3GPP TSG RAN WG1#9 Tdoc R1-(99)J08
Dresden, Germany
November 30th - December 3rd, 1999

Agenda Item: Adhoc4, Rate Matching

Source: Siemens

Title: End puncturing for short convolutional codes

Document for: Discussion

Abstract

End puncturing had been shown to provide an improved performance for short rate 1/3 convolutional codes (in the order of 24 to 120 payload bits) [1]. In this contribution we show that end punctured rate 1/2 convolutional codes also show such a performance improvement, both for AWGN and fading environments. We also show that the scheme can be combined with blind rate detection for fixed positions in the downlink.

We further show that this scheme also achieves a good performance for very small numbers of payload bits (e.g. only 3 bits). Here the performance is even better than the performance of the biorthogonal coding scheme (used for TFCI coding) which was proposed in [2] for this particular application.

An other application which we will highlight is the use of end puncturing to anticipate radio frame equalisation.

We have already shown, that the advantages of end puncturing are not turned into a disadvantage, even if it is followed by and independent puncturing/repetition operation in the rate matching step. For further improvements we have developed a method that integrates both operations in a single procedure, but we will only suggest this to be studied for release 2000 due to time constraints.

We will therefore suggest to incorporate this scheme in [3] and [4] to improve the performance of convolutional codes for small and very small numbers of input bits.

Principle

In this chapter we shortly recap the construction principle for end puncturing. The scheme relies on the fact, that the first and last bits of a block have a better BER than the bits in the middle. This is caused by the fact, that the trellis of the convolutional code begins in a known state and also ends in a known state. These implicit known states are available for decoding so that the information bits at the beginning and at the end of a block are better protected against transmission errors than the bits in the middle of a block. However, this does not achieve the best overall performance, because energy is wasted to obtain the extra protection of the bits at both ends. This waste of energy can be avoided if some more bits are punctured at both ends of the code to align the error probability of the bits at the ends with the one of the bits in the middle. This can be obtained by puncturing a predetermined pattern of bits at the beginning and end of the coded bits. This operation can be implemented right after the convolutional coder before of rate matching.

The principle of end puncturing, compared with regular puncturing is shown in Fig. 1. The number of error events in the middle of the block is reduced whereas the error events at both ends are increased, where it does not ‘hurt’ because the error rate is anyhow below average. The fixed puncturing pattern is only applied to the beginning and end of the block.

For end puncturing of rate 1/3 codes always 8 coded bits are punctured on both ends, i.e. 16 coded bits are punctured in total. The following end puncturing pattern is used (‘x’ means that this bit is punctured):

x x 1 x 1 x 1 x 1 1 x 1 1 x 1 1 x 1 1 1 1 ... 1 1 1 1 x 1 1 x 1 1 x 1 1 x 1 x 1 x 1 x x

After this fixed puncturing pattern the normal rate matching is performed to match the number of transmitted bits with the available bits which are offered by the slot structure. Even though two independent puncturing or repetition operations are not generally optimum, the performance advantage of end puncturing is not offset by this effect.

