Evaluation of Bluetooth communication: Simulation and experiments

Technical report 02/03, Department of Computer Science, University of Copenhagen

by **Martin Leopold**

Department of Computer Science, University of Copenhagen
Spring 2002

Contents

Abstract

In this report we review some of the key aspects in Bluetooth point to point connections by simulation and by experiments to help evaluate the usefulness of Bluetooth in sensor network applications. We present a model of the Bluetooth device discovery (inquiry) and show the correctness by experimentation. We show how Bluetooth communication and device discovery degrades with distance. We show that Bluetooth features are fairly robust within the ranges of the devices. We present our experiences working with the Bluehoc simulator and Bluetooth Bluez host stack.

1 Introduction

Bluetooth is a promising technology for short range wireless communication. It was first conceived by Ericsson in 1994 as a cable replacement technology; it has now become a candidate of choice for communication layer of sensor networks and smart tag devices.

The report was done as a part of the Manatee [MT] project in the Distlab research group at DIKU[1] and financed by apparater.dk [AP]. The Manatee project focuses on the use of smart tags in the context of monitoring applications.

This report has 3 major goals:

- To gather expertise using Bluetooth simulation tools and actual devices for the Manatee project.
- To evaluate the usefulness of the Bluehoc simulator for further use.
- To gather measurements of some Bluetooth characteristics: Throughput and duration of inquiry in point to point connection.

We make the following contributions:

- We describe a model of the inquiry procedure. This model allows us to predict and explain the performance of inquiry operations.
- We produce an evaluation of the Bluehoc simulator.
- We present experimental results using two Bluetooth devices.

The report is structured in the following way: in section 2 we discuss and introduce the Bluetooth technology. In section 3 we take a look at Bluez (the Bluetooth stack included in the current Linux kernel series). In section 4 we look at a few available Bluetooth products and summarize their features and in section 6 we execute a set of experiments using these devices. In section 5 we look at the Bluehoc network simulator and in section 5.2 simulate a set of experiments.

2 Bluetooth

Bluetooth operates in the 2.4GHz royalty free ISM band. Because it is free the ISM band is polluted by all kinds of "noise". This requires Bluetooth to be very robust. As a result Bluetooth uses a Frequency Hopping Spread Spectrum (FHSS) scheme by dividing the ISM band into 79 logical channels[2]. Bluetooth defines a slot as a 625 μs interval in which a node either sends or receives data. After each slot the frequency of the communicating devices is changed according to a frequency hopping sequence. The sequences are unique for each device and are designed not to show repetitive patterns over short periods of time. Each device is assigned a 48 bit address known as the Bluetooth device address

[1] Department of Computer Science, University of Copenhagen

[2] For Europe, Japan and the US, 23 for France

(BD_ADDR) resembling the IEEE 802 MAC address the hopping sequences are derived from this address. The current position in the hopping sequence is denoted as the phase.

Changing the frequency takes time—the channel synthesizer requires a certain interval to settle at the new frequency. This means that some fraction of time is wasted changing frequency, for this reason multi slot packages of 3 and 5 slots are defined to better utilize the bandwidth (while increasing the likelihood of error). The packets come in two flavors: with forward error correction (FEC) named DM (Data Medium rate) packet and without FEC named DH (Data High rate).

The Bluetooth specification[BT02] defines 3 power classes based on the transmission power of the radio: 100 mW (class 1), 2.5 mW (class 2), 1 mW (class 3) reaching respectively approximately 100 m, 20 m and 10 m. Currently the most common devices are power class 2 or 3.

The building block of a Bluetooth network is a piconet. A piconet is a collection of up to 8 devices. One device is denoted as master and the other as slaves. The slaves are following the same hopping sequence of the master. The master has network management responsibilities: channel bandwidth allocation, allowing slaves to enter power save mode, etc. Each slave is given an active 3 bit Active Member Address (AM_ADDR). There is no slave to slave communication. All messages are exchanged between master and a slave. Furthermore, a slave is only allowed to transmit to the master once the master has contacted it, i.e. the slots alternate between master send and send slave send. If a node is participating in more than one piconet it has to timeshare its radio for the hopping sequence of these two piconet—this is called a scatternet.

2.1 Bluetooth Layers

The lower layers of the Bluetooth stack is divided into the layers: Base Band (BB) and Link manager (LM). They are interfaced through the Host Control Interface (HCI).

The BB layer manages the medium access—the slot timing and allocation, synchronization of local and remote clocks, channel coding (whitening, FEC, etc.) and hop sequence selection. The baseband also manages the state machine required for the lower level signaling and responses such as connection establishment (paging) and device discovery (inquiry)—these functions are sometimes denoted the link controller [BRAY01][65].

The LM layer has two purposes: first, it translates the commands received from the upper layer (HCI) to the BB layer. Second, it manages the connections: attaching a node to a piconet, connection negotiation and link shutdown all goes on in the LM layer. [BRAY01][101].

The HCI layer provides standard interface between the lower timing sensitive layers and the higher computation intensive layers. Splitting the responsibilities in two makes good sense since the host (a pc, telephone, pda, etc.) has the processing power to handle the higher layers while it would be very time consuming for it to handle the μs timing of the lower layers. Providing a standard interface makes it easy to mix and match devices and host stacks from different vendors easy. The specification provides 3 transport layers for the HCI interface: RS232, USB, UART. The UART interface is distinguished from RS232 in that it leaves error controls to the UART controller. Most consumer devices allow access to devices through the standard HCI interface and provides a set of vendor specific commands. As a consequence the tuning parameters are rather limited. We can only tune the devices through the set of HCI commands.

The upper layers of the Bluetooth stack (the host part or host stack) include the layers Logical Link Control Protocol (l2cap) and Service Discovery Protocol (SDP).

L2cap provides data link layer functions, taking data from the applications and transports it over the lower layers either as connection oriented or as connection less. It handles packet segmentation and reassembly (SAR), Quality of Service (QoS) and it provides a

protocol multiplexing feature allowing a single ACL link to be used for several l2cap connections.

RFCOMM emulates all features of an RS-232 serial port over a Bluetooth link.

SDP is implemented to support the ad-hoc nature of Bluetooth networking. It allows a Bluetooth to discover which capabilities devices in the vicinity have—say a printer, modem or a telephone.

2.2 Inquiry and paging

The heart of Bluetooth is the mechanisms that makes it possible for two devices hopping on different frequencies to discover each other (inquiry) and to establish connection (paging) [BT02][p.105ff]. The techniques used for inquiry and paging are very similar. Here we only describe the mandatory inquiry procedure (details have been left out for the sake of brevity).

In order for two devices to discover each other, they must be in two complementary states *inquiry* and *inquiry scan* at the same time. The inquiring device continuously sends out "is anybody out there" messages hoping that these messages (known as ID packets) will collide with a device performing an inquiry scan. To conserve power a device wanting to be discovered usually enters inquiry scan periodically with period T_{inq_scan} and only for a short time known as the inquiry window $T_{w\_inq\_scan}$. During this period, the device listens for inquiry messages. Since the devices are hopping on different frequencies the inquiry procedure must maximize the chance of two devices "catching" each other. To do this the inquiring device follows a fast half-slot timing, sending messages on two frequencies in one slot and listening on those frequencies on the following two half-slot while the device in inquiry scan changes it phase very slowly—once every 2048 slots (1.28 s). The half-slot hopping is possible because the ID packets are small enough to allow the channel synthesizer to change frequency twice within one slot.

T_{inq_scan} is specified bye the Core Specification to be at most 2.56 s (4096 slots) [BT02][106] and by the "General Access Profile" to be "more than once in 2.56 s" [BTP02][32]. The vendors we have seen interpret this as $T_{inq\_scan} = 2048\ slots$ (1.28 s), which we will use in the following. $T_{w\_inq\_scan}$ specified by the Core Specification as "long enough to for 16 frequencies" [BT02][106], which because of the half-slot timing means 16 slots.

The inquiring device sends out a short packet (ID) and the inquiry scanning device responds with a frequency hop synchronization (FHS) packet containing among other things information about the devices hopping sequence and the device clock timing. In order to minimize collisions from responding devices, a device receiving an ID packet in inquiry scan will return to its previous state for a random number of slots (RAND) between 0 and 1023 slots before reentering inquiry scan. Upon the next received ID packet it will reply with an FHS packet. After the FHS packet has been returned, the inquiry scanning device will move its phase one forward and reenter inquiry scan again, meaning that it is likely to hear the next ID packet from the inquiring device, and the procedure with random back off starts over.

During inquiry the two devices follow a dedicated 32 frequency inquiry hop sequence. The inquiring device splits the sequence in two 16 slot (10 ms) parts: the A train and the B train. A single train must be repeated at least $N_{inquiry} = 256$ times before a new train is tried, so one try of a train is thus $T_{train} = 4096\ slots$ (2.56 s) long. The specification recommends a total inquiry period of 10.24 s to collect all responses, but inquiry can be aborted prior to this if say enough responses has been collected. The inquiring devices uses its own clock to determine the phase in the sequence and is thus random compared to any other device.

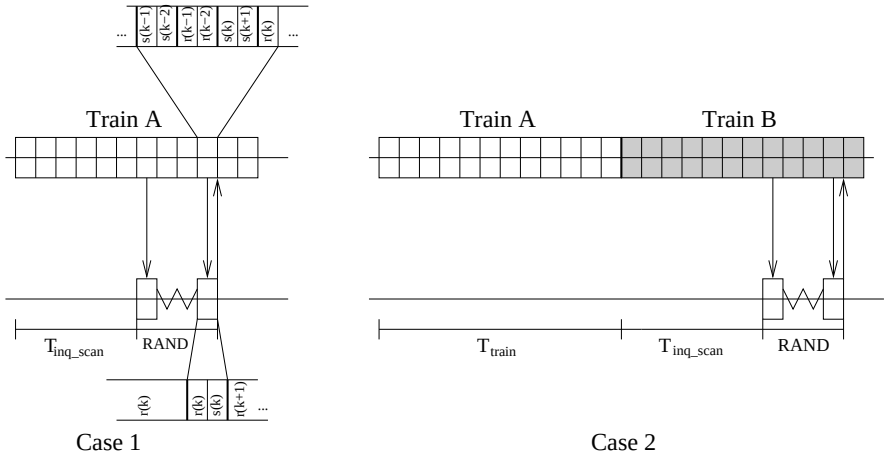

Figure 1 – *Inquiry with no errors. Since the inquirer splits the sequence in two, there are two possibilities of how long it is going to take to discover a device—even in an error free environment. In the magnification r denotes receiving and s denotes sending.*

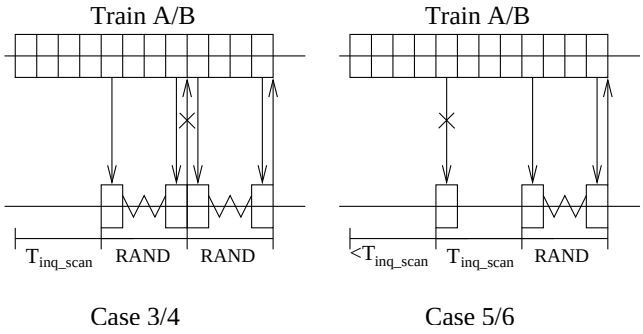

Figure 2 – *Case 3/4 - the reply FHS packet is lost requiring an additional random backoff. Case 5/6 - The inquiry ID packet is lost requiring T_{inq_scan} more to discover the device.*

2.3 Analysis of inquiry duration

The specification[BT02] does not present an analysis of the expected time of inquiry. Here we present an analysis, of the inquiry strategy described in the previous section, by presenting a set of cases each being a building block of the expected duration. This leads to a set of overlapping intervals (in table 1) in which we expect inquiry times to show up. We do not analyze the actual shape of actual graph. This would require a statistical analysis and it would require us to know whether RAND is uniformly distributed.

When the inquiring device starts sending ID packets on its A part of the 32 inquiry frequencies the remote device may be listening in either the A part or the B part: these are cases 1, 2 in figure 1.

The most probable error is that the reply FHS packet is lost. This is a much larger packet than the ID, so transferring an FHS packet is more error prone than transferring an ID packet. Losing an FHS packet can happen both when the inquiry scanning device is found in the A train or the B train: this gives us case 3 and 4 - figure 2.

