THROUGHPUT PERFORMANCE OF ARQ SCHEMES BASED ON PUNCTURED TURBO CODES IN RAYLEIGH FADING CHANNELS

M.A. Kousa* and S.A. Al-Semari
Electrical Engineering Department
King Fahd University of Petroleum & Minerals, Saudi Arabia

A.H. Muqaibel
Systems & Computer Engineering Department
Carlton University, Ottawa, ON, Canada

and

A.B. Adinoyi
Electrical & Computer Engineering Department
Virginia Polytechnic Institute & State University, VA, U.S.A.

الخلاصــة

الأداء الباهر لشفرات تربو يمكن توظيفه لتحسين إنتاجية أنظمة إعادة الإرسال التلقائي ARQ. في القنوات اللاسلكية، يمكن تحسين الإنتاجية عن طريق الموائمة المستمرة بين قدرة الشفرة على تصحيح الأخطاء، وحالة القناة.

يدرس هذا البحث إنتاجية أنظمة ARQ المهجنة باستخدام شفرات تربو المثقبة. ويأتي تحسن الإنتاجية كنتيجة لتصميم مجموعات الشفرات بتوظيف أنماط تثقيب جيدة.

ABSTRACT

The astonishing performance of turbo codes can be used to enhance the throughput efficiency of data networks based on automatic report request. In wireless channels, the throughput can be maximized by adaptively matching the error correction capability of the code to the prevailing channel conditions. This paper investigates the throughput efficiency of hybrid ARQ based on punctured turbo codes. The throughput is enhanced when code sets are designed along the guidelines of good puncturing patterns.

*Address for Correspondence:
KFUPM Box 5038
King Fahd University of Petroleum & Minerals
Dhahran 31261
Saudi Arabia
e-mail: kousa@kfupm.edu.sa

M.A. Kousa, A.H. Muqaibel, A.B. Adinoyi, and S.A. Al-Semari

THROUGHPUT PERFORMANCE OF ARQ SCHEMES BASED ON PUNCTURED TURBO CODES IN RAYLEIGH FADING CHANNELS

1. INTRODUCTION

Automatic repeat request (ARQ) is one of the most powerful techniques to enhance the reliability of transmission for data networks. A basic ARQ scheme is based on error detection and retransmission. In this basic scheme, the message to be transmitted is encoded using an error detection code. The decoder checks the received message, and if no error is detected the message is delivered to the sink. A positive acknowledgement (ACK) may be sent back to confirm the successful reception of the message. In the event that error is detected, the erroneous message is discarded and a negative acknowledgement (NACK) is sent to the transmitter requesting a retransmission of the same message. The process is repeated until the message is successfully received [1].

The basic ARQ scheme described above is highly reliable but suffers from very low throughput efficiency when the channel quality degrades. To improve the throughput efficiency of ARQ schemes without sacrificing their reliability, hybrid ARQ/FEC schemes were proposed. Such schemes can be broadly classified as Type-I ARQ or Type-II ARQ. Type-I ARQ is a fixed-rate coding schemes that jointly employs one code for error correction and another code for error detection.

Type-I ARQ is suitable for channels that are uniformly noisy and may be subject to infrequent impulses of noise. For time-varying channels, like those encountered in wireless transmissions, the throughput of the system can be maximized by adaptively matching the error correction capability of the code to the prevailing channel condition. This forms the motivation for Type-II ARQ schemes. The problem of constructing adaptive-rate error correction codes suitable for Type-II ARQ schemes has attracted many researchers for decades.

For instance, an adaptive-rate scheme based on cascaded Hamming codes has been designed and analyzed in [2]. Hagenauer [3] proposed and analyzed a class of codes called Rate-Compatible Punctured Convolutional codes (RCPC), which are families of codes obtained through puncturing of a mother convolutional code. The astonishing performance of turbo codes [4] stimulated many researchers to consider their application in ARQ schemes. Recently, a capacity-approaching Hybrid ARQ techniques based on turbo codes were presented [5]. The authors have used a parity spreading interleaver to realize a flexible puncturing scheme. Then, the reordered parity bits are transmitted sequentially in increments of predetermined size that can ensure rate-compatible codes. In this paper, puncturing will be considered to adaptively match the error correction power of turbo code to the time-varying wireless channel. The puncturing patterns are properly designed to yield the best throughput performance.

The rest of the paper is organized as follows. In the next section the structure of the turbo encoder and decoder are explained. Rate compatible punctured turbo codes will be introduced in Section 3. Based on that, an ARQ scheme is proposed in Section 4. Throughput analysis is furnished in Section 5. This is followed by the simulation results, Section 6, and conclusions, Section 7.

2. TURBO ENCODER/DECODER

In a simplified turbo code, there are two convolutional encoders in parallel. The information bits are scrambled before entering the second encoder. The codeword in a turbo code consists of the input bits — *i.e.* the code is systematic — followed by the parity check bits from the first encoder and then the parity bits from the second encoder, as depicted in Figure 1. In general, one can have multiple turbo encoders with more than two branches. The convolutional code at every branch is called the **c**onstituent **c**ode (CC). The CCs can have similar or different generator functions. However, only the usual configuration with two branches having the same CC will be considered.

A padding block is shown in the figure to append the proper sequence of bits to terminate all the encoders to the all-zero state. This is necessary because a convolutional code may be used to generate a block code if we use beginning and tail bits. If one encoder is considered then the required tail is a sequence of zeros with length equal to the memory order v. The problem of terminating both encoders simultaneously seems to be difficult because of the interleaver. However, it is still possible to do with v tail bits only [6].

