NXT in master mode can support up to three Workers. A NXT in Worker mode only supports a single BT connection.

The NXT GUI (Graphical User Interface) provides manual capabilities to manage the BT configuration on the NXT. The set of functions described here provide the ability to do all of the GUI capabilities, and more, within a user application program. You can set up and tear down connections, search for devices, remove items from the contacts list and many more functions.

The NXT firmware manages its BT communications through two lists: a connected device array and a contacts list.

- The connected device array has four elements that represent the different possible connections. Index zero (i.e port zero) in the array is always used for a Worker NXT to contain information about its single connection. Ports 1 to 3 are used for a NXT in master mode to contain information about its three possible connections to other BT devices.
- The contacts list contains a list of BT devices that have been previously found by a “search” command. Since this is a historical list, the devices in this list may not all be currently accessible. The NXT remembers in this list devices that have previously connected to it and will automatically allow subsequent connection attempts by these devices without manual password entry – the previously used password will automatically be used.

While a NXT in master mode can connect to three Worker devices / NXTs simultaneously, it can only communicate with one at a time. This means that a Worker NXT/device cannot simply send a message to the master NXT; the master may be in communications mode with one of the other two possible devices and will not be listening to this particular Worker. So, the Worker must ‘buffer’ its message and wait for a polling request from the master asking for a buffer message. The BT protocol allows for simultaneous transmission over all links; this one at a time restriction is a function of the NXT firmware and the BT chip that it uses.

When compared with USB communications, BT is fairly slow. RobotC typically takes 28 milliseconds to send a message to another device and get a reply. This is almost three times faster than most other NXT firmware solutions.

BT is implemented on the NXT via a special purpose “Bluecore” hardware module. The main CPU on the NXT communicates with the Bluecore module via an internal messaging scheme. The main CPU sends a message to the Bluecore and then waits for a response. This means that the functions described below generally do not complete in a single transaction. Your program needs to request the action and then continually check the status until the transaction has completed and the result communicated back to the NXT CPU.

The Bluecore hardware can be in one of two modes: processing BT commands or data transmission mode. It cannot be in both modes at a time. This means, for example, that if in the middle of program execution that is communicating with one BT device you make a connection to another BT device during the time the connection is being set up the communications with the first device is stalled.

BT devices have a unique 7-character (256 possible values per character) address. This can be fairly cumbersome so the BT specifications have included a "friendly name" (i.e. a standard character string of printable characters) that can be used to refer to a device.

SELECTED NXT BLUETOOTH TECHNICAL DETAILS

This section provides additional details for the technically inclined on the ROBOTC Bluetooth implementation and the NXT BT hardware. It highlights some of the areas that need careful treatment by software to implement a robust BT messaging scheme.

One of the hardware modules within the NXT is a self-contained Bluetooth (BT) module. It is called Bluecore (BC) by the module manufacturer. The NXT CPU communicates with the BC module via a high-speed serial link. The BC module can have up to three simultaneous connections to other NXTs.

Bluecore Command vs. Data Mode

The NXT CPU and BC need to operate in either CMD or DATA mode. BC cannot operate in both modes simultaneously.

In command mode, the NXT CPU is communicating with BC to perform "housekeeping" functions. Data transmitted between the two is interpreted as commands (and replies) to the BC chip; typical commands are connect/disconnect far end devices, enable/disable visibility, search for devices, etc.

Generally, all the command mode activities are performed at the beginning of a BT session. Once a connection has been set up, data "stream" mode is entered. When a session is complete, command mode is re-entered to drop (i.e. disconnect) the connection.

In data mode, BC simply transfers the data between the RF link and the serial link on the NXT CPU.

ROBOTC provides interfaces that enable user program access to most of the BC commands. For example, you can set up a connection to another NXT from within a user's program. Programs should perform all "commands" requests first before beginning data transmission (i.e. sending messages) activity.

The NXT firmware smoothly manages the transition between command and data mode. There are a couple of digital I/O pins between the CPU and BC that are used to perform the transition which typically takes several milliseconds.

When in command mode, any data received over the RF link is simply discarded by BC. Similarly the NXT CPU discards any requests for transmission of data packets over the RF link.

One Master Can Connect to Three Worker Devices

BC allows a single "master" NXT to connect to up to three "Worker" NXTs. The master provides the overall timing for the RF link used and the Workers derive their RF timing/synchronization from the master. You cannot mix master and Worker devices on a single NXT. This limits the size of a network of BT connected NXTs to four devices – one master and three Workers.

BC communicates with the NXT's CPU over a single high-speed serial link. When a NXT is in master mode, the CPU tells BC which one of the three possible Worker ports (streams) should be currently active. BC simply transfers all data on the serial link to/from the active stream. Any data received from either of the two inactive streams is simply discarded by BC.

There is a command for the NXT CPU to tell BC to make a different Worker port (stream) the active stream. This involves switching from Command to Data mode and sending a few requests to BC. It takes over 100 milliseconds to perform the active stream switch.

Most NXT Bluetooth Implementations Use Half-Duplex Operation

One implication of an architecture that supports simultaneous multiple Worker connections is that the Worker devices cannot autonomously send messages to the master device since they have no knowledge of whether their connection is currently the active stream on the master. The architecture requires a half-duplex protocol where the master device has to send a polling message to each Worker device requesting any data that is buffered waiting for such a request.

The result is that the RF link is only used in a half-duplex mode. It is only transmitting in one direction at a time.

The incremental transaction delays in simultaneously handling multiple Workers – i.e. 100 to 300 milliseconds per transaction – make it impractical for high-performance real-time operation. It's simply too slow.

FURTHER INVESTIGATION BY BENCHMARKING:

SIMPLE BENCHMARK WITH NXT-G:

1. Sending one messages from master and reading one message from worker takes 76? milliseconds.
2. Sending three messages from master and reading one message from worker takes 82? milliseconds.
3. Sending NO messages from master and reading one message from worker takes 63? milliseconds.
4. Sending one messages from master and reading NO message from worker takes 4 milliseconds.

Try to benchmark your NXT:

5. Sending one fixed-sized message from master and reading one message from worker takes ? milliseconds.
6. Sending three messages from master and reading one message from worker takes ? milliseconds.
7. Sending NO messages from master and reading one message from worker takes ? milliseconds.
8. Sending one messages from master and reading NO message from worker takes ? milliseconds.

Configuration used:

- Only use "numbers".
- Only use a pair of NXTs; i.e. the BT master is only connected to a single worker NXT.

Reminder:The NXT Bluetooth hardware allows up to three worker RF connections but only one at a time can be "streamed" to the NXT CPU. If data is received over the radio channel from a worker while the worker is not in "streaming" mode, then the data is lost; this is handled by the NXT firmware which implements a BT protocol that only allows workers to generate BT messages in response to a message request from the master.

When there are multiple workers connected to a single BT master, the master sequentially connects the single "stream" to each of the workers in a round robin fashion. This is quite a slow process -- in reading the firmware source code there is at least one delay of 100 or more milliseconds "to prevent the Bluecore BT module from crashing". With two workers, it would seem that you'd need to add at least another 200 milliseconds to the above read times.