The ID packet may be lost. If the inquiry scanning device wakes up during the first half of the train (length 2048) with time to spare for the random backoff it will get a second chance after $T_{inq\_scan} = 2048$. Furthermore if it is in the A train it will get more

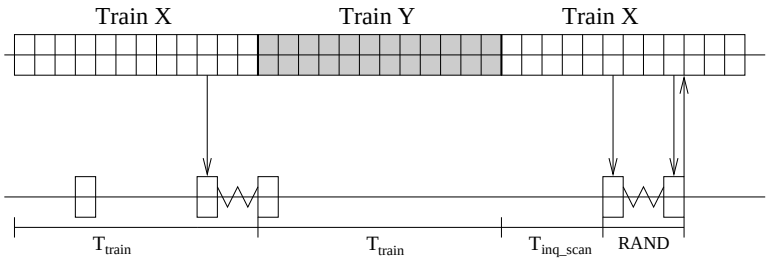

Figure 3 – *Case 7/8 - The first train is missed, the inquiry scanning devices have to wait until the next attempt of the same train to get a new chance. The figure shows the random backoff period overlapping with a train switch.*

Case	Best case	Worst case	no. slots
1	0	$T_{inq\_scan} + RAND$	0-3071
2	T_{train}	$T_{train} + T_{inq\_scan} + RAND$	4096-7167
3	0	$T_{inq\_scan} + 2 \cdot RAND$	0-4094
4	T_{train}	$T_{train} + T_{inq\_scan} + 2 \cdot RAND$	4100-8190
5	T_{inq_scan}	T_{train}	2048-4096
6	$T_{train} + T_{inq\_scan}$	$2 \cdot T_{train}$	6144-8192
7	$2 \cdot T_{train}$	$2 \cdot T_{train} + T_{inq\_scan} + RAND$	8192-11263
8	$3 \cdot T_{train}$	$3 \cdot T_{train} + T_{inq\_scan} + RAND$	12288-15359

Table 1 – *Analysis of the intervals in which the devices are expected to be discovered in each case - not counting the slots used by the package exchange.*

chances if 2 train switches are performed: corresponding to cases 5 and 6 - figure 2.

There are up to two chances to discover a device during a single train since $T_{train} = 2 \cdot T_{inq\_scan}$. If both fail, the next chance of discovering a device is during the next try of the same train. This might happen if the first attempt fails and the random backoff period of the second attempt overlaps with a train switch. If this happens for a device on the B train at least 2 repetitions of each train must be tried to give the missed device a second chance: this is cases 7 and 8 - figure 3.

The attempt in case 7 and 8 is prone to the same errors as cases 1-6, this means that we expect the pattern shown by cases 1-6 to repeat at intervals of length T_{train}

Depending on how we assign likelihood to each of these events we can predict how long the inquiry is going to take. The shape of the graph to be predicted will be based on a statistical analysis of how the probabilities of each even occurring will interact. Figure 4 illustrates where the concentration of device discoveries will be located. It is our guess that the vast majority of device discoveries will be found within case 1 and 2. Therefor we expect the average device discovery time to be slightly larger than $\frac{7167}{2} \cdot 625\ \mu s = 2.24\ s$ increased only by measurements in cases 3-7.

3 Bluez stack

The Bluez stack[BZ] is included in the Linux kernel 2.4 series. It contains an implementation of the upper layers of Bluetooth and provides a standard socket interface for the programmers. It has a preliminary SDP implementation and a port of the Axis SDP server is available. It is designed from the ground up to support multiple devices. It con-

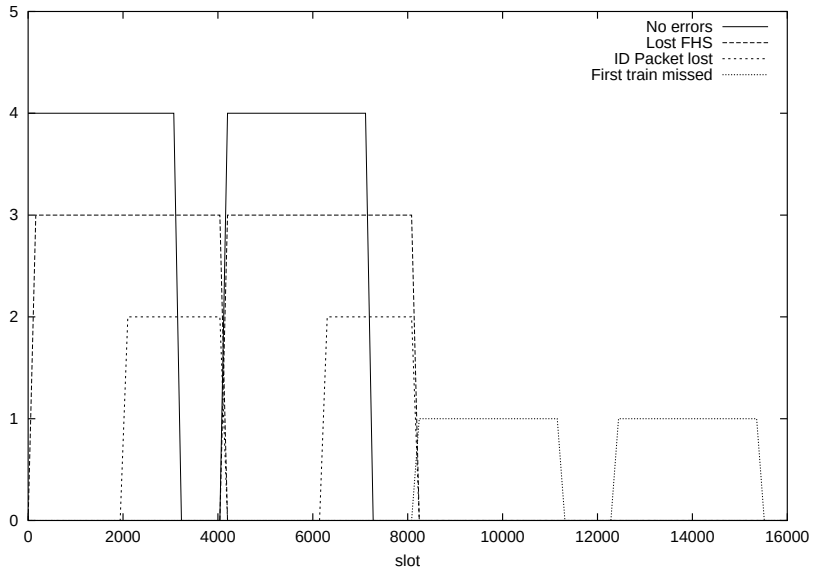

Figure 4 – *Illustration of the intervals in which device discovery is expected to happen. The figure illustrates the overlap of intervals an the repetitive pattern. The height is chosen arbitrarily, and the shape of the boxes are not representative of the shape of how the actual measurements will look.*

sists of a set of kernel modules and a userspace daemon; also included is a set of userspace programs and helpful utilities. Our experience is that this stack is more mature than the Axis OpenBT stack[AXIS].

3.1 Development using Bluez

The development of Bluez seems to be striding forward at a steady pace and the mailing lists regarding Bluez are quite active. The community surrounding Bluez has been very helpful in the development of this report with explanations and feedback.

The documentation is still limited, but good FAQs (Frequently Asked Questions) are available to get you started. Most tools resemble standard UN*X tools (hciconfig looks a lot like ifconfig) making it easy if you have some UN*X experience already. The most helpful documentation has been the example programs, but in general the API is sound and makes good sense.

4 Bluetooth Devices

In the early stages of this project we looked into a large number of Bluetooth devices, and decided to try out 2 of them. In this section we will describe those 2 devices from a functional point of view. We will focus on the relevant parameters for point to point connections.

Most consumer devices are produced partly or entirely by another company than the one selling it (often known as OEM[3]). Therefore the experiences gained here are likely to be applicable to other devices.

At this stage, the analysis of the devices is not deep. We simply connected devices and observe if the Bluez stack could register the device and perform simple tasks such as

[3]Original Equipment Manufacturer

inquiry and as well as a throughput test.

4.1 Brain Boxes

We have used Brain Boxes BL-500 PCMCIA cards. These cards contain the Cambridge (CSR) single chip Bluetooth solution BlueCore02. They have a built in antenna not visible from the outside. The PCMCIA form factor allows the cards to be concealed almost entirely inside the PC—only sticking out a few centimeters (presumably for the antenna). At the time of writing, the cards are only capable of point to point communication, but according to Brain Boxes support firmware upgrades are under development for point to multi point communication.

The CSR chips were one of the first Bluetooth producers on the market, and is used in many currently available Bluetooth add ons including: Brain Boxes USB/PCMCIA, 3Com USB/PCMCIA, Xircom Creditcard/RealPort2.

The cards contain an 16C950 UART with a maximum speed of 921600 bps. The Linux Kernel does not currently (version 2.4.17) support this speed. In order to run the serial port at full speed the kernel has to be patched. Also the *setserial* tool needs to be patched in order to obtain full speed. More than one patch is available, we decided to use the one Jean Tourrilhes from HP has written. He has a website explaining the details.

http://www.hpl.hp.com/personal/Jean_Tourrilhes/bt/Brainboxes.html

With this patch the Bluez stack initializes and recognizes the cards, the throughput seems to work with this card.

4.2 Bluefrog Development Kit

This is a Silicon Wave based kit using Reselec rbm3rc modules. It is based on the Silicon Waves SiW 1501 radio modem and SiW 1601 or SiW 1602 Link Controller and the Base-band controller is implemented using a Hitachi H8S-2238 micro controller. The device is power class 2 featuring point to point connections.

Our devices are as far as we can tell based on the module rbm3rc-u. The devices are located in an external box 15 cm by 10 cm with a few status diodes. They are connected to the PC via RS-232 or USB. The development kits have external antenna about 5 cm high. The device has the MTU of the ACL packets set to 60 bytes, while this is not an error it is certainly inefficient. Each ACL packet contain a 4 B header meaning that close to 7% of ACL communication will be headers. Compared to the 128 B MTU of the Brain Boxes card (see table 4.3) this seems peculiar.

The devices register without problems with Bluez and the connection test were successful.

4.3 Summary

The device features are summarized as:

Device	Interface	class	Mode	ACL MTU
Bluefrog	RS-232/USB	2	p2p	60
Brain Boxes BL-500	PCMCIA	2	p2p	128

We use *hciconfig -a* to obtain the following versions:

Device	HCI Ver	HCI Rev	LMP Ver	LMP Subver
Bluefrog	0x1	0x0	0x1	0x0
Brain Boxes BL-500	0x1	0x73	0x1	0x73

5 Simulations

In order to evaluate the usefulness of the Bluehoc simulator in terms of stability and configuration. We designed and executed a set of experiments.

5.1 Network Simulator and Bluehoc

We will be using NS-2b8a[NS2] with the Bluehoc 3.0 extension made by Apurva Kumar and Rajeev Gupta of IBM India[BHSITE]. Bluehoc extends the NS functionality with an implementation of the Bluetooth including the Baseband layer and the Link Manager. Bluehoc is implemented as a set of NS patches and C++ classes extending the NS hierarchy. Meaning that the TCP, and traffic generators implementation provided by NS are available over the Bluehoc Bluetooth simulation.

At the time of writing IBM has 4 releases of extensions for NS-2:

- Bluehoc 1.0 - a piconet simulator for NS-2.1b6
- Bluehoc 2.0 - a piconet simulator for NS-2.1b7a
- Bluehoc 3.0 - a piconet simulator for NS-2.1b8a
- Bluescat 0.6 - a preliminary release of Bluehoc with scatternet support. The NS version is unclear.

Bluehoc uses a free space model described in [KG01] based on (see appendix B):

1. Physical layer properties like modulation, FEC/CRC used etc.
2. Channel properties affecting interference.
3. Spatial properties like distance between interferer and receiver, topology etc.

This free space model is used to calculate static probabilities of packet loss for each packet type. For ID packets (issued in inquiry) the likelihood of packet loss is assumed to be 0. Rajeev Gupta of IBM confirmed [RG02] that this was done on purpose since the size of the ID packet is so small (see appendix C). Within a 20 m range (maximum of Bluehoc), this should be a reasonable assumption. Or tests with actual devices confirmed this assumption.

Prior to starting a connection Bluehoc calculates and requests QoS parameters based on the parameters given by the user. All parameters for the QoS request are read from the tcl script and entered into a global array *appFlowSpec* in the function *findAppNames* from the file *bt-host.cc*. For each connection *appFlowSpec* describes the QoS fields from the Specification [BT02][299], the *MTU* for connection establishment and additionally *input_qlen*, and *loss_sensitivity*. These parameters seem to be passed on to the NS traffic generator overwriting default values.

The MTU and *loss_sensitivity* parameters are used to select packet type. Based on *loss_sensitivity* Bluehoc either selects DH (0) or DM (1) packets. With the packet type and the rate Bluehoc tries to schedule the packet a priori, if a schedule is not possible the connection is rejected.

From the traces it seems that Bluehoc chooses an upper bound on l2cap packet as 1000 bytes (as no l2cap packets larger than this is sent), which differs from the default of 672 bytes [BT02][298]. Specifying packet sizes larger than 1000 bytes results in transmission of several 1000 bytes sized packets.

Default values for $T_{inq\_scan} = 8192$ and $T_{w\_inq\_scan} = 36$ are set in the file *globals.h*. We change these values to match the values of the devices: $T_{inq\_scan} = 4096$, $T_{w\_inq\_scan} = 16$.

The output from Bluehoc is called BT traces, these files contain l2cap signaling and reports the delay of each successful l2cap transmission. The successful packet delivery is printed with the packet size and the in-air transmission time. With these numbers we can calculate the throughput for each l2cap packet.

5.2 Bluehoc problems

Working with Bluehoc was not easy. During our tests, we encountered many problems and glitches. The documentation is very limited and much of is outdated—the most recent information is a FAQ [BHFAQ], and the mailing list available from the Bluehoc web site [BHSITE]. While the list has been very help full, the community surrounding Bluehoc does not seem to be very large. Some of the problems we faced are listed here, more details can be found in our notes regarding the use of Bluehoc (see appendix A):

- The installation procedures are misleading. In the documentation two procedures are listed, one doesn't work - the procedure named "IF YOU DO NOT HAVE NS INSTALLED".
- The output of Bluehoc is intermixed with output from NS making automated data processing fail since the format is corrupted.
- The documentation on how to configure the application is limited to say the least. The best way to figure out how to configure the system seems to be the code, but this being a fairly complex program, the code of the NS/Bluehoc system is very large and navigating through it takes time. In the following we will refer to the code when needed as it is our only source of information.
- The output from Bluehoc during connection establishment is undocumented. Our interpretation is included in the notes on Bluehoc (see appendix A).
- The parameters used for connection negotiation overwrite the parameters for the traffic generator, making it impossible to specify one set of parameters for the traffic generator and one for the connection negotiation.

