Custom Coding, Adaptive Rate Control, and Distributed Detection for Bluetooth

Matthew C. Valenti
Lane Dept. of Comp. Sci. & Elect. Eng.
West Virginia University
Morgantown, WV 25506-6109
mvalenti@wvu.edu

Max Robert
Mobile and Portable Radio Research Group
Virginia Tech
Blacksburg, VA 24061-0350
probert@vt.edu

***Abstract*— Three strategies for improving the performance of point-to-point Bluetooth links are presented. All of the strategies are implemented on the host computer, and therefore no modification of the Bluetooth standard is necessary. The first strategy is custom error control coding, which is achieved by transporting BCH code words within AUX1 packets. The second concept is adaptive rate control, which involves dynamically selecting the packet type that offers the best throughput for the current channel SNR. Finally, a distributed detection technique is proposed, whereby each packet is broadcast to a group of two or more receivers that are linked over a reliable backbone, and the packet is accepted if it is received correctly at any receiver in the group.**

I. Introduction

In Bluetooth, data is normally transported using one of six predefined asynchronous connection-less (ACL) packets [1]. Error control is achieved through the use of an error detecting cyclic redundancy check (CRC) code along with a simple stop-and-wait automatic repeat request (ARQ) protocol. Furthermore, the three DMx packets use a (15,10) shortened Hamming code for forward error correction (FEC) prior to error detection. In addition to the time diversity provided by the error control mechanism, frequency diversity is achieved by the use of slow frequency hopping (FH). In particular, each piconet is tuned to the same frequency for the entire duration of a packet, but then changes to a different frequency each time a new packet is transmitted or an erroneous packet is retransmitted. Since the fading and interference in the new frequency channel is likely to be significantly different than that of the previous one, the use of FH with ARQ provides an effective method of diversity at reasonably high signal to noise ratio (SNR) [2].

While the built-in error control mechanisms perform well when the average SNR is high, the throughput at low average SNR quickly falls to zero. The modest performance at low average SNR is a consequence of the poor error correction capability of Hamming codes, the inability of the system to automatically adapt to meet changing channel conditions, and the lack of available spatial diversity. However, these shortcomings can be alleviated by performing custom error control on the host computers, rather than using the standard error control mechanism available on the Bluetooth devices.

This work was supported by the Office of Naval Research under grant N00014-00-0655 and by the Bradley and AOL Wireless Home Networking Technologies Fellowships.

In this paper, several custom error control strategies are presented that directly address the shortcomings of Bluetooth's default error control mechanism. While these strategies differ from the more straightforward methods implied by the Bluetooth standard, they are all performed on the host computers and therefore do not require any modification to the Bluetooth standard. The first strategy proposed in this paper is *custom FEC coding*, which involves utilizing the user-defined 'AUX1' packet to transport codes that are more powerful than the Hamming codes used by DMx packets. The second strategy is *adaptive rate control*, which involves dynamically changing the packet type (and, if custom coding is used, the rate of the FEC code) to match the instantaneous SNR of the channel. Finally, the third proposed strategy is *distributed detection*, which involves broadcasting the packet to a group of two or more receivers that are linked over a reliable backbone.

The remainder of this paper is organized as follows: Section II reviews and improves the analysis of the throughput of Bluetooth originally presented in [2]. Section III explains how to implement custom coding in Bluetooth and presents the corresponding analysis for BCH encoded packets. Section IV discusses bounds on the performance of adaptive rate control under the assumption of perfect channel knowledge. Section V describes strategies for implementing distributed detection and summarizes the associated analysis. Finally, conclusions are drawn in Section VI.

II. Improved Throughput Analysis

The simple stop-and-wait ARQ protocol used by Bluetooth follows the state transition diagram shown in Fig. 1. In this figure, ϵ_p is the probability that the payload data is accepted, while ϵ_h is the probability that only the packet header is accepted. In order for the payload data to be accepted, three conditions must be met: (1) The receiver must synchronize with the 72 bit access code, (2) the 54 bit packet header must be correctly decoded, and (3) the payload data must be either entirely correct (if an uncoded DHx packet) or correctly decoded (if a coded DMx packet). In order for the packet header to be accepted, only conditions (1) and (2) must be met. In [2], the probabilities of conditions (1) and (2) are given by Equations (3) and (4), respectively, while the probability of condition (3) is given for DHx packets by Equation (5) and for DMx packets by Equation (6).

