IMPROVING THE QoS OF BLUETOOTH THROUGH TURBO CODING

Matthew C. Valenti
Lane Dept. of Comp. Sci. & Elect. Engr.
West Virginia University
Morgantown, WV

Max Robert
Mobile and Portable Radio Research Group
Virginia Polytechnic Institute and State University
Blacksburg, VA

ABSTRACT

Bluetooth has emerged as a viable COTS alternative for military applications involving low power wireless networks, including wireless sensor networks and personal area networks for the foot soldier. However, the reachability of Bluetooth networks is limited by the weak error control coding used by the predefined packets. In this paper, we propose a method for improving the performance of Bluetooth by using custom error control coding in general, and turbo codes in particular. An important aspect of the proposed technique is that it is compliant with the Bluetooth standard, and therefore requires no hardware modifications. More specifically, the AUX1 user-defined packet is used to transport rate compatible punctured turbo codes (RCPT). The result, as shown by a combination of analytical/simulation results, is a dramatic increase in throughput and decrease in latency at low signal to noise ratios.

INTRODUCTION

In Bluetooth, data is transmitted using one of seven packet types that differ only in length and method of payload error control [1]. Six of the packets use automatic repeat request (ARQ) to retransmit erroneous packets, and three of these packets (DM1, DM3, and DM5) also use a (15,10) Hamming code for forward error correction (FEC). Previously [2], we analyzed the throughput performance of these six packet types. At high signal-to-noise ratio (SNR) the best throughput is achieved by using long uncoded packets, while at low SNR the best throughput is achieved using shorter packets that are protected by the Hamming code. However, the Hamming code is not particularly powerful and thus custom error control techniques that utilize more powerful codes are desirable in harsh environments.

Custom codes enable a Bluetooth connection to be maintained when the SNR is so low that a conventional connection would break. Furthermore, even at SNRs that are high enough to maintain a minimal rate connection, custom codes can simultaneously increase the throughput and decrease the average latency and latency jitter. Custom coding can be achieved in Bluetooth by using AUX1 packets, for which ARQ is disabled and thus the data is delivered to the application even if it is incorrect [1]. By using AUX1 packets, additional coding and decoding can be implemented off-chip, either in a DSP coprocessor or in middleware running on the host computer itself. The coded bits are transmitted in the payload of one or more AUX1 packets, and at the receiver, the AUX1 payload is delivered to the coprocessor or host computer for FEC decoding. Since ARQ is no longer handled by the Bluetooth device, it now must be implemented by the application layer rather than in the baseband controller.

In this paper, we explore the use of *turbo codes* for custom error control in Bluetooth. Turbo codes are among the most powerful FEC codes available, performing within 0.5 dB of the Shannon capacity limit in additive white Gaussian noise (AWGN) channels [3]. Presumably, there should be a significant benefit from using AUX1 packets to transport turbo codes over a Bluetooth data link. However, in order to achieve the full coding gain, soft-decision decoding is required and the code words must be fairly long. These two requirements pose a set of technical hurdles that must be overcome in order to transport turbo codes using commercial Bluetooth equipment. In order to compensate for the hard bit decisions delivered from the Bluetooth transceiver to the decoding agent, we recommend using the received signal strength indicator (RSSI) which provides enough additional information to allow "pseudo-soft-decision" decoding to be performed. To compensate for the short length of the AUX1 packet, we suggest using a rate compatible turbo code (RCPT) [4]. With RCPT, a long turbo code is broken up into short packets, and by using an appropriate ARQ protocol, only those packets that are necessary to reliably decode the data are (re)transmitted. In the remainder of the paper, we more fully describe the proposed approach for transporting turbo codes over Bluetooth. In addition, a combination of analysis and simulation is used to illustrate the potential gains of the proposed system.