In general another interleaver can be used before the first encoder but usually it is replaced with a delay line to account for the interleaver delay and keep the branches properly synchronized.

In mobile radio communications where the channel is time varying, the SNR will instantaneously change such that some packets will be transmitted over a good channel, while others over a bad one. Therefore there is no single scheme that optimizes the throughput but an adaptive scheme. This adaptation can be obtained by puncturing. The advantages of puncturing a mother code are many and the interested reader can refer to the work in [7]. The puncturing module in Figure 1 is used for the rate adaptation of the turbo code.

Turbo codes obtained from Recursive Systematic Convolutional (RSC) codes (where the state of the internal shift register depends on the past outputs) have proven to perform better than the non-recursive ones [7, 8], as such, RSC is widely considered and is adapted here.

The decoder works in an iterative way. Figure 2 shows a block diagram of a turbo decoder. The iteration stage is shown with dotted lines to differentiate it from the initialization stage. In practice the number of iterations does not exceed 18, and in many cases 6 iterations can provide satisfactory performance [7]. The first decoder will decode the sequence and then pass the hard decision together with a reliability estimate of this decision to the next decoder. Now, the second decoder will have extra information for the decoding: an a priori value together with the sequence. The interleaver in-between is responsible for making the two decisions uncorrelated and the channel between the two decoders will seem to be memory-less due to interleaving. The details of what information to pass to the next decoder or next iteration stage are subject to research. In our simulation, a widely accepted decoding algorithm is used, which is the modified soft output Viterbi algorithm.

Viterbi algorithm is an optimal decoding method that minimizes the probability of sequence error for convolutional codes. A modified version of Viterbi algorithm, called SOVA (**S**oft **O**utput **V**iterbi **A**lgorithm), which uses soft outputs is introduced in [9, 10]. SOVA has only twice the complexity of Viterbi algorithm.

Decoding details of turbo codes are out of the scope of this paper. An interested reader is referred to [9–11] for further details.

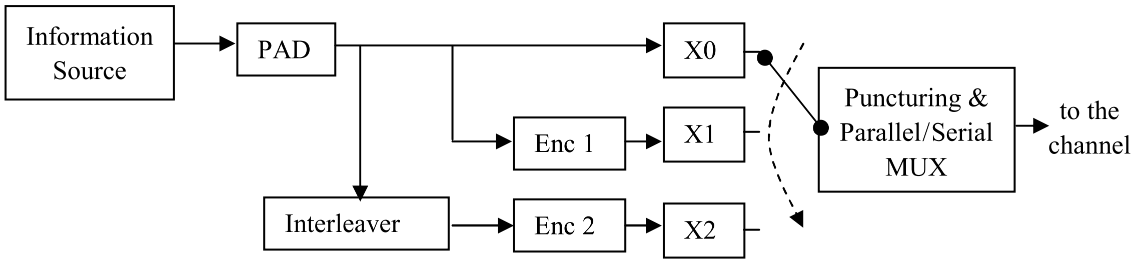

Figure 1. Simplified turbo encoder.

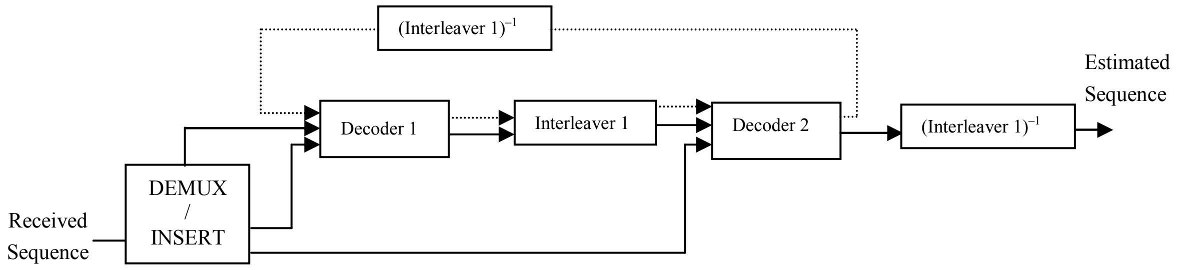

Figure 2. Block diagram of a turbo decoder.

3. RATE COMPATIBLE PUNCTURED TURBO CODES

The puncturing pattern can be described by a puncturing matrix. For a turbo code having N output branches, the puncturing matrix can be represented as follows:

$$\mathbf{P} = \begin{array}{cc} & \leftarrow \; p \; \rightarrow \\ \begin{array}{c}\uparrow\\ N\\ \downarrow\end{array} & \begin{pmatrix} p_{11} & \cdots & p_{1p} \\ \cdots & p_{ik} & \cdots \\ p_{N1} & \cdots & p_{Np} \end{pmatrix} \end{array}, \tag{1}$$

where every row corresponds to a branch of the encoder. Note that $p_{ik} \in \{0,1\}$ where 0 implies that the corresponding bit is punctured. The period of the puncturing matrix is p. A degree of freedom in controlling the code rate can be gained by increasing p.

If $w(\cdot)$ is the Hamming weight operator, then the rate of the code after puncturing with the puncturing matrix $\mathbf{P}$ is:

$$R = \frac{p}{w(\mathbf{P})} \;. \tag{2}$$

Different puncturing patterns of period 4 and rate 1/2 are illustrated in Table 1. For example, the puncturing matrix P_{41} indicates that the first and third bits are not transmitted from the first parity branch. It also indicates that the second and fourth bits are not transmitted from the second parity branch.