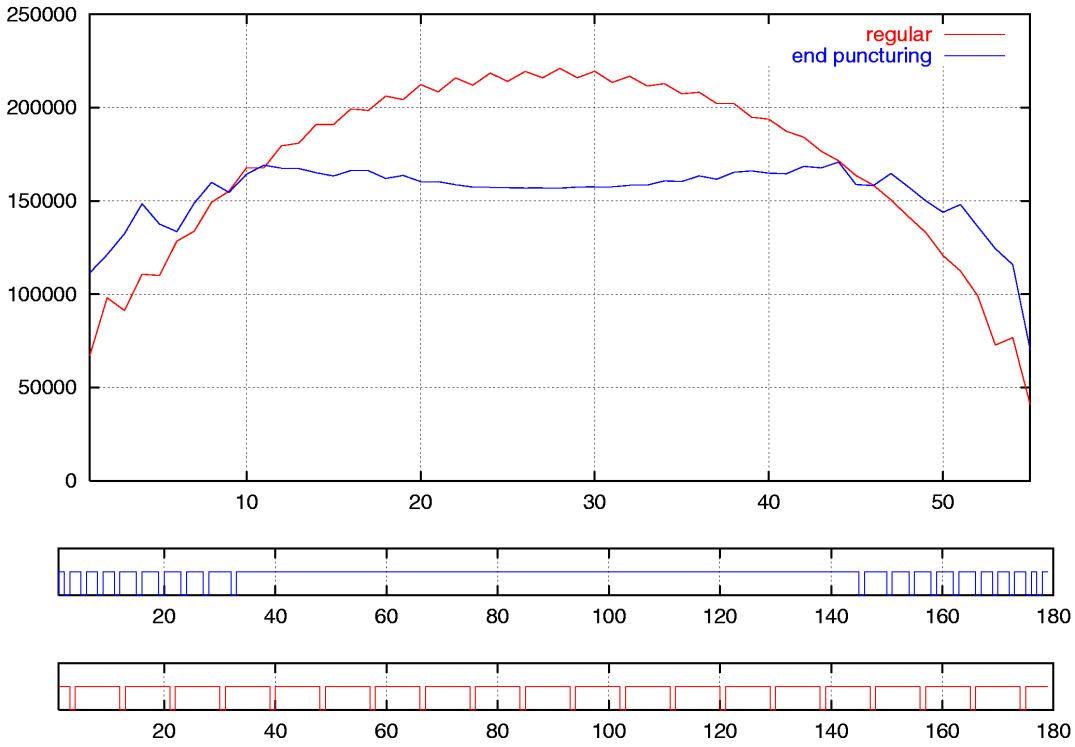

Fig. 1: Principle of end puncturing scheme

The simulation results presented in [1] showed a performance gain of about 0.4 dB for short block lengths (96 coded bits) cf. Fig. 2. At larger block lengths the performance gain is 0.3 dB for a block length of 192 coded bits and 0.2 dB for 384 coded bits. The Simulations were run under AWGN conditions. Clearly, the smaller the data blocks are the better is the performance gain of end puncturing. For very large block sizes the difference will be marginal.

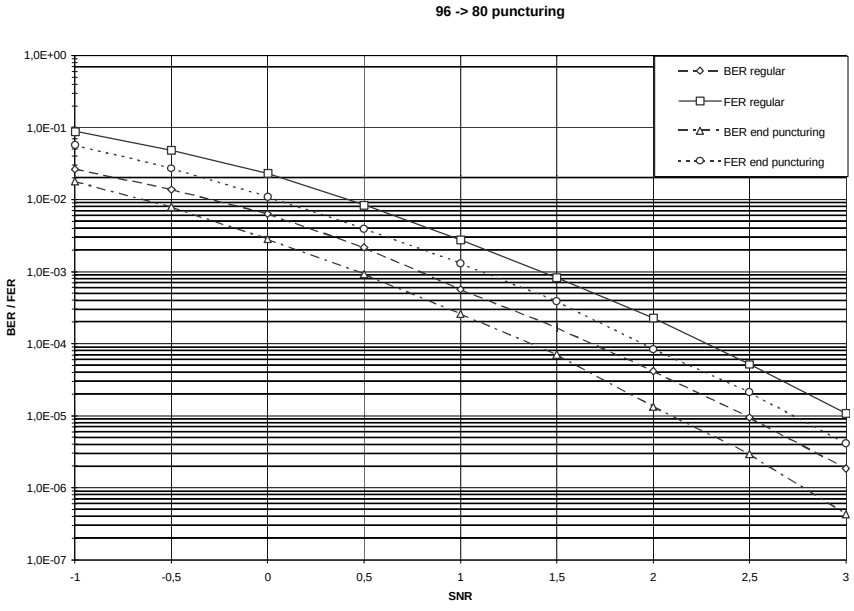

Fig. 2: Performance of end-puncturing, compared to normal puncturing for rate 1/3

End puncturing for rate 1/2 codes

The puncturing pattern presented above was designed for a rate 1/3 code. Clearly, a rate 1/2 code will require a different pattern, mainly because there are less coded bits which correspond to the known tail bits. We have also run simulations for the rate 1/2 convolutional code and found a performance improvement of up to 0.25dB too. The results are shown in Fig. 3. We have assumed 32 payload bits which are coded into 72 transmitted bits and again compared to the normal, equidistant rate matching.