RCPT-BASED ERROR CONTROL

A. Code Description

With RCPT codes, a low rate mother turbo code is transformed into a family of higher rate codes by puncturing the parity bits. The rate compatability criterion guarantees that the codewords of any particular rate code are embedded in all codes of lower rate. The source begins by sending the highest rate code. If errors remain after decoding, the destination stores the received information in a buffer and requests a retransmission. Rather than retransmitting the entire high rate code word again, the source will only transmit a subset of the punctured parity bits that, together with the information stored in the buffer, constitute the code word of the next lower rate code. This process, known as *incremental redundancy*, is used to continually lower the code rate until it reaches the rate of the mother code. If, after sending the lowest rate code word, there are remaining errors, then the entire mother code is sent repeatedly until either the word is correctly received or the system times out.

The proposed error control technique uses a mother rate $r = 1/3$ turbo code. Data is encoded in parallel by a pair of recursive systematic convolutional (RSC) encoders, each with memory eight and octal feedback generator (13) and feedforward generator (15). The data is broken into frames of 909 data bits, which are passed through a 16 bit cyclic redundancy check (CRC) encoder. A 3 bit tail is used to terminate the trellis of the upper RSC encoder and is appended to the 925 bit CRC code word. The resulting 928 bit sequence is encoded first by the upper encoder. Next, the sequence of data, CRC, and tail are interleaved by a 928 bit S-random interleaver with S=19 [5] and encoded by the lower encoder (a separate tail is not computed for the lower encoder). The overall output of the turbo encoder consists of 2784 code bits. Since the payload of an AUX1 packet contains a 1 byte payload header and 29 bytes of payload data [1], the entire turbo code word can be transmitted in the payload of 12 AUX1 packets.

Rather than transmitting the entire turbo code word at once, we use an incremental redundancy approach. A family of nine rate compatible turbo codes is created from the mother code by using the puncturing patterns shown in Table I, where a 0 indicates a punctured bit. The period of the puncturing pattern is eight, and the parity bits of both the upper and lower encoder are punctured using the same pattern (the systematic data is never punctured). These sequences were designed with the goal of protecting the end of the parity sequences (since there will be an integer number of puncturing periods in the turbo code word, this indicates a bias towards placing the ones near the end of the pattern).

Table 1: Puncturing patterns defining the RCPT code.

code rate	puncturing pattern
1	00000000
4/5	00000010
2/3	00000011
4/7	00000111
1/2	10000111
4/9	10100111
2/5	10101111
4/11	11101111
1/3	11111111

B. Protocol Description

At first, the source transmits only the 928 bits of data, CRC, and upper tail in the payload of four consecutive AUX1 packets. The destination stores the received data in a buffer, checks the CRC, and immediately requests a retransmission in the payload of a return AUX1 packet if the CRC check fails. If the source does not receive a positive acknowledgment (ACK) before its next transmit slot, then it will transmit a single AUX1 packet that contains only those parity bits required to drop down to the next lower code rate. At the destination, the packet of incremental redundancy is combined with the previously received information such that the net effect is as if a single code word of the current rate code had been transmitted. The code rate is continually lowered until either an ACK is received or all 12 packets constituting the full rate 1/3 turbo code have been transmitted. If the CRC still fails after the rate 1/3 code is transmitted, then it is likely that the channel is extremely poor. Since it is unlikely that codes of rate higher than 1/3 will succeed, the system will simply retransmit the 12 AUX1 packets that comprise the rate 1/3 code until either an ACK is received or a system timer is exceeded.

C. Decoder Modification

Because hard bit decisions are passed from the Bluetooth device to the decoding agent, the decoding process used is inherently based on hard-decisions. However, it is desirable to have soft information available at the decoder since soft-decision decoding has the potential to outperform hard-decision decoding [6]. In Bluetooth, a modest amount of soft-information can be delivered to the decoding agent by using the received signal strength indicator (RSSI), which is an indication of the average SNR of any particular packet. While not a true bit-by-bit soft-decision metric, the RSSI is still a useful quantity when the average signal strength remains constant for the duration of the packet, as is the case for the quasi-static fading channels that characterize Bluetooth [2].

We wish to consider two decoder implementations, one that can operate without knowledge of the channel SNR, and another that exploits knowledge of the frame-by-frame SNR (which is obtained by reading the RSSI) for improved performance. The first decoder type can be implemented using the max-log-MAP algorithm [7], which does not require estimates of the channel SNR in an AWGN channel. In this case, the received data $\{0,1\}$ is converted into polar form $\{-1,+1\}$ and fed into a standard max-log-MAP decoder. Note that because the demodulator performs hard bit decisions, the channel is actually modeled as a binary symmetric channel (BSC) and thus performance of the turbo code will be degraded relative to a true AWGN channel.