Table 1: Different Puncturing Patterns of Period 4.

$P_{41} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$,	$P_{42} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$,	and $P_{43} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$

It is important to mention that rate compatibility is required for Hybrid ARQ schemes. This restricts puncturing rule to such a form that all of the code symbols of high-rate punctured code are embedded in the lower rate codes. At each step the transmitter needs only to transmit supplemental code symbols, reason why Type-II ARQ scheme is often called incremental redundancy.

A family of rate compatible punctured turbo codes (RCPT) can be obtained by puncturing a rate $1/N$-mother turbo code. A set of puncturing rules is defined in the puncturing pattern. Examples of such patterns are shown in Tables 2–4 for different periods. Table 2 shows three sets of puncturing patterns for period $p = 2$: R_{20}, R_{2CC}, and R_{21}. The codes sets R_{20} and R_{2CC} produce the code rates, 1, 1/2, 1/3 while the set, R_{21}, produces the code rates 1, 2/3, 1/2, 2/5, 1/3. Note that in R_{2CC} all bits corresponding to a parity branch have been punctured. This pattern therefore, transforms the turbo code into a convolutional code.

Considering puncturing patterns of period 4, one can produce the rates, $\{1, \frac{4}{5}, \frac{2}{3}, \frac{4}{7}, \frac{1}{2}, \frac{4}{9}, \frac{2}{5}, \frac{4}{11}, \frac{1}{3}\}$. An optimistic assumption of a less pathological channel may try to use all codes in this set. Hence, high throughput may be obtained. However, we have made a rather pessimistic assumption of bad channels, and therefore considered an exponential jump in the code rate selection. This gives rise to few code rates in the set we have used in the retransmission. For example, for $p = 4$ puncturing patterns R_{41}, R_{42}, and R_{43}, shown in Table 3, were selected. Along similar lines, four sets have been considered for period 8, and these are shown in Table 4 (Note that the entries in the table are the decimal representation of the bits forming the puncturing pattern, partitioned starting from right, *e.g.* 2 = 10, 6 = 110, *etc.*).

Table 2: Rate Set Puncturing Pattern for Period 2.

Name	Pattern
R_{20}	$\begin{bmatrix}1&1\\0&0\\0&0\end{bmatrix},\begin{bmatrix}1&1\\1&0\\0&1\end{bmatrix},\begin{bmatrix}1&1\\1&1\\1&1\end{bmatrix}$
R_{2CC}	$\begin{bmatrix}1&1\\0&0\\0&0\end{bmatrix},\begin{bmatrix}1&1\\1&1\\0&0\end{bmatrix},\begin{bmatrix}1&1\\1&1\\1&1\end{bmatrix}$
R_{21}	$\begin{bmatrix}1&1\\0&0\\0&0\end{bmatrix},\begin{bmatrix}1&1\\1&0\\0&0\end{bmatrix},\begin{bmatrix}1&1\\1&0\\0&1\end{bmatrix},\begin{bmatrix}1&1\\1&0\\1&1\end{bmatrix},\begin{bmatrix}1&1\\1&1\\1&1\end{bmatrix}$

Table 3: Rate Set Puncturing Pattern for Period 4.

Name	Pattern
R_{41}	$\begin{bmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&0&0\\0&0&0&1\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&1&0\\0&1&0&1\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&1&1\\1&1&0&1\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$
R_{42}	$\begin{bmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&0&0\\1&0&0&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&1&0&0\\1&1&0&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&1&1&0\\1&1&1&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$
R_{43}	$\begin{bmatrix}1&1&1&1\\0&0&0&0\\0&0&0&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&0&0\\0&0&1&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&0&1\\0&1&1&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&0&1&1\\1&1&1&0\end{bmatrix},\begin{bmatrix}1&1&1&1\\1&1&1&1\\1&1&1&1\end{bmatrix}$

Table 4: Rate Set Puncturing Pattern for Period 8.

Name	Pattern
R_{81}	$\begin{bmatrix}3&7&7\\0&0&0\\0&0&0\end{bmatrix},\begin{bmatrix}3&7&7\\2&0&2\\1&0&1\end{bmatrix},\begin{bmatrix}3&7&7\\3&1&2\\3&2&1\end{bmatrix},\begin{bmatrix}3&7&7\\3&5&3\\3&6&5\end{bmatrix},\begin{bmatrix}3&7&7\\3&7&7\\3&7&7\end{bmatrix}$
R_{82}	$\begin{bmatrix}3&7&7\\0&0&0\\0&0&0\end{bmatrix},\begin{bmatrix}3&7&7\\2&1&0\\0&2&1\end{bmatrix},\begin{bmatrix}3&7&7\\2&5&2\\1&2&5\end{bmatrix},\begin{bmatrix}3&7&7\\3&7&2\\1&3&7\end{bmatrix},\begin{bmatrix}3&7&7\\3&7&7\\3&7&7\end{bmatrix}$
R_{83}	$\begin{bmatrix}3&7&7\\0&0&0\\0&0&0\end{bmatrix},\begin{bmatrix}3&7&7\\2&1&0\\0&4&1\end{bmatrix},\begin{bmatrix}3&7&7\\2&5&1\\2&5&1\end{bmatrix},\begin{bmatrix}3&7&7\\3&5&5\\2&7&3\end{bmatrix},\begin{bmatrix}3&7&7\\3&7&7\\3&7&7\end{bmatrix}$
R_{84}	$\begin{bmatrix}3&7&7\\0&0&0\\0&0&0\end{bmatrix},\begin{bmatrix}3&7&7\\1&0&4\\0&2&1\end{bmatrix},\begin{bmatrix}3&7&7\\1&4&6\\2&3&1\end{bmatrix},\begin{bmatrix}3&7&7\\3&5&6\\2&7&3\end{bmatrix},\begin{bmatrix}3&7&7\\3&7&7\\3&7&7\end{bmatrix}$