5.3 Inquiry

During inquiry procedure Bluehoc only treats the FHS packet as error prone, while the ID packet has perfect reception. This means that the simulations should show a sensitiveness to the distance. We define the inquiry to be the interval from the inquiring device starts to the time when the inquiring device receives the FHS packet from the discovered device. The devices are set to start inquiry and inquiry scan at time $t = 0$, so the duration of the inquiry can be read from the simulated time. We create a set of tcl script files with two nodes with increasing distance from 0 m to 18 m in 1 m steps (all distances above 18 m seem to fail producing invalid output). Bluehoc crashes with a segmentation fault during the transmission part of the connection, that is after inquiry for distances above 14 m. Since this simulation is only concerned with inquiry this not a problem

Since the random back off timer changes the duration of the inquiry, we run each experiment 2500 times.

This gives two results

- for 0 m to 15 m the FHS packet is received after 1.38 s for all 2500 runs.
- 16 m to 18 all 2500 runs receives the FHS packet after 1.53 s

The difference is expected to reflect whether the first FHS reply is lost and reply is sent.

All runs of the inquiry procedure came out with exactly the same result which suggests that the back off scheme is broken. This seems fairly odd since it is implemented and relies on the randomizing features of NS. Since NS was restarted on each run it is possible that the randomizer uses a fixed seed instead of a random number for example system clock—this could be done to give reproducible results.

Slots	Data Medium rate (DM)	Data high rate (DH)
1	134 $\frac{kB}{s}$ (21 B)	198 $\frac{kB}{s}$ (31 B)
3	399 $\frac{kB}{s}$ (125 B)	598 $\frac{kB}{s}$ (187 B)
5	492 $\frac{kB}{s}$ (2000 B)	740 $\frac{kB}{s}$ (2000 B)

Table 2 – *Maximum rate accepted by Bluehoc, l2cap packet size in parenthesis. The size is found by simply increasing it until Bluehoc rejects the connection.*

5.4 Throughput

For the throughput simulations we want to saturate the available bandwidth and measure how many packets get through. This would have been done easily with an application sending as packet as fast as the link can consume them with no retransmission and sink at the other end. Unfortunately such a traffic generator is not provided with NS/Bluehoc. Instead we choose the constant bit rate generator—CBR. The most convenient way to measure the bandwidth with this generator would have been to oversaturate the link and measure how much gets through at the baseband level ignoring what goes on at the higher protocol levels. The parameters of the NS CBR traffic generator is *packetSize* and *rate*. Unfortunately Bluehoc rejects oversaturated scenarios as impossible.

We only set the traffic generator of one of the nodes (the master) leaving the other unset, this should make the unset node act only as a sink. The specification dictates that if no data is available to send a NULL packet is to be sent [BT02][p. 79], thus we setup an asymmetric connection with data packet (DM/DH) in one direction and NULL packets in the other.

5.4.1 Variable distance

The simulation is carried out by creating a set of tcl scripts for each packet length. Within each set, the files only differ in the distance between devices increasing 1 m at a time. The simulation time is set to 100 s.

By default Bluehoc does not support passing parameters for the CBR traffic generator. Therefore we patch *bt-host.cc* to set the parameters read from the tcl script files (see appendix D). There is no clean way of setting the *loss_sensitivity* parameter, so we set it in bt-host.cc, recompile NS and run the test.

Table 2 shows the maximum rates that Bluehoc accepted along with the packet sizes used.

The average throughput for each packet transfered is calculated from the output, and plotted as a function of distance (see figure 5). Each measurement (not shown in the figure) varies wildly, the distance from the maximum to the minimum increases from about $\frac{1}{3}$ of the average at 0 m to several factors at 10 m (see table 3).

The Bluehoc simulator crashes with a *segmentation fault* in most cases with distances above 12 m. The graphs in this section indicate that very few simulations survived beyond 12 m.

The throughput graphs all seem to start decline suddenly at 5 m for DH packet and at 6 m for DM packets. The suddenness of the change looks suspicious and might suggest that the free space model gives too little penalty to distances less than 5 m and 6 m respectively.

The medium rate connections show significantly less impact as a result of increased distance than the high rate connections. Equally the single slot connections show more noise robustness than the multi slot connections.

The average simulated throughput in table 3 is approximately one half of the theoretical maximum. In the next section we'll see how the numbers compare to real mea-

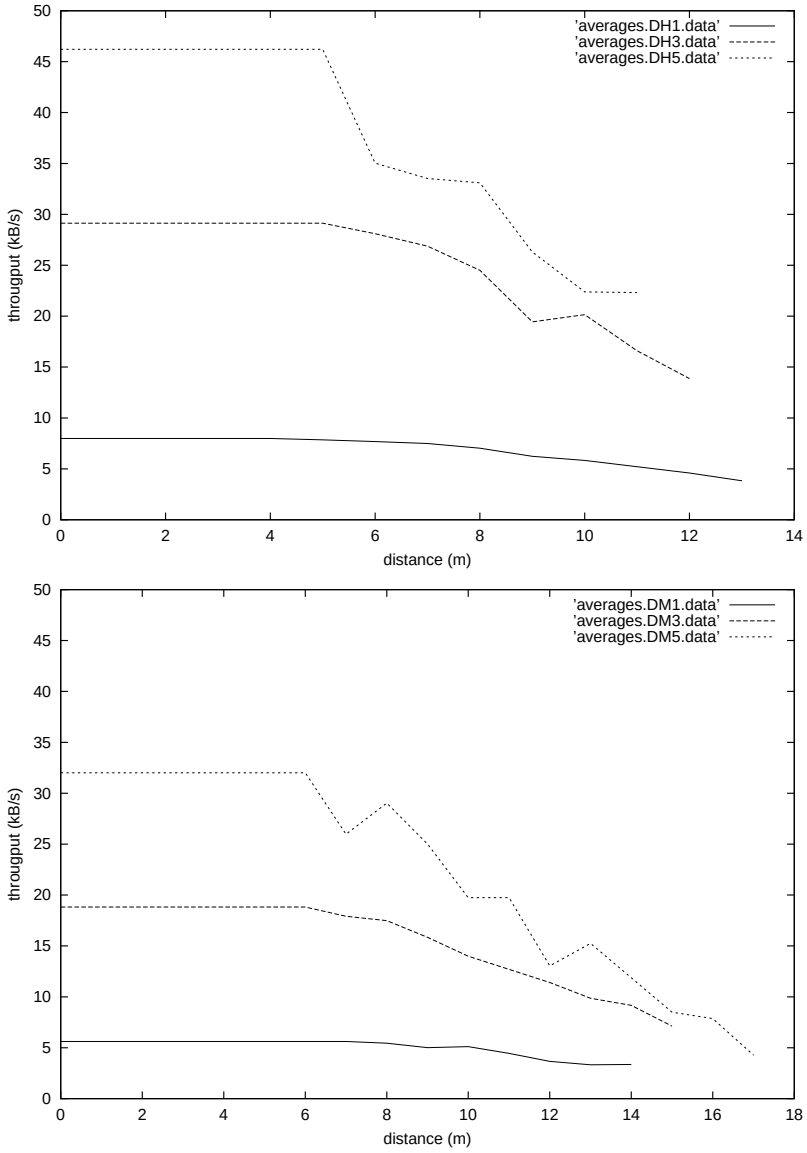

Figure 5 – *Simulated throughput as a function of distance. At the top data high rate connections at the bottom data medium rate.*

Packet type	0 m ($\frac{kB}{s}$) Min/Avg/Max	10 m ($\frac{kB}{s}$) Min/Avg/Max	Theoretical ($\frac{kB}{s}$) BaseBand
DH1	7.00/ 7.99/ 9.81	2.97/ 5.83/ 9.72	21.60
DM1	4.74/ 5.62/ 6.64	3.37/ 5.11/ 6.64	13.60
DH3	26.76/29.14/32.66	9.90/20.15/32.66	73.20
DM3	17.88/18.82/21.82	5.09/14.00/21.82	48.40
DH5	35.02/46.21/67.93	11.34/22.38/67.93	90.49
DM5	24.23/32.01/47.35	12.97/19.74/46.01	59.73

Table 3 – *Min, max and average simulated throughput at 0 m and at 10 m. The slight overhead of the l2cap connection is disregarded. Theoretical maximum bandwidth for asymmetric connections at the baseband level [BT02][65] is listed for comparison . Max/min values illustrate the large span of the measurements.*

surements, but our first impression is that the baseband model fails to produce realistic measurements.

The fluctuations of each measurements are very likely due to the packet nature of the transmission: either the packet gets through with no errors showing high bandwidth or the packet has to be retransmitted showing low bandwidth.

5.5 Conclusion

The duration of the inquiry falls within the range of possible durations from the model in section 2.3. However, since there is no variation in the measurements it is hard to say whether this confirms the model or not.

In conclusion we should say we find that Bluehoc integrates very poorly with the NS framework, making it troublesome to access the features of NS without changing Bluehoc. The graphical configuration utility lacks the ability to create more elaborate scenarios, and cannot load a previously stored scenario. A little less than half of our throughput simulations and many inquiry simulations crashes Bluehoc. The documentation of the simulator is very limited. All in all the Bluehoc simulator does not seem to be ready for widespread use and requires more work to function correctly.

6 Experiments

The purpose of our experiments is twofold: first we want to collect experimental data about actual Bluetooth devices, second we want to collect expertise for further work in the Manatee project. The experiments test throughput and inquiry using point to point connections in two cases:

- Fixed 1 m distance.
- Variable distance.

Our intuitive expectation is that the duration of inquiry will increase and throughput will decrease with distance, as errors become more likely.

6.1 Lab setup

For the throughput measurements we use the *l2test* application of the Bluez package. This application uses the Bluez socket interface to open a connection oriented l2cap data connection—the receiver binds a socket and the sender opens a connection. The sender

repeatedly sends packages with a fixed content and a sequence number. The receiver collects the packages, checks the content and the sequence number. Since the sender transmits the packets as fast as the Bluetooth device can transmit them this program can be used to measure the throughput.

The experiments are conducted in a usual office environment. To measure performance during real life usage no special care is taken to shield the experiments from noise in the area. The office has an 802.11b WLAN that we cannot not shut down during the tests, it is unknown what the activity on this WLAN is during the tests, though since the tests are carried out during the evenings, our feeling is that the activity is low or non existent. We did however make sure that no 802.11b units were active in the same office (radius 5 m). The tests are carried out in a remote office about 50 m from the 802.11b access point with several walls and offices in between.

The distance measurements are approximate. Some devices have external antennas, some are built in and not obvious to locate. We measure from antenna to antenna or from the middle of the device to the middle, so the best accuracy is within a few centimeters.

For the experiments we use 2 laptops:

- HP Omnibook 500, F2168W, 750 Mhz PIII, with an Intel 82371AB PIIX4 USB controller and a TI PCI1410 Cardbus controller.
- Toshiba Satellite 320CDS, PII 233 Mhz, with an NEC USB controller and an Toshiba CardBus controller.

On both machines we run Debian Linux with kernel version 2.4.17. For the Brain Boxes cards Jean Tourrilhes's patch is applied (see section 4.1). The inquiry experiments is carried out using *bluez-2.0-pre6*. Unfortunately this release contains an error for setting up asymmetric connections, so the throughput test are carried out using *bluez-utils-2.0-pre9* and *bluez-kernel-2.1*.

The experiments are carried out using a set of perl scripts issuing the Bluez utility commands. And data extraction from the output files is also done with perl scripts.

6.2 Inquiry

To time the duration of inquiry we make sure that only one Bluetooth device is within range and start inquiry waiting for exactly one device. We do this by we modify the *hcitool* command from the Bluez suite to include an option for specifying a maximum number of inquiry replies (see appendix G) and use the Unix *time* command to measure the duration of the inquiry. To break the correlation between consecutive measurements we wait a random amount of time between each measurement. The inquiry timeout is set to the Bluez default—10 (in 1.28 s units) giving timeout of 12.8 s, and *time* measures exit after 12.822 s.

In order to read the default vendors inquiry parameters we add reading and writing this as an option to *hciconfig* (see appendix E and F). With this we read out the inquiry parameters of our devices.