The puncturing pattern used for this case was:

1 1 x 1 x 1 1 1 x x 1 1 1 1 ... 1 1 1 1 x 1 x x 1 1 x 1

Again an x denotes a punctured bit.

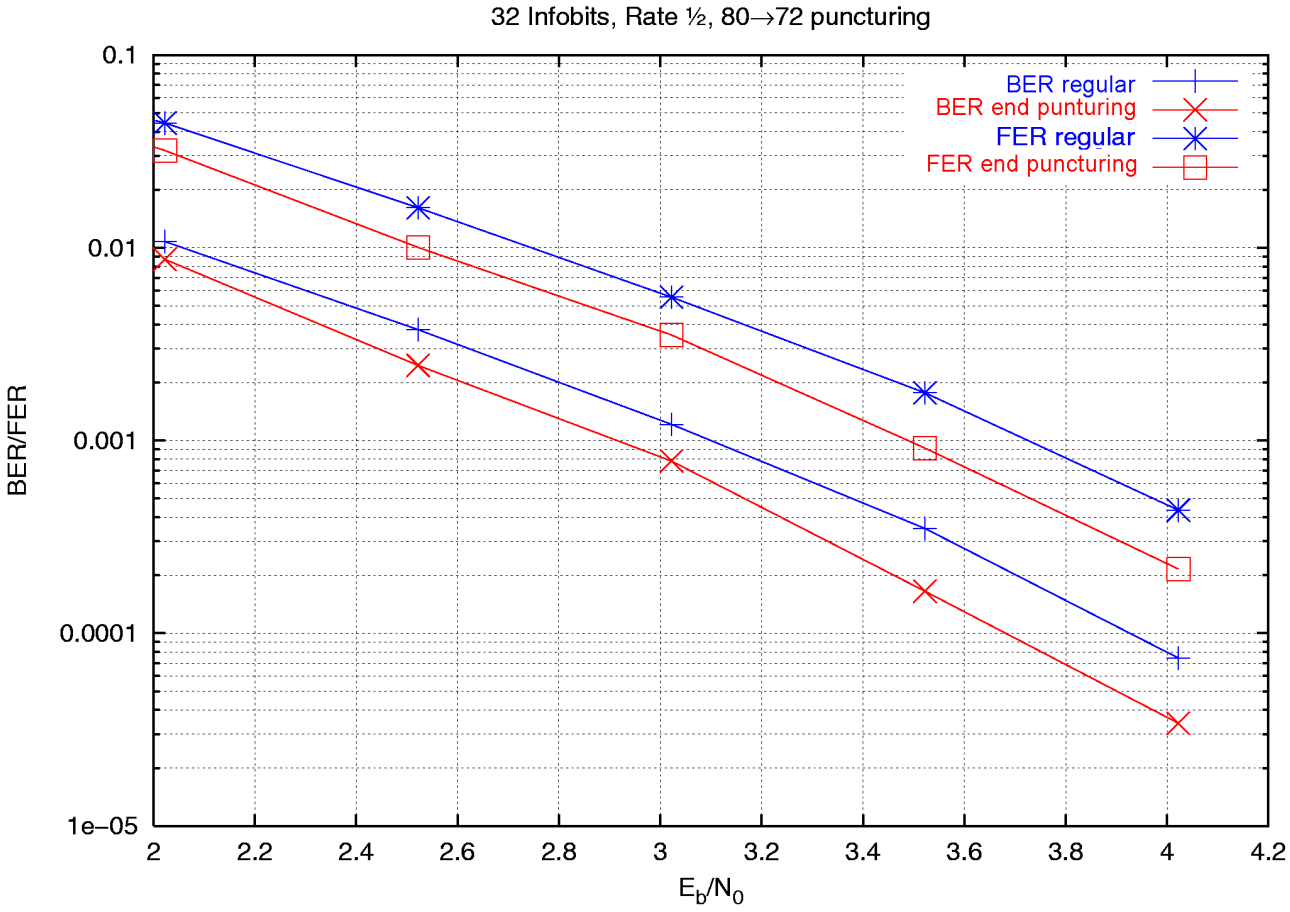

Fig. 3: Performance of end-puncturing, compared to normal puncturing for rate 1/2, AWGN channel

We have also performed simulations for the same scenario under fading conditions (30 km/h, no power control, ideal channel estimation, block interleaving) and found a FER performance improvement of even 0.35 dB as shown in Fig. 4.