4. RCPT-BASED ARQ PROTOCOL

The RCPT encoder encodes the information packet and stores all the coded bits in buffers for any possible retransmissions. The rate selector determines which parity bits are to be transmitted to ensure the codes adaptation to the state of the channel. The Rayleigh flat fading channel is assumed. The fade samples are zero mean and unit variance complex Gaussian variables. It is assumed that error-free (reliable), low capacity feedback channel does exist over which the receiving system can transmit the acknowledgements.

ARQ systems can be implemented by choosing RCPT codes in a strategic way for each retransmission. The RCPT codes can be implemented with stop-and-wait, go-back-*N*, or selective-repeat schemes. However, selective-repeat ARQ strategy is considered for its throughput efficiency and the ease at which it can be adapted to other schemes [12]. For simplicity of analysis, we have considered an infinite buffer size. The general steps involved in the RCPT–ARQ protocol are given as follows.

(1) Encode the k^{th} packet with the rate 1/3 turbo code and buffer all the coded bits. Choose a set of puncturing patterns. Let *m* denote the set size.

(2) First transmit the bits specified in the first pattern of the set (usually the information bits only).

(3) The receiver assembles the received bits, inserting erasures for the bits that have not been transmitted according to the puncturing, and decoding is attempted. We have assumed that all error patterns can be detected. If the frame received by the sink is error free, an ACK is sent to the transmitter and the protocol is reset and returned to step 1. Otherwise, a NACK is transmitted back.

(4) Upon reception of NACK the transmitter sends additional bits according to the next pattern of the set, then back to step 3.

The first *m* transmissions will all be distinct. They exhaust all the code rates in the set, reaching the full turbo code of rate 1/3. We say this scheme has a degree of freedom of adaptation of *m*. If an error-free packet cannot be passed yet, different design options can be considered. The simplest is to accept a given frame error rate and pass the frame as is. Another option is to hold the decoder in full turbo but the transmitter is reset to step 2. In this case popular code combining is done.

5. THROUGHPUT ANALYSIS

Turbo codes are linear and hence they can be analyzed using the union bound. The fact that a turbo code is linear can be seen by recalling that every constituent code (*i.e.* the convolutional code that forms the turbo code) is linear and the interleaver performs a linear operation. Therefore, the union bound can be used for comparing the performance of different turbo codes. It can also be used to evaluate the effect of changing certain code ingredients.

Turbo code performance at high SNR is best evaluated using analytical bound due largely to prohibitive amount of simulation required. The bound is good at high SNR where neglecting higher weights is less detrimental to the approximate bound. The evaluation of the union bound requires finding the weight distribution of turbo codes. This can be obtained through the transfer function of the code. Different techniques are developed to account for the effects of the interleaver and the puncturing pattern [13]. Since the focus of this paper is on throughput analysis, it is assumed that the frame error rate, FER, is known. For calculation of BER and FER, the reader is referred to [13].

The throughput, ρ, is a measure of the transmission efficiency. For selective-repeat ARQ scheme this is defined as the ratio of the length of the information frame to the average total number of bits, N_a, required to be transmitted before the information frame is passed to the sink.

For the type-II HARQ system, the average number of frames needed to transmit one frame of information is bounded as follows. Let F_k denote the probability of frame error after processing the k^{th} transmission of bits belonging to the frame. Let us consider the case of three stages of adaptation, where firstly the systematic bits are transmitted (frame error F_1), followed by first set of parity (frame error F_2) and then second set of parity (frame error of TC F_3). Note that in this case the parity frames are of the same length as that of the information frame. If error-free decoding is not achieved, this sequence is repeated but the decoder will be held in full turbo decoding. That is on the k^{th} transmission, $k > 3$, the decoder utilizes the newly transmitted sequence with the most recent transmissions of the other *two* sequences. For RCPT–ARQ system with three stages of adaptation ($m = 3$) N_a can be obtained as,

$$N_a = 1(1-F_1) + 2F_1(1-F_2) + 3F_1F_2(1-F_3) + 4F_1F_2F_3(1-F_3) + 5F_1F_2F_3^2(1-F_3) + \ldots$$

$$= 1(1-F_1) + 2F_1(1-F_2) + F_1F_2(1-F_3)\sum_{n=0}^{\infty}(n+3)F_3^n\,, \tag{3}$$

Where, $F_1 = (1-(1-\varepsilon)^K)$, ε is the channel bit error probability, F_2 depends on the puncturing pattern used and F_3 is the frame error rate for the turbo code. After some manipulations of (3), we arrive at,

$$N_a = 1 + F_1 + F_1F_2/(1-F_3). \tag{4}$$

Finally the throughput efficiency, ρ, can be found,

$$\rho = 1/N_a\,. \tag{5}$$

Along the same lines, we can evaluate the throughput of the ARQ scheme when more adaptation steps are utilized. It is worth noting that by increasing the number of adaptation steps the size of the parity frames becomes a fraction of that of the information frame. For example, at $m = 5$ the size of the parity frame is half that of the information frame.