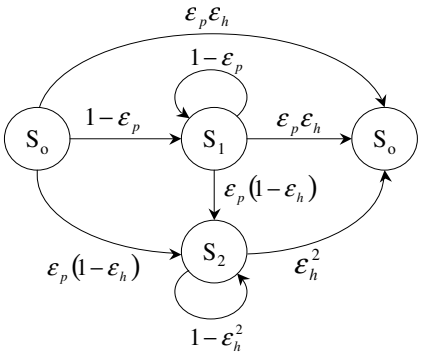

Fig. 1. State diagram representation of Bluetooth ARQ scheme.

The system starts in state S_o by transmitting a new packet. If at any time both the payload data of the forward packet and the packet header of the return packet (which contains the ACK flag) are accepted, then the system returns to state S_o and transmits the next packet. If the forward payload data is not accepted, then the system moves to state S_1 and the packet is retransmitted. The system will remain in state S_1 until the receiver accepts the payload data of the forward packet. If at any time the receiver accepts the payload data of the forward packet but the return packet header is not accepted, then the system will move to state S_2. While in state S_2, the receiver will only examine the header of the forward packet (since it has already correctly decoded the payload data) and thus the system will only move back to state S_o if the headers of both the forward and return packets are accepted (it does not matter if the payload data of the forward packet is correct).

The average number of transmissions can be found by analyzing Fig. 1. Each branch must be multiplied by the variable T, which indicates the amount of time required to make a state transition [3]. Next, Mason's gain rule is used to obtain the graph's generating function:

$$G(T) = \frac{\epsilon_p\epsilon_h T\left[1-T(1-\epsilon_h)\right]}{\left[1-T(1-\epsilon_p)\right]\left[1-T(1-\epsilon_h^2)\right]} \quad (1)$$

Finally, the average number of state transitions required for the system to move from state S_o back to state S_o is found by taking the partial derivative of $G(T)$ with respect to T and setting $T = 1$

$$\bar{N} = \frac{\partial}{\partial T}G(T)\bigg|_{T=1} = \frac{\epsilon_p + \epsilon_h^2 - \epsilon_p\epsilon_h}{\epsilon_p\epsilon_h^2} \quad (2)$$

The throughput can then be found from (2) by using:

$$\bar{R} = \frac{K}{\tau D\bar{N}} \quad (3)$$

where K is the number of payload data bits per packet, D is the number of slots occupied by the forward packet and its corresponding return packet (either 2,4, or 6), and $\tau = 625 \times 10^{-6}$ is the duration of a slot (in seconds).

Note that the above analysis differs slightly from the simpler analysis presented in [2], where $\bar{N} = 1/(\epsilon_f\epsilon_h)$. This is because the analysis in [2] did not separate states S_1 and S_2. In other words, when the forward payload data was accepted but the return packet header was not accepted, then the payload data of the next packet would have to be accepted in order for the system to return to state S_o. This is in contrast to the present analysis, where the forward payload data is not re-examined once it is received correctly.

III. Custom Coding in Bluetooth

The key to achieving custom FEC coding in Bluetooth is the AUX1 packet, which is a seventh ACL packet that is often overlooked. When the AUX1 packet is used, the Bluetooth device does not encode the payload (for either forward error correction or for error detection) and ARQ is turned off. At the destination, the Bluetooth device simply passes the received bits up to the host regardless of whether they are correct or not. Thus, while the six standard ACL packets provide a reliable link with random delay (which approaches infinity at low SNR), the AUX1 packet provides an unreliable bit pipe with deterministic delay (one slot).

Therefore, custom coding can be achieved by encoding the packet on the host itself. The packet must be first encoded by a CRC code for error detection and then by any appropriate FEC code. In a related paper, we considered using turbo codes [4] but found that the lack of soft information and relatively small packet sizes prevented the full potential of turbo codes from being realized. In this paper, we suggest to simply use BCH codes, which are more appropriate for a hard decision decoded channel.