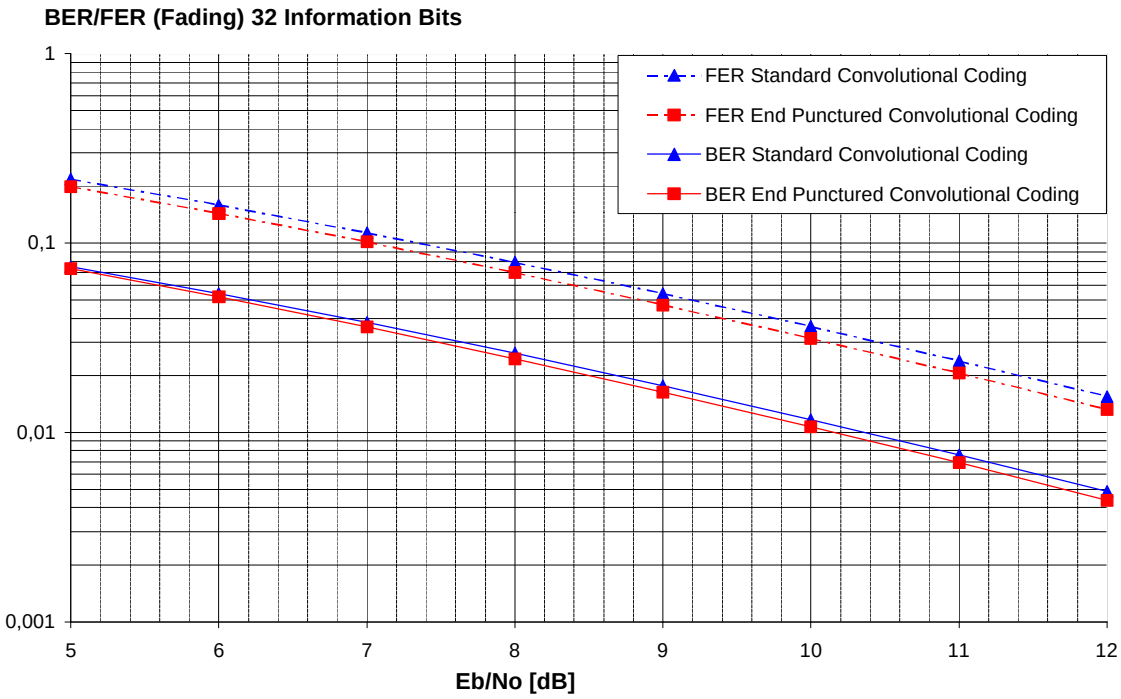

Fig. 4: Performance of end-puncturing, compared to normal puncturing for rate 1/2, fading channel

Blind rate detection in conjunction with fixed bit positions

For blind rate detection in conjunction with fixed bit positions, which is applicable in the downlink, concerns were raised at the last meeting. These concerns were justified.