We have assumed in the previous analysis that frames are retransmitted until received without errors. This could end up with a very long (theoretically infinite) delay. Such cases cannot be tolerated in real-time applications. Another strategy suitable for real-time applications is to stop the retransmission at certain maximum number of requests thus limiting the delay to an acceptable level and tolerate some frame error. With this strategy, N_a for $m = 3$ becomes,

$$N_a = 1(1-F_1) + 2F_1(1-F_2) + 3F_1F_2(1-F_3) + 4F_1F_2F_3(1-F_4) + \ldots + nF_1\ldots F_{n-1}\,(1-F_n)$$

$$= \sum_{k=1}^{n} k(1-F_k)\prod_{j=1}^{k-1}F_j\,, \tag{6}$$

where n is the number of allowable decoding attempts. The system will then be evaluated in terms of frame error rate in addition to the throughput efficiency. The frame error rate is given by:

$$FER = F_n\,. \tag{7}$$

The performance analysis presented above indicates that we need to evaluate the FER for the turbo code for the different parity transmitted based on the puncturing patterns used in order to determine the throughput for the RCPT–ARQ scheme.

For the completeness of the analysis we mention some factors that will slightly reduce this throughput. These are the CRC bits and tail bits needed to force the constituent encoder to a zero state. Being few compared to the frame size, their effect on throughput is minimal. We have therefore ignored them for the sake of simplicity.

The frame error rate bounds on the probabilities for TC, or by extension PTC are known to be weak. The extreme weakness of this bound has been revealed in [12]. This is the main reason why the throughput performance of RCPT–ARQ is almost always estimated through simulation, and this work is not an exception. However, the bound of Equation (6) is still useful to show for certain configurations of the puncturing schemes as it reveals the relative performance of these schemes. This is shown in Figure 3 for the region in which the bounds make some sense. Figure 3 shows the bound for a turbo-based FEC/ARQ where uncoded, convolutional or rate-1/3 turbo code is used with no rate adaptation. Also shown is a scheme where rate adaptation is considered. We have used three steps ($m = 3$); the uncoded (rate = 1), best puncturing pattern of period 2 (rate = 1/2), and turbo code (rate = 1/3). It can be seen how an adaptive scheme together with good puncturing patterns can improve the throughput.

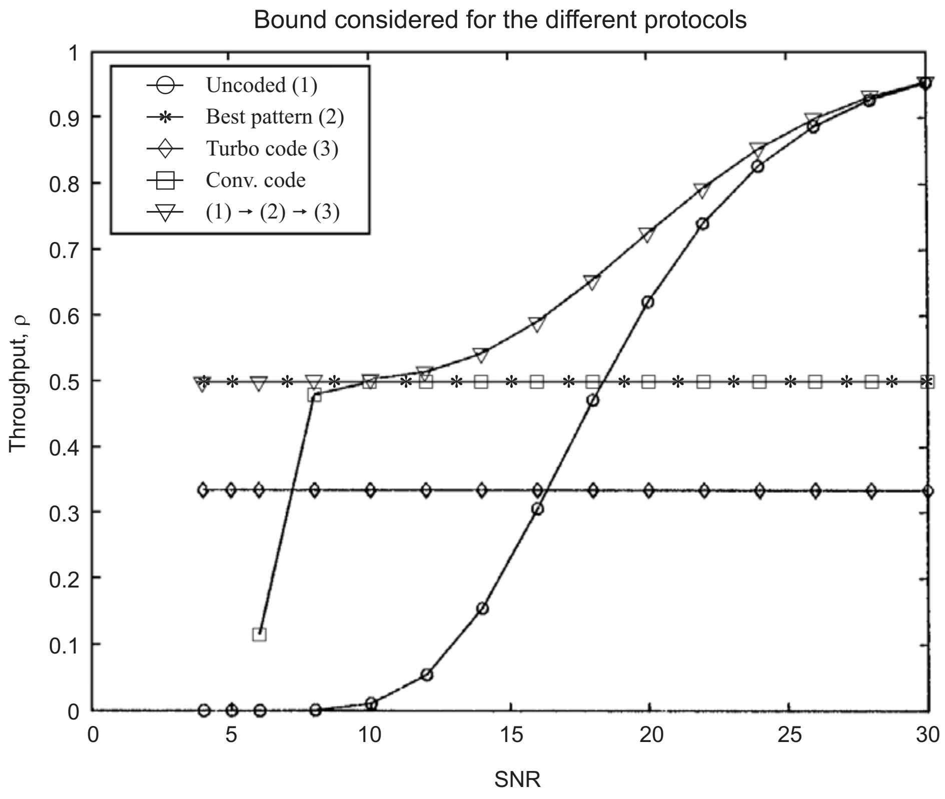

Figure 3. The through bound for different retransmission strategy.

6. THE SIMULATED THROUGHPUT RESULTS

The throughput simulation results are presented for turbo-hybrid ARQ scheme for different parameter settings. Firstly, the effect of the period of the puncturing pattern on the throughput performance is considered. Secondly, the effects of different puncturing patterns are evaluated. And thirdly, the performance changes due to increasing the number of adaptation steps.

We start by examining the effect of the selection of the puncturing patterns. For that purpose the period of puncturing p and the size of the code set (adaptation stages) are kept fixed. Two code set sizes: $m = 3$, 5, and puncturing periods, $p = 2, 4, 8$, are considered.