Performance can be improved by using the log-MAP algorithm [7]. However, this algorithm requires knowledge of the symbol-by-symbol SNR of the received packet. Since it is generally assumed that the SNR remains constant for the duration of any one AUX1 packet, the RSSI can be used to derive the SNR for all of the symbols in the packet. The bits at the input of the log-MAP decoder must be in log-likelihood form, which for a BSC is

$$\begin{aligned} \Lambda(s_i) &= \ln \frac{P[s_i=+1]}{P[s_i=-1]} \quad (1) \\ &= \hat{s}_i \ln\left(\frac{1-\epsilon(\gamma)}{\epsilon(\gamma)}\right), \quad (2) \end{aligned}$$

where s_i is the transmitted symbol, $\hat{s}_i = \{-1,+1\}$ is the hard decision output of the demodulator, γ is the SNR of the channel, and ϵ is the error probability of the demodulator, which is an implementation-dependent parameter.

A lower bound on ϵ for noncoherent detection can be found by considering the performance of noncoherently detected nonorthogonal full response FSK, whose performance can be found using the following set of equations [6]

$$\begin{aligned} \epsilon(\gamma) &= e^{-\gamma/2}\left\{\frac{1}{2}I_o(ab)+\sum_{k=1}^{\infty}\left(\frac{a}{b}\right)^k I_k(ab)\right\} \\ a &= \sqrt{\frac{\gamma}{2}\left(1-\sqrt{1-\rho^2}\right)} \\ b &= \sqrt{\frac{\gamma}{2}\left(1+\sqrt{1-\rho^2}\right)} \\ \rho &= \frac{\sin(2\pi h)}{2\pi h}, \quad (3) \end{aligned}$$

where h is the modulation index, which for Bluetooth satisfies $0.28 \le h \le 0.35$. This is a lower bound on error probability because it does not account for the additional losses due to the intersymbol interference (ISI) induced by the use of partial response GFSK signaling (Bluetooth uses Gaussian pulse shaping with $BT = 0.5$). However, an exact analysis requires knowledge of the receiver implementation and must take into account both predetection and postdetection filtering [8] and goes beyond the scope of this paper.

If the decoder has knowledge of the function $\epsilon(\gamma)$, which can be estimated using the above analytical expressions, by simulation, or by measurement, and the SNR γ which can be derived from the RSSI, then the reliability $|\Lambda(s_i)|$ can be computed. The reliability is multiplied by the polar hard bit decision $\{-1,+1\}$ to form the LLR, which is then passed into the standard log-MAP algorithm. Since the only soft-information available to the decoder is the average SNR over each entire AUX1 frame, performance will again be degraded relative to a true soft-decision decoder. However, in a fading environment, where the SNR may vary from one AUX1 packet to the next, the reliability is used by the decoder to place more confidence on the strong packets and less emphasis on the weak packets. This is especially important when dealing with long custom codes such as the proposed turbo code, since they will span multiple AUX1 packets.

PERFORMANCE ANALYSIS

The quality of service (QoS) can be measured using four parameters, each of which is a function of the SNR: Throughput, residual frame error rate (FER), average latency, and latency jitter. Each of these parameters can be found given the probability $P_r(r_i,\gamma)$ that a frame must be retransmitted, which is a function of the SNR γ and, for the RCPT case, a function of the code rate r_i. A frame is retransmitted if any of the following occurs: (1) The destination fails to synchronize with the 72 bit access code of the forward packet, (2) The destination fails to decode the 18 bit packet header which is protected by a triple redundancy code, (3) The destination fails to properly decode the payload, (4) The source fails to synchronize with the access code of the return packet, and (5) The source fails to decode the header of the return packet. The probability of each of these events for the six asynchronous Bluetooth packets that use ARQ is derived in [2] and not reproduced here.