Device	Inquiry window	Inquiry interval
Bluefrog	18 slots (11.25 ms)	2048 slots (1280.00 ms)
Brain Boxes	18 slots (11.25 ms)	2048 slots (1280.00 ms)

Both the patches have later been included in the distribution of Bluez (Bluez utils version 2.0-pre8 and above).

6.2.1 Variable distance

The wait interval between each measurement used in this experiment is a multiple of the train period: 2.5 s. To separate the devices enough with no obstacles i between, the

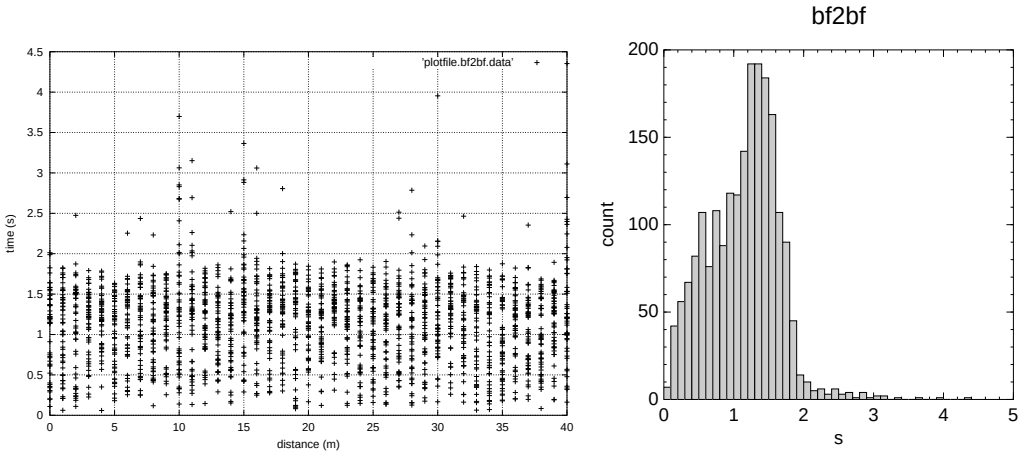

Figure 6 – *Device discovery time as a function of distance using Bluefrog.*

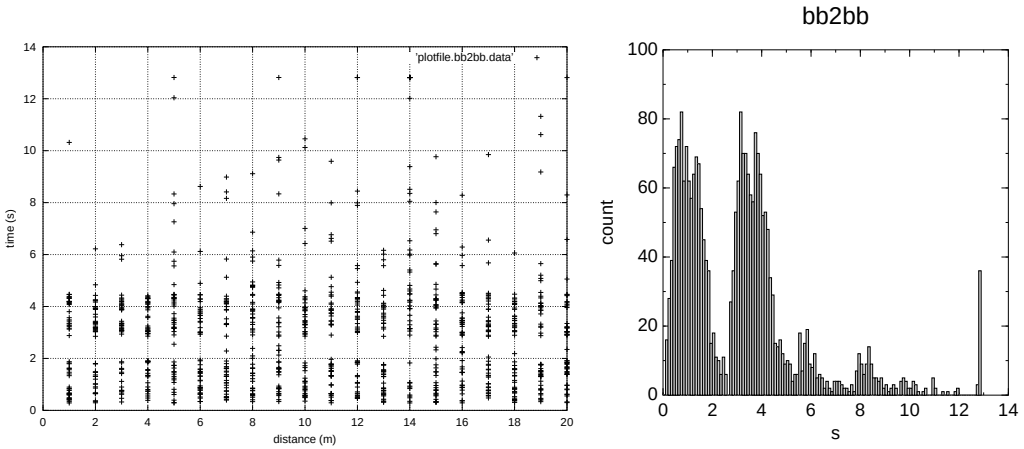

Figure 7 – *Device discovery time as a function of distance using Brain Boxes.*

experiments are carried out in an office hallway during late office hours with few people around and none in the hallway. We run the experiment 50 times at distances increasing from 0 m in 1 m steps.

The result is plotted as a function of distance and the distribution in time using the Bluefrog and Brain Boxes devices respectively in figure 6 and 7.

6.2.2 1 m fixed distance

For this experiment we wish to duplicate the conditions of the experiment carried out by Oliver Kasten and March Langheinrich[KL01]. Their setup was similar to ours: using two preseries Ericsson ROK 100 007 development kits one meter apart they did 1500 inquiries and measured the duration. Their results are:

- The average inquiry time is 2221 ms
- After 1910 ms, 4728 ms and 5449 ms, the target unit had been found in 50, 95, and 99 percent of all tests respectively.

In order to replicate the environment of the experiments just described we increase the random wait interval to match the one used in [KL01]: 12.8 s. Also after each inquiry

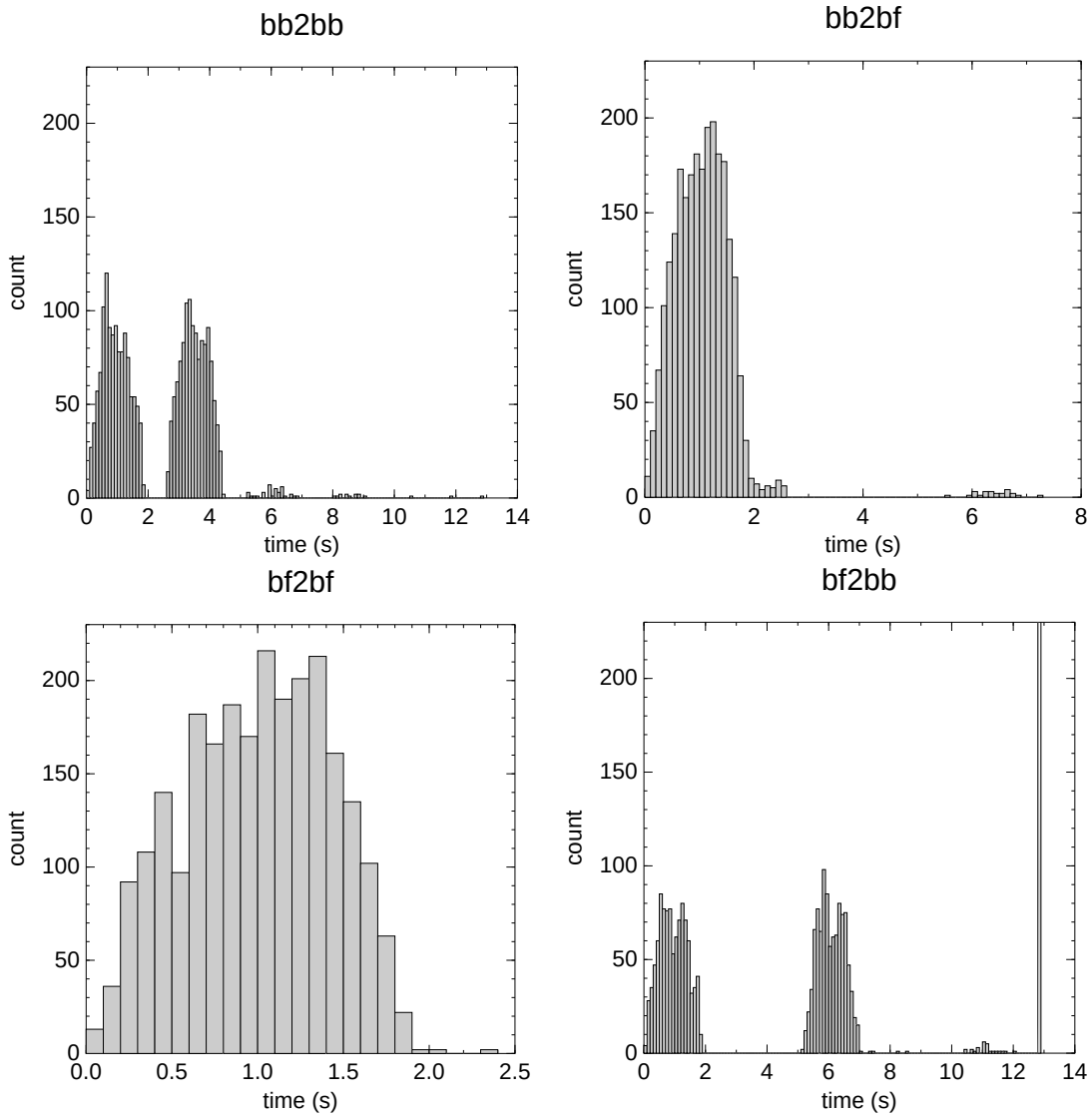

Figure 8 – *Fixed 1 m distance. Distribution of discovery times for the two devices Bluefrog (bf) and Brain Boxes (bb). 12.8 s is the inquiry timeout.*

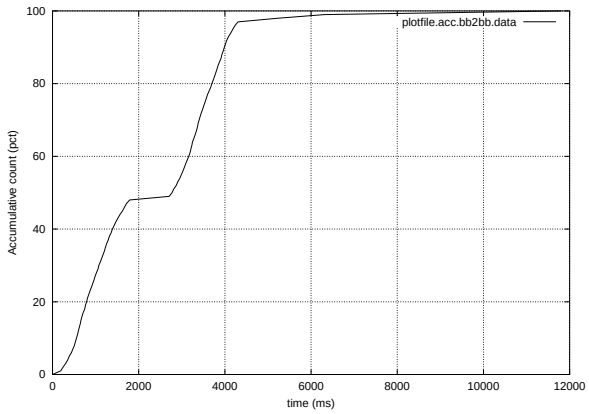

Figure 9 – *Accumulative device discovery times using Brain Boxes devices. There was a single measurement that timed out. This was rejected.*

we reset the devices using *hciconfig* and unload the drivers to make sure that any on device clock timing cache will be cleared. One difference between the experiment carried out by Oliver Kasten and March Langheinrich and ours is that while we stop the inquiry experiment as soon as the first reply is received they continue the full inquiry period even though the reply is received before. We repeat the experiment 2500 times.

The experiment is setup using all combinations of the Bluefrog and Brain Boxes devices. The distribution of discovery times of the results are plotted (see figure 8).

To compare with the experiments described we also plot the accumulative count as a function of time for the Brain Boxes to Brain Boxes experiment (see figure 9). The average device discovery for this experiment is time is 2.33 s. After 2772 ms, 4184 ms, 6311 ms the target unit had been found 50, 95, and 99 percent for all tests respectively.

6.2.3 Conclusion

Based on the Brain Boxes to Brain Boxes experiments (fig 8 top left) we draw the following conclusions:

- The device discoveries are centered in intervals of the train length (see figure 8). It seems equally probable to find a device in either the first or the second train (see figure 9). This experiment confirms the prediction of the model from section 2.3, and also our intuitive feeling that in practice few errors occur during inquiry.
- Looking closely at the first two chunks in the Brain Boxes to Brain Boxes experiment (top left figure 8) it is possible to make out that each peak has two distinct tops. Looking at inquiry as a function of distance using the Brain Boxes cards (figure 7) we see this two-peak pattern much more distinctly: as distance increases, errors are likely, and as a result more retries are used. The position of the retries (the second top) is predicted by our model.
- The average found (2.33 s) is very close to the ideal (2.23 s) as predicted by our model.
- In contradiction to our intuitive expectation the inquiry procedure does not seem to be sensitive to distance. The variable distance experiment shows (figure 6, 7) that good property is due to the error recovery mechanisms giving a device a second chance of being discovered within the duration of a train.

These results lead us to conclude that the model described in section 2.3 is correct.

Our experiments seem to confirm the findings by Oliver Kasten and March Langheinrich. The distribution patterns are very different, ours showing distinct peaks while theirs show a much more flat distribution. Even so the averages and accumulative counts seem to agree fairly well.

Based on the experiments with the Bluefrog kit we conclude:

- The Bluefrog to Bluefrog experiment (fig 8 bottom left) shows that the Bluefrog kit does not use the procedure described in the specification. The experiment shows that the method chosen is very fast using only Bluefrog devices. It seems that the Bluefrog kit does not split the inquiry frequencies in two trains since all devices are found within one train period.
- From the Brain Boxes to Brain Boxes experiments it seems safe to assume that the Brain Boxes cards follow procedure from the specification. The Brain Boxes to Bluefrog experiment (fig 8 top right) shows that the Bluefrog device is discovered in first train or in the repetition of this train, never in the second train.
  - Since the A train of the Brain Boxes card is chosen arbitrarily with respect to the Bluefrog kit this suggests the Bluefrog device is able to listen at two frequencies separated by 16 frequencies. This would mean that no matter how the train is split at least one of the frequencies would be right
  - This would give the Bluefrog devices missed during the A train a second chance during the B train since it would always listen to a frequency in both the A and the B train. No devices are found during the B train, so this cannot be the case.