Figure 4 presents the throughput performance curves for schemes using puncturing patterns of period 4 from Table 1. For all curves, the puncturing patterns for the first and last stages are the same: only information bits, and the full code, respectively. For the intermediate stage, the middle (third) puncturing pattern of each of P_{41}, P_{42}, and P_{43} shown in Table 1 is used. It is clearly seen that the performance obtained with P_{43} is woeful in comparison with others. The results show that the gain in throughput due to selecting a good puncturing pattern could reach more than 15%.

The throughput performance evaluation is extended to $m = 5$ (five steps of adaptation). It is worth noting that for higher periods additional flexibility is obtained in the number of intermediate steps before reaching the full turbo code. This implies that more degree of freedom is allowed in the puncturing rule. In fact, this can yield a better code performance compared to the case of shorter puncturing periods. Figure 5 shows the effect of the position of the punctured bits for period 4. It is seen that code rate set of R_{41} gives a better performance than R_{42} or R_{43}. We have generally observed that for the code structure used, the bits retained or punctured of the parity sequences have direct effect on the code performance contrary to the opinion contained in [14]. Their conclusion may be based on the context of their investigation.

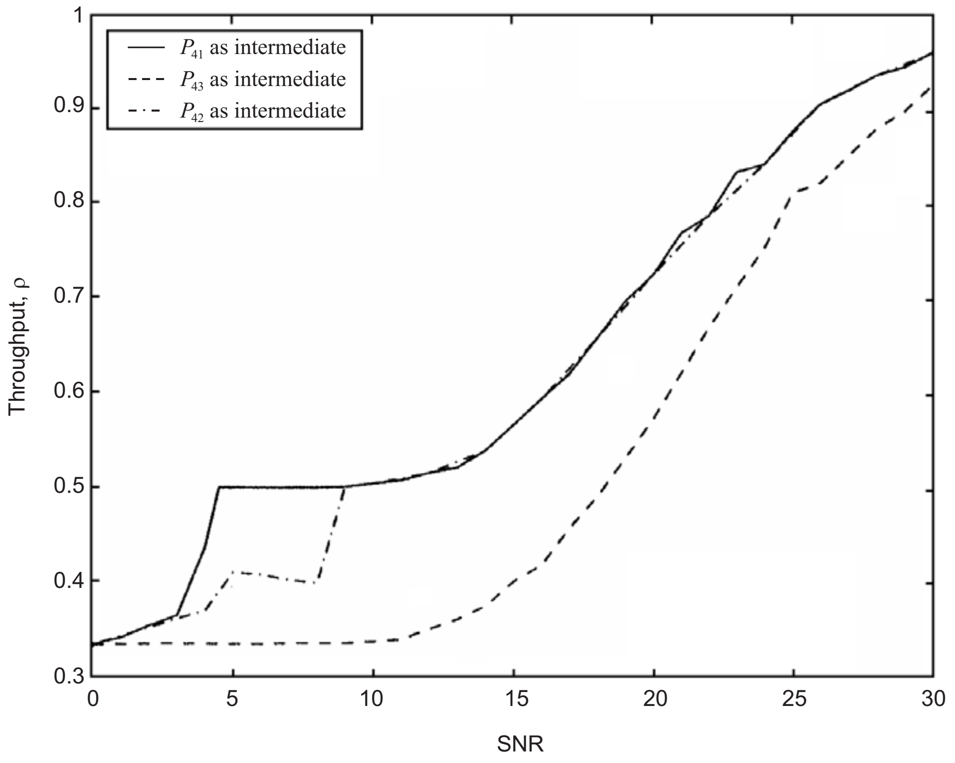

Figure 4. Throughput comparison of different patterns for $p = 4$, $m = 3$.