At the destination, the Bluetooth device passes the received AUX1 packet up to the host. Because ARQ is turned off, the host will receive a set of unreliable hard bit decisions which are then passed through a FEC decoder. After FEC decoding, the outer CRC code can be used to detect any remaining errors. If the CRC check passes, then the packet will be accepted by the host; otherwise a retransmission must be requested. Note that since ARQ is turned off on the Bluetooth device, it must now be implemented by the hosts (in the application layer).

The AUX1 payload consists of a 1 byte payload header and 29 bytes of payload data. Custom coding can be implemented by transporting (in the payload data field) a (232,k) BCH code obtained by shortening a (255,k+23) BCH code. The input to the BCH encoder is a combination of the source data and a 2 byte CRC (we use the same CRC code used by the other ACL packets for error detection) and thus the number of source data bits per AUX1 packet is K=k-16.

The analysis is identical to that used for the other six packet types, with the exception of how the probability of correctly decoded payload data is computed (i.e. condition (3) given in Section II). For BCH codes capable of correcting t errors, the probability of a correctly decoded payload is

$$\sum_{k=0}^{t}\left(\begin{array}{c}232\\k\end{array}\right)p^k(1-p)^{232-k} \quad (4)$$

This probability is multiplied by both the probability of synchronization and the probability of a correct packet header to

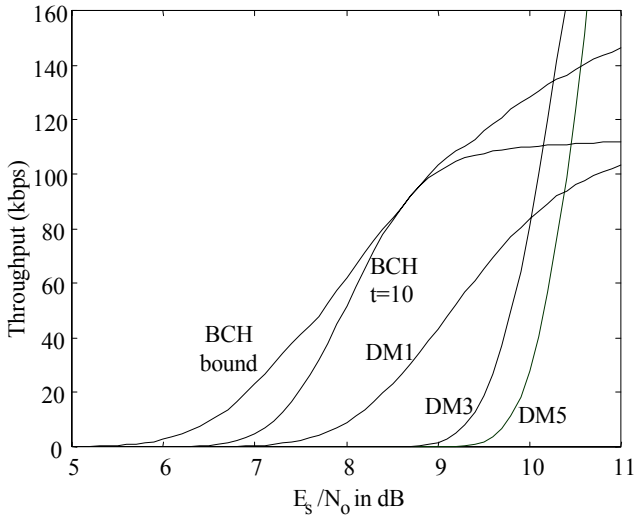

Fig. 2. Average throughput of BCH encoded AUX1 packets in AWGN.

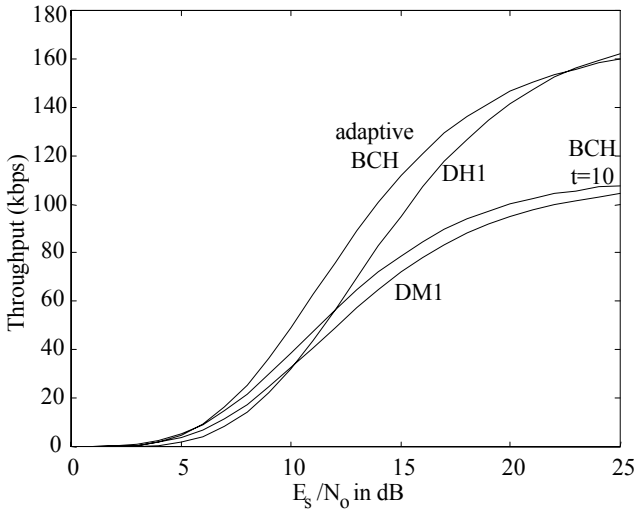

Fig. 3. Average throughput of adaptive BCH encoded AUX1 packets in quasi-static Rayleigh fading.

obtain ϵ_p, the probability that the payload data is accepted. Tables relating k to t can be readily found in the literature (see, for instance, [5]).