In AWGN, the SNR remains constant and thus the probability of retransmission is the same from packet to packet for any particular type of payload code. It is convenient to define a geometric random variable N which enumerates the number of times a packet must be retransmitted. Since a packet will be accepted on the nth trial only if it failed on all $(n-1)$ previous transmissions yet succeeded in the nth trial, the pmf of N is

$$p_N[n] = (1-P_r(r_n,\gamma))\prod_{i=1}^{n-1} P_r(r_i,\gamma), \quad (4)$$

where r_i is the rate of the payload code for the ith transmission. For the DMx packet, $r_i = 2/3\ \forall i$ since all packets are retransmitted using the same (15,10) Hamming code. For AUX1 packets carrying the proposed turbo code, r_i will vary in accordance to the protocol described in the previous section. Specifically, $r_1 = r_2 = r_3 = 0$ since no information has

been conveyed until the fourth code word has been transmitted. Then $r_i = 4/i$ for $4 \leq i \leq 12$ in accordance with table I. Finally, $r_i = 1/3$ when $i > 12$ and $gcd(i, 12) = 12$, otherwise if $i > 12$ then $r_i = 0$.

If a packet must be transmitted N times, the associated latency is $\tau = (DN)(625 \times 10^{-6})$, where D is the number of occupied slots per transmission including the return packet. For the DM1 and uncoded AUX1 packet types $D = 2$, for the DM3 packet $D = 4$, and for the DM5 packet $D = 6$. The average latency $\bar{\tau}$ is found by taking the expected value of τ with respect to N, i.e. $\bar{\tau} = E_N\{\tau\}$. Likewise, the latency jitter σ_τ is found by computing the standard deviation of τ, i.e. $\sigma_\tau = \sqrt{E_N\{\tau^2\} - \bar{\tau}^2}$.

The throughput in bits per second is $R = K/\tau$, where K is the number of data bits in the packet. For the proposed turbo coded AUX1 packet $K = 909$, for the DM1 packet $K = 136$, for the DM3 packet $K = 968$, and for the DM5 packet $K = 1792$. The average throughput is then found by taking the expected value of R with respect to N, i.e. $R_{avg} = E_N\{R\}$.

The residual frame error rate P_f is the probability that after a finite number N_{max} of retransmissions the frame is still in error. As $N_{max} \to \infty$, $P_f = 1$ if $\min_{r_i} P_r(r_i, \gamma) = 1$, otherwise $P_f = 0$. The residual frame error rate is related to the pmf of N by

$$P_f = 1 - \sum_{n=1}^{N_{max}} P_N[n] \quad (5)$$

A combination of analysis and Monte Carlo simulation was used to generate numerical values that compare the QoS of the proposed error control technique with the QoS of the DMx packets. Whenever possible, analytical results were used. However, simulation was used to generate the probability that a turbo coded payload was not successfully decoded. The turbo code simulation ran until 100 code word errors occured for each value code rate and of $\gamma = E_s/N_o$ considered (E_s is the energy per transmitted code symbol and N_o is the one-sided noise spectral density). A maximum of ten iterations of log-MAP (perfect RSSI case) or max-log-MAP (no RSSI case) decoding was used to decode the turbo code, although the decoder halted early once the CRC indicated that there were no residual errors. The error probability of the channel was determined using (3) with $h = 0.32$ and thus the impact of the ISI due to Gaussian pulse shaping was neglected (the ISI will cause all curves to move over to the right by an equal amount and thus does not affect the *comparison* of packet types presented here). The threshold for packet synchronization was set to $T = 60$, i.e. 60 of the 72 bits in the access code must match in order to achieve synchronization [2]. The maximum retransmit time was set to 60 msec, which limits the maximum number of packet transmissions before a residual frame error is declared.