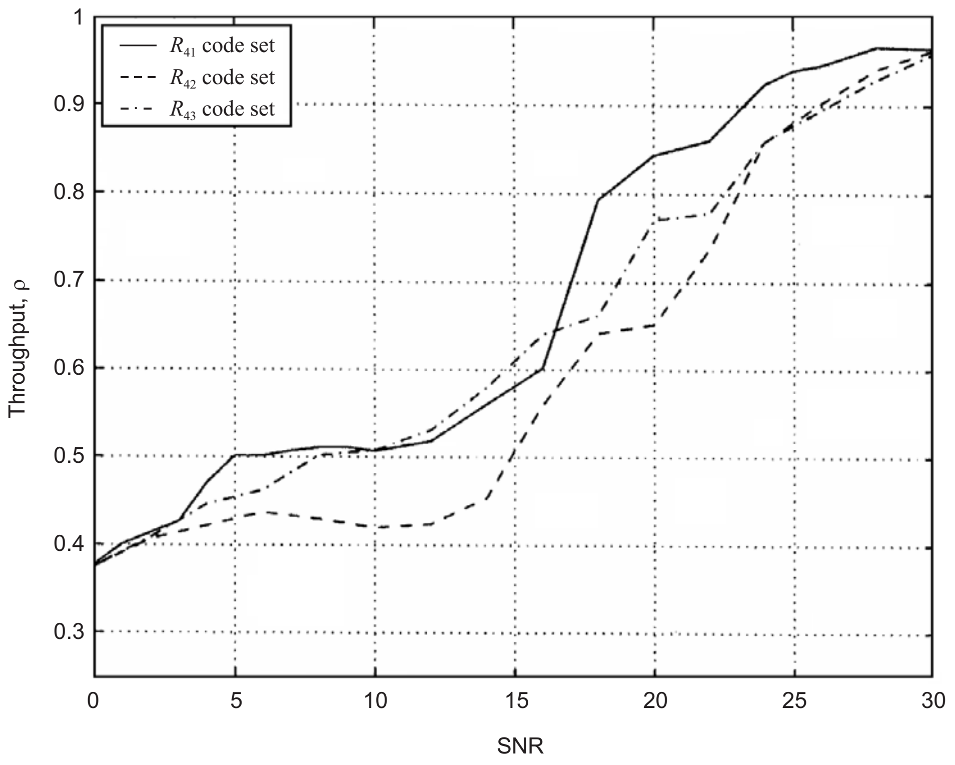

Figure 5. Throughput comparison of different patterns for $p = 4$, $m = 5$.

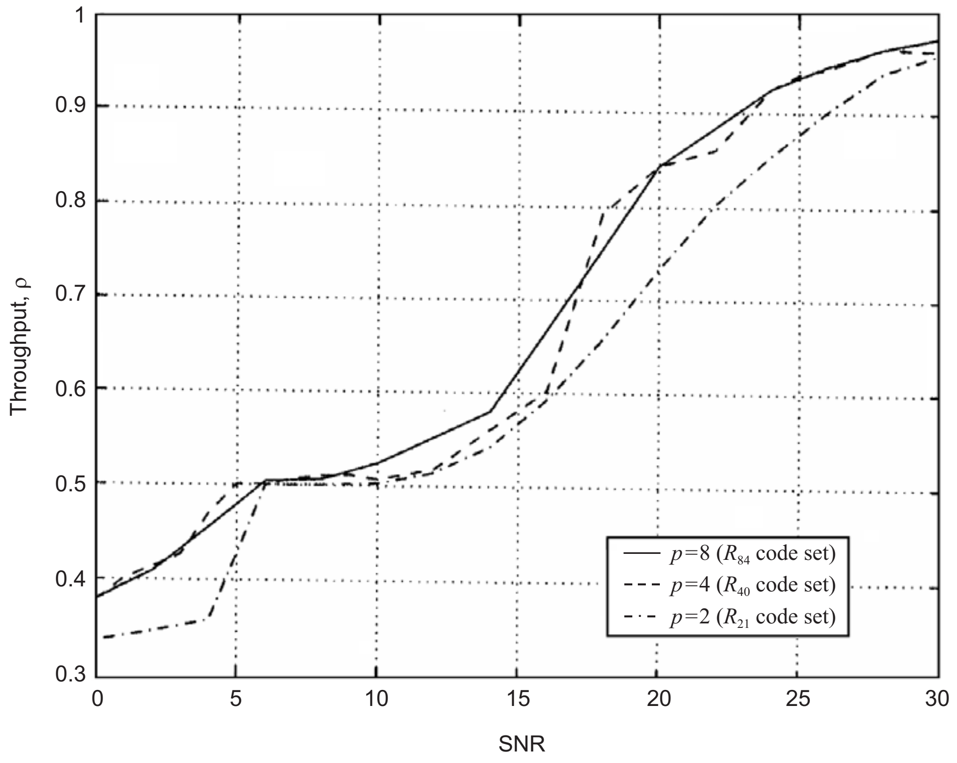

Figure 6. Throughput comparison of different periods with $m = 5$.

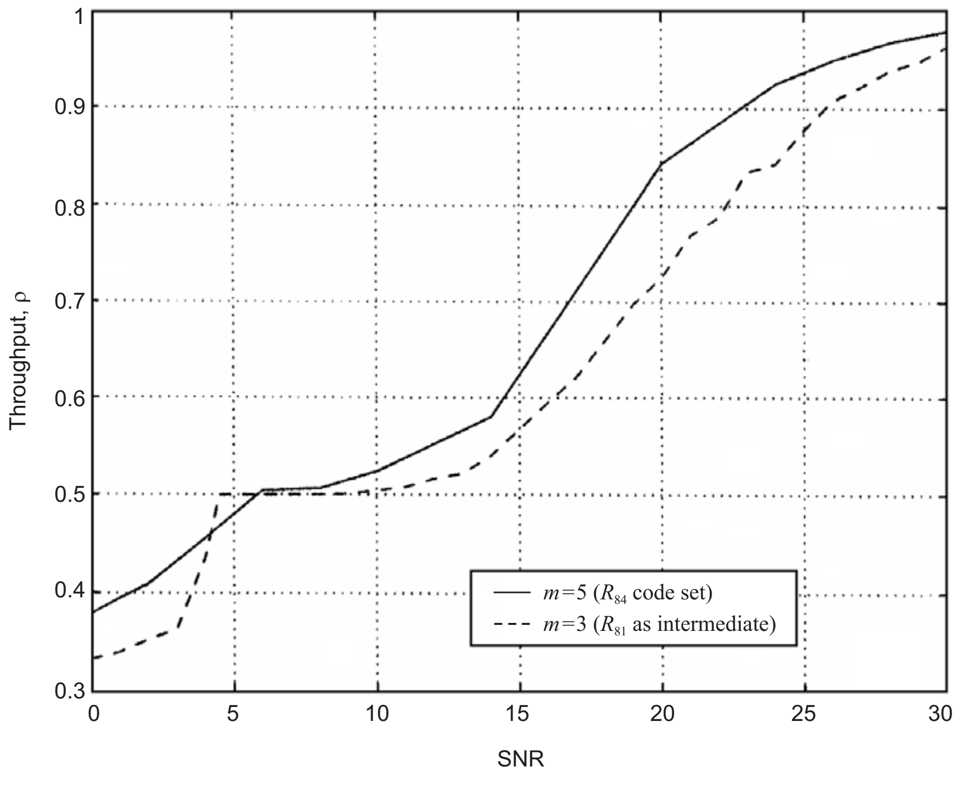

Figure 7. Throughput comparison of different intermediate rates of periods 8.