Fig. 2 compares the throughput (in AWGN) of an AUX1 packet carrying a (232,156) BCH code (which has $t = 10$) against the throughput of the three ACL packets protected by Hamming codes (DM1, DM3, and DM5). These curves were produced under the assumption that six errors could be tolerated in the access code (i.e. $T = 6$ in Equation (2) of [2]). This particular custom error control code has a gain of about 1 dB over the DMx packets at throughput below 100 kbps. In addition, the throughput for all suitable BCH codes with $1 \leq t \leq 43$ was calculated. Rather than plotting all of these codes independently, a single curve is presented in Fig. 2 which gives the maximum throughput over all codes at each value of SNR. This curve can be considered to be an upper bound on throughput when using BCH codes. As indicated, an additional coding gain (about 0.5 dB) at low SNR (below 50 kbps) could be realized by switching to a different BCH code with higher t.

IV. Adaptive Rate Control

As implied by Fig. 2, the value of t that maximizes throughput, and likewise the optimal value of the code rate, depends on the channel SNR. However, in a frequency hopping system such as Bluetooth, the SNR is likely to change significantly from hop to hop. Frequency hopping systems with a variable SNR are often modeled as quasi-static fading channels, whereby the SNR is constant for the duration of a hop but changes from hop to hop according to some statistical distribution. In quasi-static Rayleigh fading, the envelope of the received signal varies from hop to hop according to a Rayleigh distribution.

Thus, additional throughput gains in quasi-static fading channels can be achieved by adapting the rate of the BCH code to match the prevailing channel conditions. The transmitter could select the value of t based on the current channel SNR, always sending the BCH code which maximizes throughput. From an information theoretic perspective, this is equivalent to water-filling. However, such a method would require that the transmitter be able to accurately predict the SNR of the channel, which could be a difficult task in practice. However, since Bluetooth only uses a finite number of channels (79 to be exact), the same channel is revisited every $79\tau = 49.375$ msec. Given the low mobility of typical Bluetooth applications, it may be possible to track each of the channels by using past knowledge of the SNR of each channel to predict the future SNR. Of course, any SNR prediction algorithm would need to take into account the statistics of the interference, which could be quite unpredictable.

Fig. 3 illustrates the potential benefits of using adaptive rate control in a quasi-static Rayleigh fading channel under the assumption that the channel SNR is perfectly known by the transmitter. In order to provide a fair comparison, only packets that are one slot long are considered. Four curves are shown corresponding to (1) the DM1 packet, (2) the DH1 packet, (3) nonadaptive BCH, that is an AUX1 packet transporting a $t = 10$ BCH code, and (4) adaptive BCH, that is an AUX1 code carrying a BCH code whose value of t is chosen to match the SNR of the current slot. While the nonadaptive $t = 10$ BCH code provides a modest throughput gain at low average SNR (below 12 dB), for higher average SNR, the DH1 packet offers superior throughput. This is because at higher average SNR, the channel is likely to have a high instantaneous SNR, and thus the overhead used by the BCH code is wasted.

The performance when using adaptive BCH coding is much improved. In fact, the adaptive BCH coded system outperforms all the other single slot systems for average SNR below 22 dB. The superior performance of the adaptive system is due to the ability of the transmitter to automatically choose powerful low rate codes for slots with low SNR and bandwidth efficient high rate codes for slots with high SNR. While these gains are promising, it should be noted that the maximum gain is only about 1.5 dB, and the gain achieved in practice is likely to be smaller when channel prediction is taken into account.