  It is possible that the inquiry scanning Bluefrog devices chooses its phase in the inquiry sequence based on previous inquiries. Between each measurement inquiring device is (software) reset and long random wait interval is introduced—the random wait interval should make any caching at the inquiry scanning end invalid. In order to shed more light on the matter it would be necessary to conduct a further experiment reseting the inquiry scanning device on each run.

  Unloading the drivers and resetting the devices should clear all caches at the inquiring end—starting each inquiry from scratch having no idea on which frequency to start sending ID packets. As the Bluefrog to Bluefrog experiment is carried out exactly like the Brain Boxes to Brain Boxes experiment. It does not seem probable that the inquiring devices has a cache not being cleared.

- The Bluefrog to Brain Boxes experiment (fig 8 bottom right) shows that the method chosen to send inquiries cannot be the one we described either. A large part of the inquiries simply time out. The graphs show two distinct peaks suggesting that two trains are chosen. It is tempting to guess that the gap between the peaks (about 2.5 s long) stems from a train run that is always wrong. An other interpretation could be that a non standard train length is used. Yet another possibility is that the device uses only a limited amount of frequencies even though it reports to use the full frequency set (US, Japan, Europe).

6.3 Throughput

The throughput measurements are carried out using the *l2test* program of the Bluez package. This program sends known content from one end to the other and inserts a sequence number in each packet sent. At the other end the content as well as the sequence number is checked to see if transmission was without errors. The throughput is calculated

as an average over several l2cap packets, by recording the time just before receiving the first packet and just after receiving the last. Here we choose to calculate throughput every time 20 kB has been transfered.

At the baseband level the tunable parameter are the packet size (1, 3, 5) and the data encoding (medium/high rate). At the l2cap level the tunable parameters are the size of the l2cap packets. To minimize the overhead this could be chosen very high, say 10 kB, but devices tend to do cross layer optimizations assembling several lower layer packets on the device before handing an l2cap packet up through HCI. The devices have limited memory and tend to be optimized toward small packet sizes (the default is 672 kB), here we choose a conservative 2 kB for the l2cap packet size. The channel retransmissions is set to infinite retransmissions or "reliable channel" by setting the flush timeout to 65535. The socket opened is of type *SOCK_SEQ_PACKET* ensuring a reliable connection oriented connection.

The packet type is chosen with "*hcitool ptype*" which set the default packet type, however if no data is available the Link Manager sends a NULL packet [BT02][p. 79]. This means that in order to control which packets flow in both directions we have to send in both directions. The *l2test* program of Bluez-2.0-pre9 does not support bidirectional communication, so we implement this functionality as the options "-x" and "-y" in lack of better names, see appendix H.

For both fixed distance and variable distance we run the *l2test* program for approximately 2 minutes at each experiment. This is hand-timed using a wrist watch, so the duration may be longer, but no less. During the tests the system stop transmitting frequently after approximately 2 minutes. Unloading the drivers removing the cards and starting over allowed the experiment to continue. The tests are carried out by setting the default packet type manually on both nodes using hciconfig and starting *l2test*.

Even with the l2cap layer set to infinite retransmissions and a reliable socket the connections showed packet loss as missed sequence numbers with multi slot packets using the Brain Boxes cards. A discussion with Maksim Krasnyanskiy (one of the main authors of Bluez) on the Bluez mailing list did not shed any light on the matter other than either the hardware or the software has a bug. The best guess we can make is the serial driver patch.

The Bluetooth drivers for the Brain Boxes cards is somewhat troublesome requiring kernel patches and the *hciattach* command. The stability of this setup seems to be questionable, and during our tests crashes and lockups were frequent. Most notably during the bidirectional tests one end of the socket just stopped sending at some point for no apparent reason, while receiving continued with no problems. This is seen by the link manager which starts to send NULL packets as return in effect allocating more slots to the opponent. This showed quite clearly as sudden increase in bandwidth at one end, when this occurred the experiment were aborted and the obviously wrong measurements discarded.

Some part of the Linux/Bluez/Bluefrog seems to have problems with multi slot packet sizes, a discussion with Maksim Krasnyanskiy on the Bluez mailing list did not provide a solution in time, and the experiments using these devices were dropped.

Progress seems to be on the way on all fronts, but at the time of writing none of the issues have been solved.

The tunable parameters for the fixed distance case is the packet type and the slot length. We will setup both symmetric and asymmetric connections. The asymmetric connections consists of a multi slot packet and its single slot counterpart DM3 one way DM1 the other for one experiment, DH3 one way DH1 the other for the next, etc.

The experiment is carried out by setting one node to listen for a connection and one to connect using the our newly added "-x/-y" options of *l2test*. The incoming MTU is set with "-I" as 2 kB, and the l2cap sending packet length is set with "-b" as 2 kB.

We plot the throughput as a function of the slot length for data high rate and medium

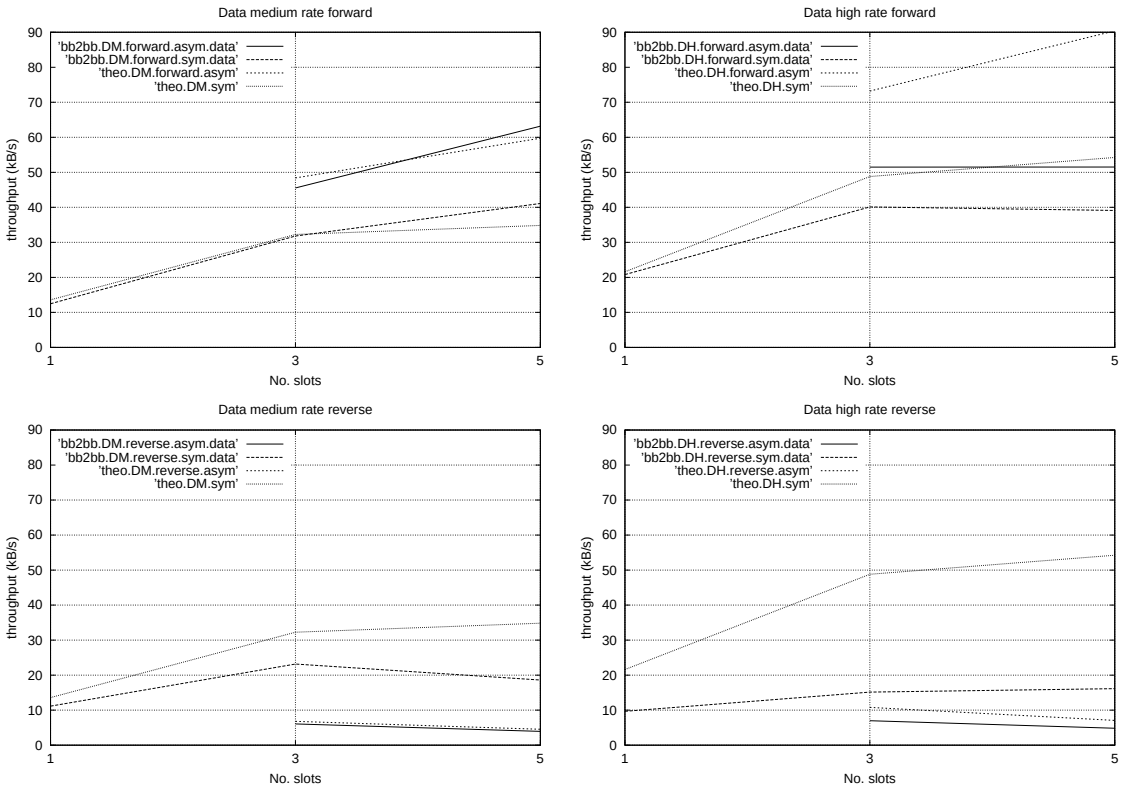

Figure 10 – *Throughput measurements at 1 m fixed distance as a function of slot length. The left side shows data medium rate experiments, the right side data high rate experiments. The figures on the top row show the forward direction of the measurements, while the figures on the bottom show the reverse direction. A point in the top row matches the same experiment in the bottom row in the opposite direction, e.g., the top left point (5,41) is the forward direction and the bottom left point (5,18) is the reverse direction of the experiment: medium rate, 5 slot packets in both directions.*

rate connections in both directions as well as the theoretical maximum bandwidth at the Baseband level in figure 10.

6.3.1 1m fixed distance

The figures show the experiments where the master is using the highest slot length. We conducted experiments where the slave was using the highest slot lengths these experiments show significantly lower bandwidth is given to the slave when it is using a high slot length as opposed to the master. For some reason the master (which is control of the slot allocation) does not assign the same slot allocation if the slave wishes to use the highest slot length.

6.3.2 Variable distance 0-20m

For the sake of time we only conduct this experiment using two packet types: the most robust (DM1) and the most error sensitive (DH5) with transmission in one direction only. With these two packet types we expect DM1 to show more robustness than DH5. The experiments are carried out in the same hallway as the inquiry experiments.

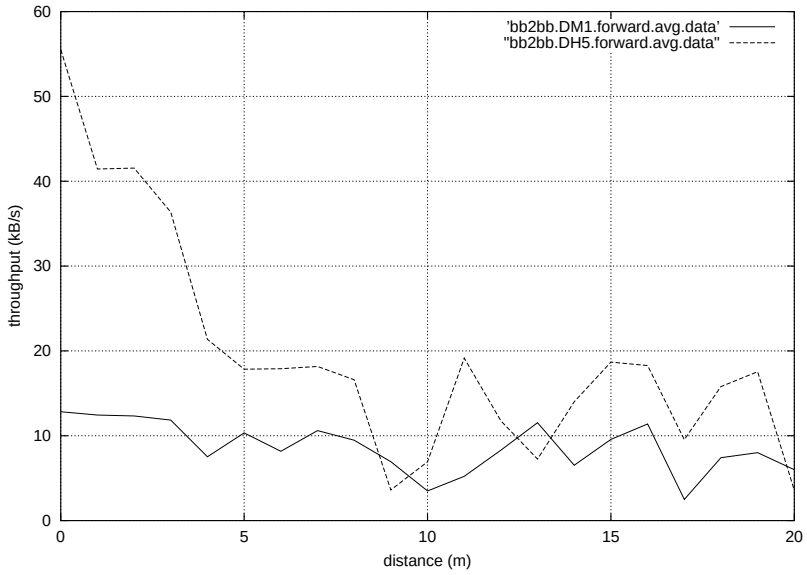

Figure 11 – *Average measured bandwidth as a function of distance. The solid line shows a symmetric DM1 connection, while the dashed lines shows a symmetric DH5 connection.*

The experiments are carried out by selecting one node as receiver and one as sender with the *l2test* "-r" and "-s" options. On the receiver the incoming MTU is set with "-I" to 2 kB and the amount of data to receive before calculating throughput is set with "-b" to 20 kB. On the sender the sending packet size is set to 2 kB with "-b" (notice that the meaning of -b is different for sender and receiver).

We plot the measured throughput as a function of distance in figure 11.

6.3.3 Conclusion

In the fixed distance case the maximum bandwidth exceeds the maximal theoretical bandwidth at several point suggesting that out measuring method is inaccurate.

- The large variation of the throughput as the distance increases can suggest several things:
  - The experiments were run not run long enough. It is possible that the variations would have been less dramatic if the experiments were allowed to run longer. This was rather troublesome as the system had problems running longer than 2 minutes.
  - The noise in the area was much larger than anticipated which can make the measurements fluctuate wildly.
  - We are not too confident in the stability of the serial driver for the Bluez cards. It is possible that the serial driver has some blame.
- For the data high rate connection all the measurements below 5 m lie above approximately 20 $\frac{kB}{s}$ and all the measurements above 5 m seem to be below. For the medium rate connections it is not possible to see such a distinction. We can thus with some certainty say that we have shown that the high rate connections are less robust than medium rate.

Because of the large variation of the measured throughput it is hard to do a quantitative comparison with the simulations. Even, so we are going to make the following

observations:

- The simulation of the DH5 connection experiment seem to have a maximum at about 47 $\frac{kB}{s}$, while the DH5 experiments have maximum at about 53 $\frac{kB}{s}$ compared to the theoretical maximum at 90.4 $\frac{kB}{s}$.
- The degradation of the average of the measurements with distance of throughput for the DH5 experiments seems much more rapid than the simulations.
- The simulation of the DM1 connection is below 6 $\frac{kB}{s}$ while the experiment varies between 12 $\frac{kB}{s}$ and 4 $\frac{kB}{s}$ compared to the theoretical maximum at 13.60 $\frac{kB}{s}$.
- The downhill trend of the of the simulations cannot be confirmed by the experiments above 5 m. In the experiments it becomes more unpredictable what the bandwidth will be as the distance increases.