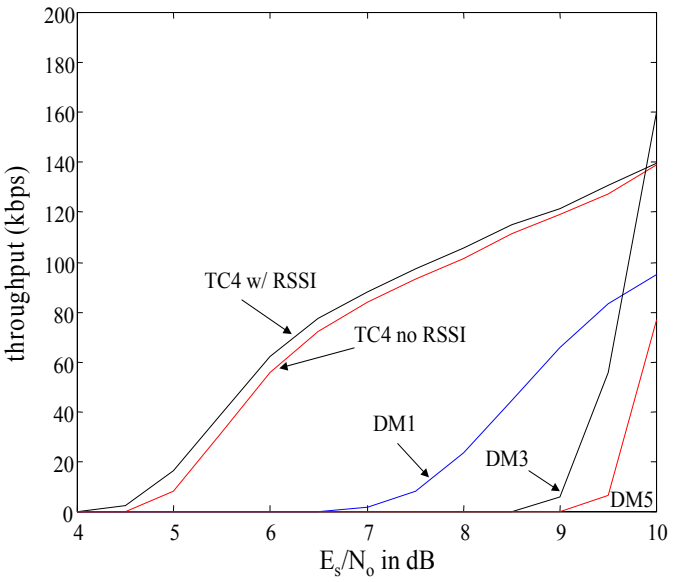

Figure 1: Comparison of the average throughput of turbo coded AUX1 packets (denoted TC4) and standard Bluetooth DMx packets in AWGN assuming noncoherent detection, modulation index h=0.32, no ISI, and synchronization threshold $T = 60$.

The QoS of the proposed turbo code based error control scheme is compared against that of the standard Bluetooth DMx packets in Fig. 1-4. In particular, Fig. 1 shows the average throughput in kbps, Fig. 2 shows the average latency in msec, Fig. 3 shows the latency jitter in msec, and Fig. 4 shows the residual frame error rate. At $E_s/N_o = 6.9$ dB, the turbo coded system can achieve a throughput of 85 kbps with an average latency of 11 msec and residual FER of less than 10^{-3} (this residual FER is acceptable since the application will know when decoded turbo code words are erroneous and thus can take more drastic countermeasures). At this SNR, the throughput of all three DMx frames is zero and thus communications at any rate is impossible. It is not until $E_s/N_o = 9.6$ dB that the DM1 packet can offer the same throughput and $E_s/N_=$8.2 dB that it can offer the same average latency. Thus, the gain due to using the turbo coded AUX1 packets at an average throughput of 85 kbps is 2.7 dB, while the gain for an average latency of 11 msec is 1.3 dB. Positive gains can be found for all average throughputs less than 138 kbps and average latencies greater than 8.2 msec. Thus use of the proposed turbo code technique has benefits in terms of the ability to maintain a link at very low SNR as well as a coding gain in terms of throughput and average latency. It should be noted that the latency jitter is similar in all cases. While performance suffers slightly by ignoring the RSSI and using the max-log-MAP algorithm, this loss is only about 0.2 dB, and thus the error control strategy is still effective even without the use of the RSSI.

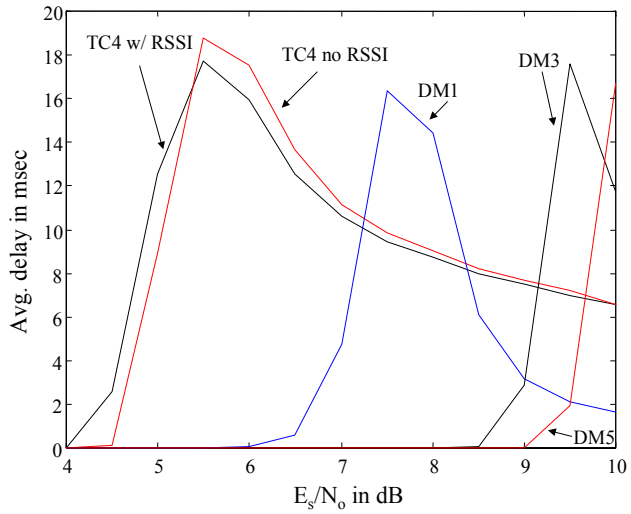

Figure 2: Average latency in AWGN.