At high SNR, multi-slot packets provide higher throughput

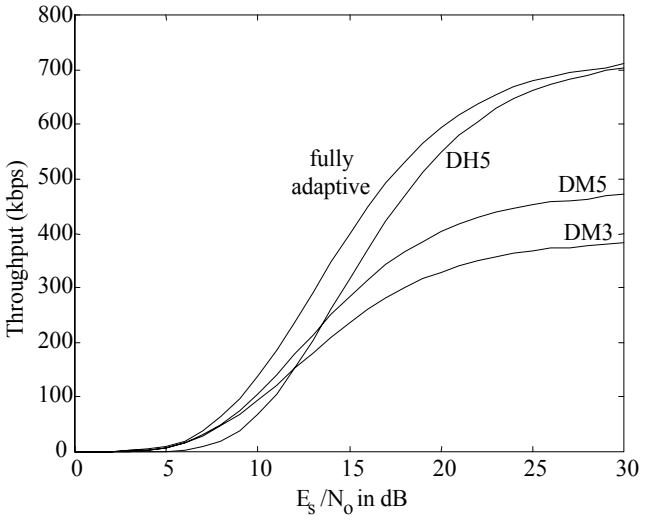

Fig. 4. Average throughput in quasi-static Rayleigh fading of a fully adaptive system that can select among all 6 ACL packets and any custom BCH coded AUX1 packet.

than single-slot packets [2]. Thus, the system could be further improved by allowing it to not only select variable-rate BCH-encoded AUX1 packets when the SNR is low, but also to select multi-slot (DM3, DM5, or DH5) packets when the SNR is high. In Fig. 4, the performance is shown for a "fully adaptive" system that adapts the type of packet on a hop-by-hop basis in order to maximize the throughput for the instantaneous SNR of each hop. The packet could be a BCH-encoded AUX1 packet, or it could be one of the six standard ACL packets (although DM1, DH1, and DH3 are never selected because they never have maximum throughput at any instantaneous SNR). As can be seen, such a system provides throughput which is higher than any one particular packet type, and a coding gain of about 1.5 dB is achieved over the conventional ACL packets at moderate bit rates (200-700 kbps). Interestingly, very little of the gain is due to using custom BCH codes, as similar (within 0.1 dB) performance can be achieved by simply selecting among DM1, DM3, DM5, and DH5[1]. Thus, there appears to be some benefit to adapting the packet type. It should be noted, however, that while the Bluetooth standard does not necessarily prohibit the adaptation of the packet type on a hop-by-hop basis, many Bluetooth implementations require that the packet type be specified when the connection is established and cannot handle requests to change the packet type. Thus, this approach might not be applicable for all Bluetooth implementations.

V. Distributed Detection

Bluetooth piconets typically contain two main categories of devices. The first type of device, which we call "class A", is a device that is connected to a high capacity network and a continuous power source, as typified by a desktop computer with a wired Internet connection and access to AC power from the wall. The second type of device, which we call "class B", is an untethered, low-power unit typical of a cell phone, PDA, or wireless sensor. A key distinction between these two classes is that power conservation is much more important for class B devices than for class A devices.

Consider a case in which a class B device wishes to access the Internet. In a conventional system, it connects to a single class A device which serves as a bridge to the Internet. However, in some scenarios, there may be more than just one class A device within the class B device's range. In such a situation, it makes sense to exploit the inherent macrodiversity by allowing the class A device to simultaneously communicate with all class B devices that are within range. By performing distributed detection among the multiple class B devices, it is possible to reduce the amount of energy required for the class B device to transmit its data.

A simple example of how to implement distributed detection is as follows. Let there be a single class B master device and a pair of class A slaves. The master broadcasts a packet, which is received by both slaves. Each slave must synchronize with the access code, decode the packet header, decode the payload data, and perform a CRC check. If the CRC check passes at either slave, then the packet can be forwarded to the Internet. Otherwise, the master will need to rebroadcast the packet.

Because there are now two destination radios, ARQ becomes a bit tricky. The simplest implementation would have both slaves send replies back to the master. The replies would be transmitted during subsequent slave slots, and the master slot between the two slave slots would go unused. Because of the extra overhead involved in transmitting ACKs from two slaves, this strategy would only improve overall throughput if the data is broadcast by the master in the form of a long (DM5 or DH5) packet. If the master transmits a 5-slot packet, then a total of 8 slots will be required for the forward Dx5 packet and pair of replies (in contrast to 6 slots when only one slave must reply).