It is hard to make any solid conclusions about the free space model and Bluetooth implementation of Bluehoc based on theses observations. Both the simulation and the experiments show large fluctuations, even so the average in the simulations show a downhill trend more clearly than the experiments. This could suggest that the model imposes less noise than what is present in our environment. It seems peculiar that the simulations show somewhat slower throughput than both the experiments and theory for both high rate and medium rate connection. This suggests that either the setup is wrong or the simulator has problems nearing peak bandwidth.

As a remark about Bluez we would like to say that we found Bluez very pleasant to work with. The Bluez stack provides a very convenient programming environment. It is well structured and provides a good API for writing applications. The utilities were easily expanded to fit our needs and could serve as good examples. The community surrounding Bluez, has been very helpful and development seems to stride forward at a steady pace.

7 Conclusions

This project had several purposes. One was to collect experiences working with Bluetooth and Linux for the Manatee project. We chose the Bluez stack as it is included in the Linux kernel. During this project is has become apparent, that while the Bluetooth support is well underway and has good and solid design there are still some glitches to be worked on—at least with the hardware we used.

- The Bluefrog devices failed to run the throughput tests.
- Crashes were frequent during the throughput tests with the Brain Boxes devices.
- We extended the Bluez utilities with useful features

We evaluated the Bluehoc simulator. While the infrastructure for the simulator seems to be sensible it still has major issues. We experienced traffic generators not supported, many segmentation faults and problems passing parameters for the traffic generators. However one of the major hurdles for using Bluehoc was the lack of documentation.

- The comparison between the inquiry simulations and the experiments was inconclusive. While the only simulated result obtained was within the acceptable range, this does not by any means show that the inquiry simulation is working as we expect it to.
- The maximum throughput of the simulation and the experiments did not show a large resemblance. The DM1 experiment showed almost twice higher throughput as

the simulation. The DH5 experiment show significantly higher throughput over the simulation. This seems odd and suggests that either the setup is wrong or the BB simulation model is flawed.

- The degradation of throughput with distance of the simulation and experiments did not show the same characteristics. The DM1 experiment did not degrade but instead varies wildly as distance increases. The DH5 experiment showed much more rapid drop off than the simulation. We conclude that the free space model of Bluehoc does not match our conditions very well.

We measured some key features of Bluetooth showing inquiry procedure as well as the single slot medium rate connection to be distance robust. In the smart tag context these results are important lessons using Bluetooth.

- We proposed a model of the inquiry procedure. This model allowed us to predict and explain the performance of the inquiry procedure. We validated our model with experiments.
- We saw that the inquiry time of the two devices had very different inquiry characteristics. Based on the measurements presented here we cannot say whether this is a general pattern, but our guess is that this is not the case and most devices will follow the specification.
- We showed good agreement with the experiment conducted in [KL01].

References

[KG01] Arpurva Kumar, Rajeev Gupta: *Capacity Evaluation of Frequency Hopping Based Ad-hoc Systems*, Sigmetrics, 2001.

[BRAY01] Jennifer Bray, Charles F Struman: *Bluetooth Connect Without Cables*, Prentice Hall, 2001

[RG02] Rajeev Gupta http://www-124.ibm.com/pipermail/bluehoc-discussion/2002-February/000373.html

[KL01] Oliver Kasten, Marc Langheinrich: *"First Experiences with Bluetooth in the Smart-Its Distributed Sensor Network"*, proc. PACT 2001.

[BHFAQ] Abhinandan Sharma, Kaushik Raghunath, Kavitha Subramanian: *Bluehoc And Bluescat Frequently Asked Questions* http://www.cse.iitb.ernet.in:8000/proxy/everest/ abhish/faq.htm

[BHSITE] Apurva Kumar, Rajeev Gupta (IBM India): *Bluehoc simulator* http://www-124.ibm.com/developerworks/projects/bluehoc

[BT02] Bluetooth SIG: *Specification of the Bluetooth System, Core*, version 1.1, 2002 http://www.bluetooth.org/docs/Bluetooth_V11_Core_22Feb01.pdf

[BTP02] Bluetooth SIG: *Specification of the Bluetooth System, Profiles*, version 1.1, 2002 http://www.bluetooth.org/docs/Bluetooth_V11_Profiles_22Feb01.pdf

[MT] DIKU: *Manatee project* http://www.distlab.dk/manatee/goal.htm

[AP] *Apparater.dk* http://www.apparater.dk

[BZ] Maksim Krasnyanskiy and others: *Bluez* http://bluez.sourceforge.net/

[AXIS] Axis: *OpenBT* http://developer.axis.com/software/bluetooth/

[NS2] Informationan Sciences Institute, University of Southern California: *The Network simulator - NS-2* http://www.isi.edu/nsnam/ns/

Appendix A - Bluehoc notes

This appendix present our notes on working with the Bluehoc simulaor.

Installation (Bluehoc 3.0 NS 2.1b8)

Get the ns-allinone-2.1b8a.tar.gz archive from ISI. Get Bluehoc from the Bluehoc project site, Bluehoc 3.0 is at the time of writing only available from CVS under the name bluehoc2.0.
We have had the most success using gcc 2.95.2, using gcc 2.96 did not seem successfull.
First run the installation of NS then install Bluehoc. The procedure is described as:
Use the first method called "IF YOU ALREADY HAVE NS INSTALLED", the other won’t work! The patches are different and doesn’t acomplish the same.

Command	Explenation
emacs -nw SetUpEnvironment	Create a script setting up NS_HOME and PATH as descibed in the README file. Include ns-allinone-2.1b8a/otcl-1.0a7 and ns-allinone-2.1b8a/lib in LD_LIBRARY_PATH (at DIKU be carefull what else you have in LD_LIBRARY_PATH - including wrong things might make the install fail with obscure errormessages). Finaly set TCL_LIBRARY=$NS_HOME/ns-allinone-2.1b8a/tcl8.3.2/library.
cd $NS_HOME && tar zxfv ns-allinone-2.1b8a.tar.gz	Untar the archive
cd ns-allinone-2.1b8a/ns-2.1b8a	
./install	Build NS without bluehoc
tar -zxvf bluehoc3.0.tar.gz	Untar, copy or get the directory *bluehoc* containing the bluehoc installation files.
	If installing Bluescat replace the files from this distribution with those included in Bluescat.
cp bluehoc/src/*.cc bluehoc/src/*.h .	
cp bluehoc/tcl/ns-btnode.tcl tcl/lib/	
patch -p1 -b < bluehoc/patches/bt_patch	Apply the patches

emacs Makefile	You need to manually add the BLUEHOC C++ files for compilation in OBJ_CC: Add the following line to OBJ_CC line:
	`baseband.o bt-classify.o bt-drr.o bt-host.o bt-lc.o l2cap.o lbf.o lmp.o\`
	and add ns-btnode.tcl in the NS_TCL_LIB macros of the Makefile. Add the following line to NS_TCL_LIB:
	`tcl/lib/ns-btnode.tcl \`

touch *.cc && make -j4	Remake the executable
	Running simulations
mkdir run	Make a spare directory
cp bluehoc/tcl/*.tcl bluehoc/examples/*.tcl run/	
cd run;../ns simXYZ.tcl	Run your simulations

Tunable parameters

Bluehoc 3.0 has the following parameters for tuning the node configuration

1. Sim(Transport): A list of transport layers to be simulated for each link. Options include (se Bluhoc manual for more info): UDP, TCP/Tahoe, TCP/NewReno, TCP/Sack1, TCP/Vegas
2. Sim(Application): A list of applications for each link.
   Apparently support for setting options for traffic generators is fairly limited. In the *findAppNames* in the file *bt-host.cc* apparently only checks for packetSize for TrafficExponential. It looks to me like no other options are read.
   Options are set by including a line like the following in the .tcl simulation script.
   Application/Traffic/Exponential set packetSize_ 5000;
   Traffic generators from NS include NS traffic generators including (from the NS manual):
   Rate is specified in kBps.
   Defaults are located in *ns-default.tcl*

Name	Parameters	Explenation
FTP		FTP application
Telnet		Telnet application
Traffic/Exponential	packetSize_, burst_time_, idle_time_, rate_, shape_	ON/OFF bursty traffic eg. voice. Generates traffic according to an Exponential On/Off distribution. Packets are sent at a fixed rate during on periods, and no packets are sent during off periods. Both on and off periods are taken from an exponential distribution. Packets are constant size.
Traffic/Pareto	packetSize_, burst_time_, idle_time_, rate_, shape_	Same as above, but the on and off periods are taken from a pareto distribution. These sources can be used to generate aggregate traffic that exhibits long range dependency.
Traffic/CBR	packetSize_,rate_,random_	Apparently not supported!Constant bit rate. generates traffic according to a deterministic rate. Packets are constant size. Optionally, some randomizing dither can be enabled on the interpacket departure intervals.
Traffic/Trace	filename_	Generates traffic from a trace. generates traffic according to a trace file. Each record in the trace file consists of 2 32-bit fields. The first contains the time in microseconds until the next packet is generated. The second contains the length in bytes of the next packet.

3. Sim(NumDevices): No. devices
4. Sim(xpos_), Sim(ypos_): List of device positions
5. InqTimeout: The period that the master will perform inquery
6. NumResponses: The number of inq respones that the master will accept
7. The estimates on the likelyhod of packetloss i located in the file *distfer.h*
8. To get an nonzerro error estimate on ID packages insert a new row in *distfer.h* and comment out the following line (no. 888) in baseband.cc:
   fer = (bt->type == BT_ID) ? 0.0 : fer;

QOS negociation

Prior to connecting Bluehoc negociates QOS parameters. This is done by mapQoS in bt-drr.cc. The QoS is setup based on the *appFlowSpec* parameters in bt-host.cc. The negociated parameters are sent to the scheduler, and if imposible parameters are requested the negociation fails - meaning that these parameters cannot be used to over saturate a link.

1. If *loss_sensitivity* is set it asks for FER encoding ei. DM packets otherwise DH packets.
2. The packet type is selected on basis of the MTU and whether DM og DH has been selected. The numbers specify maximum MTU for this packet size.

DH	**DM**	
343 and above	228 and above	slotlength 5
187	125	slotlength 3
31	21	slotlength 1
		rejected.

3. *token_rate* is used to set the poling interval: pi = packetLength*8/(token_rate*1000*625*1e-6));

Limitations

At this point the following limitations have been found

1. Bluhoc only supports simplex transmissions from Master to slave. The link has information on settin up communication the other way. Click here.
2. Tuning parameters like the following are missing: Support for setting up multislot packet communication, automatic or explicit controll of how the master allocates slots for each slave, tuning the inq scan states.
3. Bluescat 0.6 has no NAM support
4. The drawings of NAM trace apparently has little to do with the positions of the nodes.
5. The Maximum dist. is set to 20m, to raise it further you would have to recalculate the error table in distfer.h

Nam traces

A simple setup of two nodes doing first inquiry, then pageing and then connection with FTP traffic looks something like this in nam. I’ve written my best guess on what the state changes are:

clock	Slave	Master	
0	ST	I	Standby/Inquery
4.96	PS		PageScan
	ST		Standby
5.15	IS		InqueryScan
	ST		Standbty
5.63	IR		InqueryResponse
5.64	IS	P	InqueryScan/Paging
5.13	ST		Standby
6.26	PS		PageScan
6.26	SR	MR	SlaveResponse/MasterResponse
6.27	C	C	Connected/Connected

Output format

Connection establishment

If I setup a simple connection with only 2 nodes and an FTP-app the output looks something like:

appname[0] = Application/FTP

INQ_MSG ****-->1	CLKN: 16472	clock: 5.147767e+00
INQ_MSG AFTER BO *-->1	CLKN: 18040	clock: 5.637767e+00
FHS_PKT 1-->0	CLKN: 18043	clock: 5.638710e+00
PAGE_MSG ****-->1	CLKN: 20056	clock: 6.267767e+00
PAGE_ACK 1-->0	CLKN: 20058	clock: 6.268397e+00
MASTER FROZEN	CLKE: 20058	
FHS_PKT 0-->1	CLKN: 20061	clock: 6.269330e+00
FHS_ACK 1-->0	CLKN: 20062	clock: 6.269642e+00
POLL_ACK 1-->0	CLK: 20067	clock: 6.271205e+00
LMP_HOST_CONNECT_REQ	AM_ADDR: 1	LMP_ACCEPTED AM_ADDR: 1 ACCEPTED PDU: LMP_HOST_CONNECTION_REQ
LMP_QOS_REQ	AM_ADDR: 1	LMP_ACCEPTED AM_ADDR: 1 ACCEPTED PDU: LMP_QOS_REQ
RECV L2CAP_CONNECT_REQ	CH: 0	
RECV L2CAP_CONNECT_RSP	CH: 0	
L2CA_DATA_WRITE	SIZE: 40	CID: 2 DEST_IP 1 clock 6.274955

....