Next, the effect of the puncturing period on the code performance is investigated. For that reason m is kept fixed, and the best pattern for each case is used. The value $m = 5$ is considered in Figure 6 for the best performance of periods 2, 4, and 8. The improvement in the throughput is observed using a larger period. This can be attributed to the freedom offered by larger periods in distributing the punctured bits. However, the improvement is of diminishing nature. The improvement of going from $p = 4$ to $p = 8$ is less than that from $p = 2$ to $p = 4$.

Finally, the effect of the adaptation steps on the throughput performance is studied. For this case we have compared the code rate set {1, 2/3, 1/2, 2/5, 1/3} with the set {1, 1/2, 1/3} for $p = 8$ in Figure 7. Generally, the higher the degree of freedom in selecting the code rate the higher the throughput performance. It was observed that the gain is more pronounced for larger periods than for shorter ones, an improvement of 10% is obtained due to increasing the number of adaptation steps for $p = 4$, 8.

7. CONCLUSIONS

This paper examined hybrid ARQ schemes built on punctured turbo codes, and evaluated their performance by simulation. It is found that the puncturing pattern have a significant effect on the throughput. Simulation results show that a throughput difference of 15%–20% between the best pattern set and the worst pattern set is possible. When the period of puncturing is prolonged, even higher throughput values are possible, provided that the puncturing patterns are properly selected. However, the effect of prolonging the period diminishes as the period increases. It was also shown that increasing the degree of freedom of adaptation (more transmissions before repetition starts) can improve the system throughput, provided that the puncturing period is long enough to allow the design of good puncturing pattern sets.

ACKNOWLEDGMENT

This work was done under Project FT2000-01. The authors would like to acknowledge KFUPM for funding and supporting this research.

REFERENCES

[1] S. Lin, D. Costello, and M. Miller, "Automatic Repeat Request Error Control Schemes", *IEEE Transactions on Communication*, **22** (1984), pp. 5–17.

[2] M. Kousa and M. Rahman, "An Adaptive Error Control System Using Hybrid ARQ Schemes", *IEEE Transactions on Communication*, **COM'39(7)** (1991), pp. 1049–1057.

[3] J. Hagenauer, "Rate Compatible Punctured Convolutional Codes (RCPC-codes) and their Application", *IEEE Transactions on Communication*, **36** (1988), pp. 389–400.

[4] Claude Berrou and Alain Glavieux, "Near Optimum Error Correcting Coding and Decoding: Turbo-Codes", *IEEE Transactions on Information Theory*, **44** (1996), pp. 1261–1271.

[5] R. Mantha and F.R. Kschischang, "A Capacity Approaching Hybrid ARQ Scheme Using Turbo Codes", *Global Telecommunication Conference 1999 (GLOBECOM '99)*, pp. 2341–2345.

[6] W.J. Blackert, E.K. Hall, and S.G. Wilson, "Turbo Code Termination and Interleaver Conditions", *Electronic Letters*, **31(24)** (1995), pp. 2082–2084.

[7] C. Berrou, A. Glavieux, and P. Thitimajhima, "Near Shannon Limit Error Correcting Coding and Decoding: Turbo-Codes", *Proceedings of ICC '93, Geneva*, May 1993, pp. 1064–1070.

[8] Sergio Benedetto and Guido Montorsi, "Design of Parallel Concatenated Convolutional Codes", *IEEE Transactions on Communications*, **44(5)** (1996), pp. 591–600.

[9] J. Hagenauer and P. Hoeher, "A Viterbi Algorithm with Soft-Decision Outputs and Its Applications", *Proceedings of GLOBECOM '89, Dallas, Texas*, November 1989, pp. 47.11–47.17.

[10] J. Hagenauer and L. Papke, "Decoding Turbo Codes with the Soft-Output Viterbi Algorithm (SOVA)", in *Proceedings of International Symposium on Information Theory, Trondheim, Norway*, June 1994, p. 164.

[11] L. Papke and P. Robertson, "Improved Decoding with the SOVA in a Parallel Concatenated (Turbo Code) Scheme", *IEEE Int. Conference on Communications ICC '96,* 1996, pp. 102–106.

[12] Douglas N. Rowitch and Laurence B. Milstein, "On The Performance of Hybrid FEC/ARQ Systems Using Rate Compatible Punctured Turbo RCPT) Codes", *IEEE Transactions on Communication*, **48** (2000), pp. 948–959.

[13] A. Mugaibel and M. Kousa, "Evaluation of Transfer Functions for Punctured Turbo Codes", *IEE Electronics Letters*, **36**(**9**) (2000), pp. 805–807.

[14] Fan Mo, S.C. Kwatra, and Junghwan Kim, "Analysis of Puncturing Pattern for High Rate Turbo Codes", *IEEE Military Communication Conference '99*, November 1999, vol. I, pp. 547–550.

Paper Received 7 June 2003; Revised 7 January 2004; Accepted 13 January 2004.