A retransmission is now only necessary if the packet is not accepted by either slave, or if is accepted by one (or both) of the slaves, but the packet header of the corresponding reply message(s) is not accepted by the master. Thus, the probability of an accepted packet is

$$\epsilon \;=\; 1-\left(1-\epsilon_{p_1}\epsilon_{h_1}\right)\left(1-\epsilon_{p_2}\epsilon_{h_2}\right), \qquad (5)$$

where ϵ_{p_i} is the probability that the payload data is accepted at slave i and ϵ_{h_i} is the probability that the packet header from slave i is accepted by the master. If the payload data of retransmitted packets must be correctly decoded, then the average number of retransmissions would be $\bar{N} = 1/\epsilon$. Performance could be improved by exploiting the reliable network that connects them in such a way that only one ACK is needed. For instance, the first slave could tell the second slave if it received the data correctly, which in turn would only signal an ACK back to the master if it also did not receive the data correctly.

Fig. 5 shows the performance of the proposed diversity detection technique in quasi-static Rayleigh fading for the case that there are two slaves receiving a transmission from a single master. It is assumed that the receivers have the same average SNR and that they communicate through the reliable backbone in such a way that only one return packet is required. The diversity detection technique can use either DM5 or DH5 pack-

[1]Throughput curves were also computed for a system that only used standard ACL packets, but because the performance is within 0.1 dB of the curve marked "fully adaptive", it is not shown.

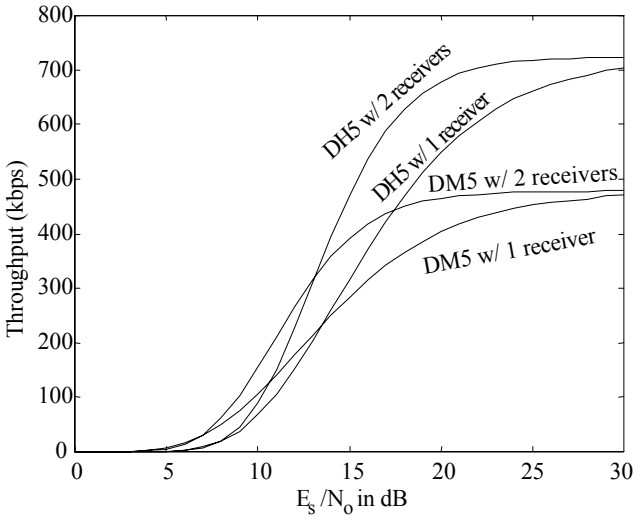

Fig. 5. Average throughput in quasi-static Rayleigh fading of a diversity reception system using two slaves.

ets, and throughput results are compared for each of these two packet types with the performance achieved in a conventional (single receiver) system. As can be seen from the figure, gains of up to 8 dB (at 700 kbps) can be realized by using this system. It should be noted that unlike the adaptive coding technique proposed in the previous section, this gain is achieved without requiring a prediction of the channel SNR.

VI. Conclusion

Several techniques were proposed for improving the throughput of a Bluetooth data link. Custom coding is the simplest of the techniques, and offers improved performance in AWGN at low SNR. Adaptive rate control provides throughput gains in quasi-static fading, but requires accurate predictions of the channel SNR and might not be supported by some Bluetooth implementations. Diversity reception also provides gains in quasi-static fading, but does so without requiring SNR prediction. However, diversity reception requires two or more slaves within range of the master and connected through a reliable backbone, and also requires slight modifications to the ARQ process.

References

[1] Bluetooth SIG, "Specification of the Bluetooth system," *Core Version 1.1*, Feb. 22, 2001.

[2] M. C. Valenti, M. Robert, and J. H. Reed, "On the throughput of Bluetooth data transmissions," in *IEEE Wireless Commun. and Networking Conf.*, (Orlando, FL), Mar. 2002.

[3] S. Wicker, *Error Control Systems for Digital Communicationsand Storage*. Englewood Cliffs, NJ: Prentice Hall, Inc., 1995.

[4] M. C. Valenti and M. Robert, "A turbo code based error control strategy for Bluetooth," in *Proc. IEEE Military Commun. Conf. (MILCOM)*, (Anaheim, CA), Oct. 2002. to appear.

[5] J. Proakis, *Digital Communications*. New York, NY: McGraw-Hill, Inc., fourth ed., 2001.