From reading baseband.cc I come up with the following:
INQ_MSG ****-->

Is printed as soon as the slave sees ID packages (that is when the master is transmitting and the slave is in INQ_SCAN state) just before starting random back off for reply.

INQ_MSG AFTER BO *-->
Is printed when the second ID message after the back off is received, the slave is in ICQ_RESP state.

FHS_PKT %d-->
FHS packet received at the master with clock and address information

So at the FHS_PKT time the inq is over. The question is when did the master start sending ID packets? The Bluehoc manual hints that it does that right away meaning clock=0. So for this scenario it took 5.638710s for the master to discover the slave.

As far as I can tell the master instantly switches to paging. The connection can be said to started once booth ends are in the CONNECTED state. This has happened in the slot after the ID package sent as ACK for the FHS (I'm guessing FHS_ACK). The master is required to send an Poll packet at the next slot.. Whether this is part of the connection establishment or the actual connection is a matter of definition. For my purposes the minimal timing difference is not important. So for this example it took an additional 6.271205s-5.638710s=0.632495s.

Connected transmission

The BT traces contain information about l2cap signaling and for each sent packet the BT traces contain lines like (see Bluehoc tutorial):

BD_ADDR 1 DELAY 4.375000e-03 SIZE 40 clock 6.279330e+00

This line is printed each time L2CAP has a complete packet to send to the upper layer. And thus indicates a successfull l2cap transmission.

The 4.375000e-03 indicates the time elapsed from L2CA_dataWrite method has released the packet to the L2CA_dataRead receives it. SIZE is the size in bytes.

Bugs

1. The CVS version of Bluehoc 3.0 is located in the project dir Bluehoc 2.0 wich can be confusing.
2. The *Payload* array in globals.h seems to contain wrong values. Instead of payload lengths it contains entire packet length+2. Whether the Bluehoc compensates for this some otherplace is hard to say.

3. I create a simple scene of 2 nodes, set up an ftp connection between them, and select the graphs and BT trace output. Which fails with the error at the bottom.

   If I look at the BT trace output file I find this odd looking line: BD_ADDR 1 DELAY 2.981250e-01 SISimulation over

   To mee it seems that since both bt-host.cc write to stdout, the prints somehow get mixed up.

   I did a quick hack by letting bt-host.cc write to a file of it's own and bluehoc.tcl read this file to get the "DELAY..." lines. This seems to work and I get my graphs, but a more elegant solution would be appriciated (the BT-trace file could be opened somewhere else and all the output now going to stdout

could be written to this file).

An other solution is to remove the printouts from the bottom of *run.tcl*.

e247net mentioned that increasing the simulation works as well. If this is true I asume it is because the "Simulation over" is delayed or by accident put on a line where it doesn’t interfere. syntax error in expression "2.3604e+05+over"

while executing
"expr $thr($addr)+$bytesrecv"
(procedure "plotXgraph" line 25)
invoked from within
"plotXgraph $delayop 3 1 5 7 5e-3 100 $fileid "
(procedure "dump-config" line 74)
invoked from within
"dump-config"
(procedure "save-file" line 9)
invoked from within
"save-file"
invoked from within
".#menubar.#menubar#mFile invoke active"
("uplevel" body line 1)
invoked from within
"uplevel #0 [list $w invoke active]"
(procedure "tkMenuInvoke" line 29)
invoked from within
"tkMenuInvoke .#menubar.#menubar#mFile 1"
(command bound to event)

Appendix B
[Bluehoc-discussion] Re: Inquiry has perfect reception regardless of distance!?

Rajeev Gupta grajeev@in.ibm.com
Thu, 28 Feb 2002 16:17:22 +0530

- Previous message: [Bluehoc-discussion] Re: Inquiry has perfect reception regardless of distance!?
- Next message: [Bluehoc-discussion] sim007.tcl problems
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hi,
You can see our paper published in Sigmetrics’2001 titled "Capacity
evaluation of frequency hopping based ad-hoc systems", where we have
described our simulation model and parameters of interest. Frame error rate
is function of:
1. Physical layer properties like modulation,  FEC/CRC used etc.
2. Channel properties affecting interference.
3. Spatial properties like distance between interferer and receiver,
topology etc.
 These are used to calculate BER with/without FEC.
BER is used to calculate FER depending on frame-size as
FER=1-(1-BER)^(no. of bits)
Hope this will satisfy your queries.

Thanks and regards.
-Rajeev Gupta
Research Staff Member
IBM India Research Lab
Block 1, Indian Institute of Technology
Hauz Khas, New Delhi, INDIA 110016
Email: grajeev@in.ibm.com
Phone: 91-11-686-1100
Fax: 91-11-686-1555
```

- Previous message: [Bluehoc-discussion] Re: Inquiry has perfect reception regardless of distance!?
- Next message: [Bluehoc-discussion] sim007.tcl problems
- **Messages sorted by:** [date] [thread] [subject] [author]

Appendix C
[Bluehoc-discussion] Inquiry has perfect reception regardless of distance!?

Rajeev Gupta grajeev@in.ibm.com
Thu, 28 Feb 2002 09:44:52 +0530

- Previous message: [Bluehoc-discussion] Apurva and Rajeev still working on Bluehoc?
- Next message: [Bluehoc-discussion] Re: Inquiry has perfect reception regardless of distance!?
- **Messages sorted by:** [date] [thread] [subject] [author]

```
Hi,
FER(Frame error rate) for ID packet was assumed to be 0 because of its very
samll size.

Thanks and regards.
-Rajeev Gupta
Research Staff Member
IBM India Research Lab
Block 1, Indian Institute of Technology
Hauz Khas, New Delhi, INDIA 110016
Email: grajeev@in.ibm.com
Phone: 91-11-686-1100
Fax: 91-11-686-1555
```

- Previous message: [Bluehoc-discussion] Apurva and Rajeev still working on Bluehoc?
- Next message: [Bluehoc-discussion] Re: Inquiry has perfect reception regardless of distance!?
- **Messages sorted by:** [date] [thread] [subject] [author]

```
--- bluehoc/src/bt-host.cc      Fri Mar 22 15:31:40 2002
+++ bt-host.cc  Tue Mar 26 16:03:43 2002
@@ -402,6 +402,34 @@
                tcl.evalf("%s info class", trafgen_[i]->name());
                appNames_[i] = tcl.result();
                const char* p = appNames_[i].c_str();
+
+               if (strcmp(p,"Application/Traffic/CBR")==0) {
+                 tcl.evalf("%s set packetSize_", trafgen_[i]->name());
+                 int mtu = atoi(tcl.result());
+                 appFlowSpec_[i]->mtu = mtu;
+
+                 tcl.evalf("%s set rate_", trafgen_[i]->name());
+                 int rate = atoi(tcl.result());
+                 appFlowSpec_[i]->token_rate = rate;
+
+                 appFlowSpec_[i]->loss_sensitivity = 0;
+
+                 appFlowSpec_[i]->token_bucket = 0; //No token bucket
+                 appFlowSpec_[i]->input_qlen = 30;
+                 appFlowSpec_[i]->peak_bw = 1e8;
+                 appFlowSpec_[i]->latency = 100;
+
+                 cout << "Parameters: MTU " << appFlowSpec_[i]->mtu
+                      << ", token_rate " << appFlowSpec_[i]->token_rate
+                      << ", loss_sensitivity " << appFlowSpec_[i]->loss_sensiti
vity
+                      << ", token_bucket " << appFlowSpec_[i]->token_bucket
+                      << ", input_qlen " << appFlowSpec_[i]->input_qlen
+                      << ", peak_bw " << appFlowSpec_[i]->peak_bw
+                      << ", peak_bw " << appFlowSpec_[i]->peak_bw
+                      << ", latency " << appFlowSpec_[i]->latency
+                      << endl;
+                 }
+
                if (strcmp(p,"Application/Traffic/Exponential")==0) {
                        tcl.evalf("%s set packetSize_", trafgen_[i]->name());
                        int mtu = atoi(tcl.result());
```

```
Binary files bluez-libs-2.0-pre7.clean/conftest and bluez-libs-2.0-pre7/conftest
 differ
diff -ru bluez-libs-2.0-pre7.clean/include/hci.h bluez-libs-2.0-pre7/include/hci
.h
--- bluez-libs-2.0-pre7.clean/include/hci.h     Fri Mar  8 22:10:07 2002
+++ bluez-libs-2.0-pre7/include/hci.h   Tue Mar 19 11:28:27 2002
@@ -206,6 +206,23 @@
 /* Host Controller and Baseband */
 #define OGF_HOST_CTL   0x03
 #define OCF_RESET              0x0003
+#define OCF_READ_INQ_ACTIVITY  0x001D
+
+typedef struct {
+        uint8_t         status;
+       uint16_t         interval;
+       uint16_t         window;
+} __attribute__ ((packed)) read_inq_activity_rp;
+
+#define READ_INQ_ACTIVITY_RP_SIZE 5
+
+#define OCF_WRITE_INQ_ACTIVITY 0x001E
+typedef struct {
+       uint16_t         interval;
+       uint16_t         window;
+} __attribute__ ((packed)) write_inq_activity_cp;
+#define WRITE_INQ_ACTIVITY_CP_SIZE 3
+
 #define OCF_READ_AUTH_ENABLE   0x001F
 #define OCF_WRITE_AUTH_ENABLE  0x0020
        #define AUTH_DISABLED           0x00
```

```
diff −ru bluez−utils−2.0−pre7.clean/tools/hciconfig.c bluez−utils−2.0−pre7/tools
/hciconfig.c
−−− bluez−utils−2.0−pre7.clean/tools/hciconfig.c        Fri Mar  8 22:12:35 2002
+++ bluez−utils−2.0−pre7/tools/hciconfig.c      Tue Mar 19 12:12:09 2002
@@ −38,6 +38,7 @@
 #include <sys/ioctl.h>
 #include <sys/socket.h>
 #include <asm/types.h>
+#include <asm/byteorder.h>

 #include <bluetooth.h>
 #include <hci.h>
@@ −377,6 +378,70 @@
                ver.manufacturer);
 }

+void cmd_inq_parms(int ctl, int hdev, char *opt)
+{
+       struct hci_request rq;
+       int s;
+       if ((s = hci_open_dev(hdev)) < 0) {
+               printf("Can't open device hci%d. %s(%d)\n", hdev, strerror(errno
), errno);
+               exit(1);
+       }
+
+       memset(&rq, 0, sizeof(rq));
+
+       if (opt) {
+         uint16_t window, interval;
+         if (sscanf(opt,"%4u/%4u", &window, &interval)==2) {
+           write_inq_activity_cp cp;
+
+           rq.ogf = OGF_HOST_CTL;
+           rq.ocf = OCF_WRITE_INQ_ACTIVITY;
+           rq.cparam = &cp;
+           rq.clen = WRITE_INQ_ACTIVITY_CP_SIZE;
+
+           cp.window = __cpu_to_le16(window);
+           cp.interval = __cpu_to_le16(interval);
+
+           if (window<0x12 || window>0x1000)
+             printf("Warning: inquiry window out of range!\n");
+
+           if (interval<0x12 || interval>0x1000)
+             printf("Warning: inquiry interval out of range!\n");
+
+           if (hci_send_req(s, &rq, 1000) < 0) {
+             printf("Can't set inquiry parameters name on hci%d. %s(%d)\n",
+                    hdev, strerror(errno), errno);
+             exit(1);
+             }
+         } else {
+           printf("Unkown argument for command, remeber: one integer/one intege
r NO SPACES and one \"/\"\n");
+         }
+       } else {
+         uint16_t window, interval;
+         read_inq_activity_rp rp;
+
+         rq.ogf = OGF_HOST_CTL;
+         rq.ocf = OCF_READ_INQ_ACTIVITY;
```