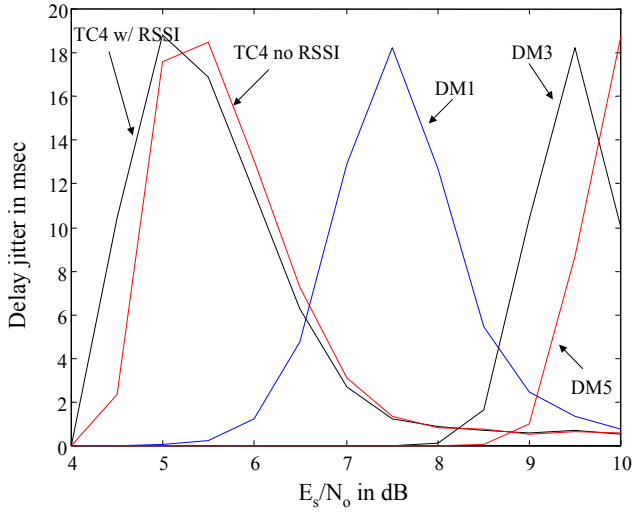

Figure 3: Latency jitter in AWGN.

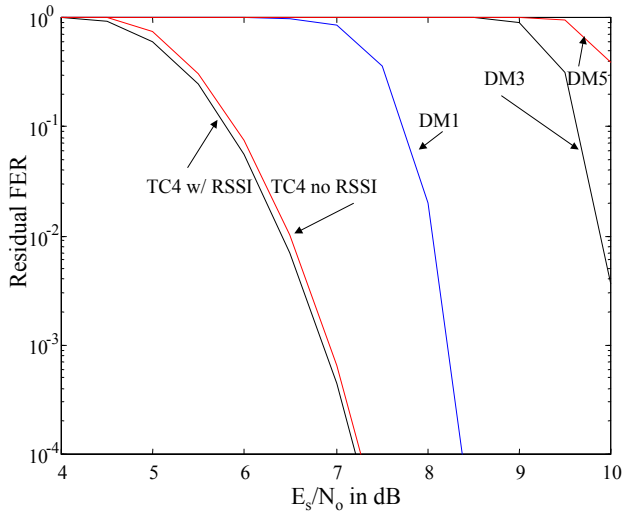

Figure 4: Residual frame error rate in AWGN with maximum retransmission time of 60 msec.

CONCLUSION

The proposed turbo code based error control strategy improves the QoS at low data rates. In particular, a coding gain of approximately 2.7 and 1.3 dB were observed at low SNR with respect to throughput and average latency, respectively. While decoding using the RSSI offers the best performance, neglecting the RSSI only introduces a 0.2 dB penalty in AWGN channels. The proposed strategy requires no modifications to the Bluetooth standard or device, and can be implemented off-chip and entirely in software.

While this paper only presents results for a particular length turbo code, we also considered shorter and longer turbo codes. In general, longer turbo codes perform better than shorter ones. Thus, we found that performance was worse for turbo codes that were shorter than the one presented here. However, longer turbo codes will require more packets, and thus the probability that one of the packets will either not be synchronized or its packet header will be incorrect increases as the length of the code increases. We found that turbo codes that were longer than the one considered in this paper actually perform worse when applied to Bluetooth. This is because the increase in interleaver gain due is less than the loss due to the increased probability that one of the headers or synchronization words fails.

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] C. Berrou, A. Glavieux, and P. Thitimasjshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes(1)," in *Proc., IEEE Int. Conf. on Commun.*, (Geneva, Switzerland), pp. 1064–1070, May 1993.

[4] D. N. Rowitch and L. B. Milstein, "On the performance of hybrid FEC/ARQ systems using rate compatable punctured turbo (RCPT) codes," *IEEE Trans. Commun.*, vol. 48, pp. 948–959, June 2000.

[5] S. Dolinar and D. Divsalar, "Weight distributions for turbo codes using random and nonrandom permutations," *JPL TDA Progress Report*, vol. 42, pp. 56–65, Aug. 15th 1995.

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

[7] P. Robertson, P. Hoeher, and E. Villebrun, "Optimal and sub-optimal maximum a posteriori algorithms suitable for turbo decoding," *European Trans. on Telecommun.*, vol. 8, pp. 119–125, Mar./Apr. 1997.

[8] M. Shimizu, N. Aoki, K. Shirakawa, Y. Tozawa, N. Okubo, and Y. Daido, "New method of analyzing BER performance of GFSK with postdetection filtering," *IEEE Trans. Commun.*, vol. 45, pp. 429–436, Apr. 1997.