```
+         rq.rparam = &rp;
+         rq.rlen = READ_INQ_ACTIVITY_RP_SIZE;
+
+         if (hci_send_req(s, &rq, 1000) < 0) {
+           printf("Can't read inquiry parameters on hci%d. %s(%d)\n",
+                  hdev, strerror(errno), errno);
+           exit(1);
+         }
+         if (rp.status) {
+           printf("Read inquiry parameters on hci%d returned status %d\n",
, rp.status);
+           exit(1);
+         }
+         print_dev_hdr(&di);
+
+         window = __le16_to_cpu(rp.window);
+         interval = __le16_to_cpu(rp.interval);
+         printf("\tInquiry interval: %u slots (%.2f ms), window: %u slots (
 ms)\n", interval, interval*0.625, window, window*0.625);
+       }
+}
+
 void print_dev_hdr(struct hci_dev_info *di)
 {
        static int hdr = −1;
@@ −445,6 +510,7 @@
        { "lp",      cmd_lp,      "[policy]", "Get/Set default link policy" },
        { "name",    cmd_name,    "[name]",   "Get/Set local name" },
        { "class",   cmd_class,   "[class]",  "Get/Set class of device" },
+       { "inqparms",cmd_inq_parms, "[int/win]","Get/Set device inquiry wind
nd iterval" },
        { "version",    cmd_version, 0,  "Display version information" },
        { "features",   cmd_features, 0,"Display device features" },
        { NULL, NULL, 0}
```

```
--- ../../pis/bluez-2.0-pre6/tools/hcitool.c    Wed Feb 20 19:07:17 2002
+++ hcitool.c   Thu Feb 28 23:21:45 2002
@@ -43,11 +43,17 @@
 #include <hci.h>
 #include <hci_lib.h>

+#include <getopt.h>
+#include <ctype.h>
+
 extern int optind,opterr,optopt;
 extern char *optarg;

 static int ctl;

+/* Global option vars.. Should probably be named something more hinting in that
 regard... */
+int num_rsp = 0, flags=0, length=10;
+
 static int for_each_dev(int flag, int(*func)(int d, long arg), long arg)
 {
        struct hci_dev_list_req *dl;
@@ -173,22 +179,13 @@
 static void cmd_inq(int dev_id, char **opt, int nopt)
 {
        inquiry_info *info;
-       int i, num_rsp = 0, length, flags;
+       int i;
        bdaddr_t bdaddr;

        if (dev_id < 0)
                dev_id = get_route(NULL);

-       if (nopt >= 1)
-               length = atoi(opt[0]);
-       else
-               length = 10; /* 10 seconds */
-
-       flags = 0;
-       if (nopt >= 2)
-               flags |= !strncasecmp("f", opt[1], 1) ? IREQ_CACHE_FLUSH : 0;
-
-       printf("Inquiring ...\n");
+       printf("Inquiring on device %i...\n", dev_id);
        info = hci_inquiry(dev_id, length, &num_rsp, NULL, flags);

        if (!info) {
@@ -304,13 +301,21 @@
        char *doc;
 } command[] = {
        { "dev",  cmd_dev,  0,          "Display local devices"      },
-       { "inq",  cmd_inq,  "[lenght] [flush]", "Inquire remote devices"     },
+       { "inq",  cmd_inq,  "--length, -l inq timeout   Inquire remote devices\n
            --resps, -r stop after n responses\n            --flush, -f flush
known device cache", " "},
        { "con",  cmd_con,  0,          "Display active connections" },
        { "cc",   cmd_cc,   "<bdaddr> [pkt type] [role]", "Create connection to
remote device" },
        { "dc",   cmd_dc,   "<bdaddr>", "Disconnect from remote device" },
        { NULL, NULL, 0}
 };

+struct option option_desc[] = {
```



```
+  {"i", 1, 0, 'i'},
+  {"length", 1, 0, 'l'},
+  {"resps", 1, 0, 'r'},
+  {"flush", 0, 0, 'f'},
+  {"help", 0, 0, 'h'}
+};
+
 static void usage(void)
 {
        int i;
@@ -327,21 +332,55 @@

 int main(int argc, char *argv[], char *env[])
 {
-       int opt, i, dev_id = -1;
-       char *dev;
+       int opt, i=0, dev_id = -1;
+       char *dev, *endptr;

-       while ((opt=getopt(argc, argv, "i:h")) != EOF) {
-               switch(opt) {
-               case 'i':
-                       dev    = strdup(optarg);
-                       dev_id = atoi(dev + 3);
-                       break;
-
-               case 'h':
-               default:
-                       usage();
-                       exit(0);
-               }
+       while ((opt=getopt_long(argc, argv, "i:r:l:fh",option_desc, &i))>0)
+         switch(opt) {
+         case 'i':
+           if (!(strncmp("hci", optarg, 3)) && strlen(optarg)>=4
+               && isdigit(optarg[3])) {
+             dev    = strdup(optarg);
+             dev_id = (int) strtol(dev + 3, &endptr,10);
+             if (endptr==NULL || *endptr==' ' || *endptr == '\0') {
+               break;
+             }
+           }
+           printf("Wrong argument for option -i\n");
+           usage();
+           exit(1);
+         case 'l':
+           length = (int) strtol(optarg, &endptr,10);
+           if (endptr!=NULL && *endptr!=' ' && *endptr!='\0') {
+             printf("Wrong argument for option -l\n");
+             usage();
+             exit(1);
+           }
+           break;
+         case 'r':
+           num_rsp = (int) strtol(optarg, &endptr, 10);
+           if (endptr!=NULL && *endptr!=' ' && *endptr!='\0') {
+             printf("Wrong argument for option -r\n");
+             usage();
+             exit(1);
+           }
+           break;
+         case 'f':
```

```
+           flags |= IREQ_CACHE_FLUSH;
+           break;
+         case ':':
+           printf("Argument missing\n");
+           usage();
+           exit(0);
+         case '?':
+           printf("Unknown syntax\n");
+           usage();
+           exit(0);
+         case 'h':
+         default:
+           usage();
+           exit(0);
+         }
        }

        if (argc - optind < 1) {
```

Appendix H **bluez−utils−2.0−pre9.bidirectional.patch**

```
--- bluez-utils-2.0-pre9.org/tools/l2test.c     Wed Mar 27 06:30:35 2002
+++ bluez-utils-2.0-pre9/tools/l2test.c Sun May 26 23:52:43 2002
@@ -54,7 +54,9 @@
        DUMP,
        CONNECT,
        CRECV,
-       LSEND
+       LSEND,
+       BILISTEN,
+       BICONNECT
 };

 unsigned char *buf;
@@ -74,6 +76,13 @@
 int auth = 0;
 int encrypt = 0;

+
+uint32_t verify_data(unsigned char *buf, int r, uint32_t seq);
+void bi_send_recv(int s);
+
+unsigned char *bi_recv_buf;
+int bi_recv_count=10000;
+
 float tv2fl(struct timeval tv)
 {
        return (float)tv.tv_sec + (float)(tv.tv_usec/1000000.0);
@@ -246,9 +255,7 @@
                gettimeofday(&tv_beg,NULL);
                total = 0;
                while (total < data_size) {
-                       uint32_t sq;
-                       uint16_t l;
-                       int i,r;
+                       int r;

                        if ((r = recv(s, buf, data_size, 0)) <= 0) {
                                if (r < 0)
@@ -257,26 +264,7 @@
                                return;
                        }

-                       /* Check sequence */
-                       sq = btohl(*(uint32_t *)buf);
-                       if (seq != sq) {
-                               syslog(LOG_INFO, "seq missmatch: %d -> %d", seq,
 sq);
-                               seq = sq;
-                       }
-                       seq++;
-
-                       /* Check length */
-                       l = btohs(*(uint16_t *)(buf+4));
-                       if (r != l) {
-                               syslog(LOG_INFO, "size missmatch: %d -> %d", r,
l);
-                               continue;
-                       }
-
-                       /* Verify data */
-                       for (i=6; i < r; i++) {
-                               if (buf[i] != 0x7f)
```

Appendix H **bluez−utils−2.0−pre9.bidirectional.patch** Pag

```
-                                       syslog(LOG_INFO, "data missmatch: by
d 0x%2.2x", i, buf[i]);
-                       }
+                       seq = verify_data(buf, r, seq);

                        total += r;
                }
@@ -311,6 +299,101 @@
        }
 }

+void bi_connect_mode(char* svr) {
+  int s;
+  if( (s = do_connect(svr)) < 0 )
+    exit(1);
+  bi_send_recv(s);
+}
+
+void bi_send_recv(int s) {
+  int i, r;
+  long total;
+  uint32_t recv_seq, send_seq;
+  struct timeval tv_beg, tv_end, tv_diff;
+  fd_set rset, wset;
+
+  /* Incoming packets are limited by the -I option */
+  if (!(bi_recv_buf = malloc(imtu))) {
+    perror("Can't allocate data buffer");
+    exit(1);
+  }
+  for(i=6; i < imtu; i++) buf[i]=0x7f;
+
+  send_seq = 0;
+  recv_seq = 0;
+  total = 0;
+  FD_ZERO(&rset);
+  FD_ZERO(&wset);
+  gettimeofday(&tv_beg,NULL);
+  while(1) {
+    FD_SET(s, &rset);
+    FD_SET(s, &wset);
+    if (select(s+1, &rset, &wset, NULL, NULL)<0){
+      syslog(LOG_ERR, "Select failed. %s(%d)", strerror(errno), errno);
+      exit(-1);
+    }
+
+    if(FD_ISSET(s, &wset)) {
+      *(uint32_t *)buf = htobl(send_seq++);
+      *(uint16_t *)(buf+4) = htobs(data_size);
+
+      if( send(s, buf, data_size, 0) <= 0 ) {
+       syslog(LOG_ERR, "Send failed. %s(%d)", strerror(errno), errno);
+       exit(1);
+      }
+    }
+    if(FD_ISSET(s, &rset)) {
+      if ((r = recv(s, bi_recv_buf, imtu, 0)) < 0) {
+       if (r < 0)
+         syslog(LOG_ERR, "Read failed. %s(%d)", strerror(errno), errno);
+       return;
+      }
+      recv_seq = verify_data(bi_recv_buf, r, recv_seq);
```

```
+      total += r;
+
+      if(total >= bi_recv_count) {
+       gettimeofday(&tv_end,NULL);
+       timersub(&tv_end,&tv_beg,&tv_diff);
+       syslog(LOG_INFO,"Received %ld bytes in %.2f sec, %.2f kB/s", total,
+              tv2fl(tv_diff), (float)(total / tv2fl(tv_diff) ) / 1024.0);
+       tv_beg=tv_end;
+       total=0;
+      }
+    }
+  }
+}
+
+uint32_t verify_data(unsigned char* buf, int r, uint32_t seq){
+  uint32_t sq;
+  uint16_t l;
+  int i;
+
+  /* Check sequence */
+  sq = btohl(*(uint32_t *)buf);
+  if (seq != sq) {
+    printf("seq missmatch: %d -> %d", seq, sq);
+    syslog(LOG_INFO, "seq missmatch: %d -> %d", seq, sq);
+    seq = sq;
+  }
+  seq++;
+
+  /* Check length */
+  l = btohs(*(uint16_t *)(buf+4));
+  if (r != l) {
+    syslog(LOG_INFO, "size missmatch: %d -> %d", r, l);
+    goto done;
+  }
+
+  /* Verify data */
+  for (i=6; i < r; i++) {
+    if (buf[i] != 0x7f)
+      syslog(LOG_INFO, "data missmatch: byte %d 0x%2.2x", i, buf[i]);
+  }
+ done:
+  return seq;
+}
+
 void reconnect_mode(char *svr)
 {
       while(1){
@@ -361,7 +444,9 @@
                "\t-u connect and receive\n"
                "\t-n connect and be silent\n"
                "\t-c connect, disconnect, connect, ...\n"
-               "\t-m multiple connects\n");
+               "\t-m multiple connects\n"
+               "\t-x biderectional send/recv - listen\n"
+               "\t-y biderectional send/recv - connect\n");

        printf("Options:\n"
               "\t[-b bytes] [-S bdaddr] [-P psm]\n"
@@ -381,7 +466,7 @@

        mode = RECV; need_addr = 0;
```



```
-       while ((opt=getopt(argc,argv,"rdscuwmnb:P:I:O:S:MAE")) != EOF) {
+       while ((opt=getopt(argc,argv,"rdscuwmnxyxb:P:I:O:S:MAE")) != EOF) {
                switch(opt) {
                case 'r':
                        mode = RECV;
@@ -424,6 +509,15 @@
                        data_size = atoi(optarg);
                        break;

+               case 'x':
+                       mode=BILISTEN;
+                       break;
+
+               case 'y':
+                       mode=BICONNECT;
+                       need_addr = 1;
+                       break;
+
                case 'S':
                        baswap(&bdaddr, strtoba(optarg));
                        break;
@@ -476,6 +570,14 @@
        openlog("l2test", LOG_PERROR | LOG_PID, LOG_LOCAL0);

        switch( mode ){
+               case BILISTEN:
+                       do_listen(bi_send_recv);
+                       break;
+
+               case BICONNECT:
+                       bi_connect_mode(argv[optind]);
+                       break;
+
                case RECV:
                        do_listen(recv_mode);
                        break;
```