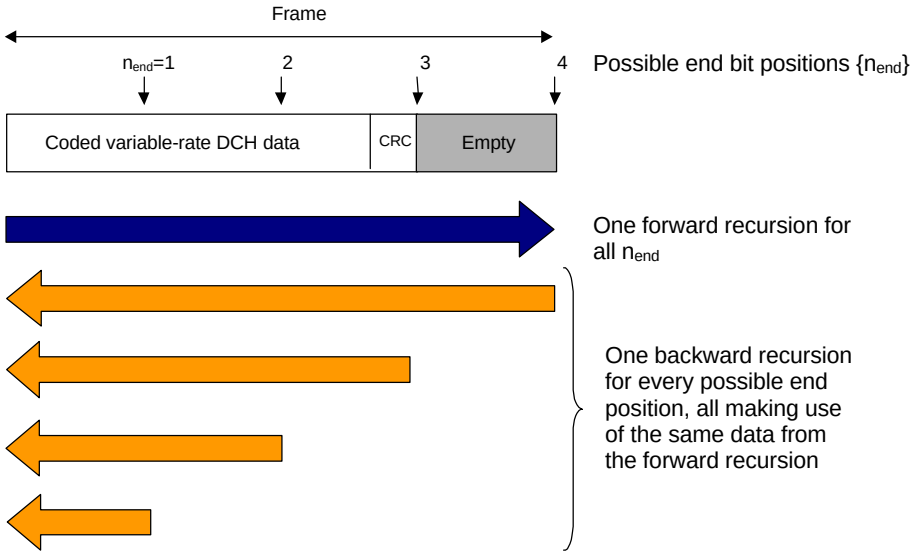

Fig. 5: Blind rate detection using fixed positions without end puncturing

The principle of blind transport format detection using fixed positions with CRC is described in Annex A.1.2 "Blind transport format detection using CRC" of [3]. Here the trellis of a Viterbi decoder is traced back from all potential end positions, as shown in Fig. 5. The correct end position is verified by a CRC check. If an end puncturing pattern is applied after coding, the data from a single forward recursion cannot be used, because the puncturing pattern will be at different positions, depending on which end position is applicable. Therefore multiple runs of the forward recursion must be done. It is not necessary to perform complete runs, only the last part, corresponding to the length of the end

puncturing pattern, has to be redone. This is shown by the short forward arrows in Fig. 6. The white circles indicate the positions of the punctured bits at the end.

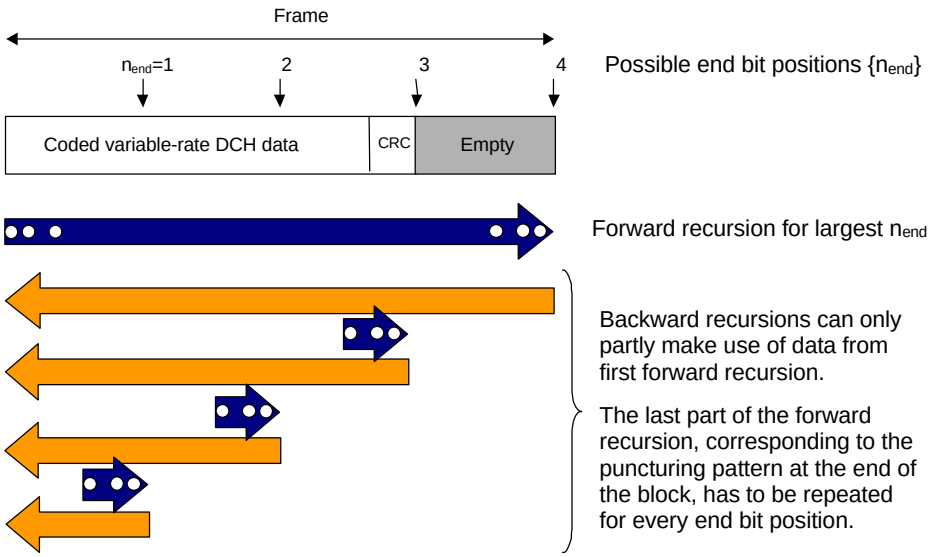

Fig. 6: Blind rate detection using fixed positions with end puncturing. White circles symbolise the end puncturing positions.

Obviously, blind rate detection is not impossible in this case, but it requires more computations for the multiple parts of the forward recursion. Still the complexity is much less than for the blind rate detection for variable positions, because in this case a complete forward recursion has to be performed for each possible block size, not only a partial forward recursion. As can be seen from Fig. 6, it is the puncturing at the end, not at the beginning, which causes the trouble. Therefore we propose not to apply the puncturing at the end but only at the beginning. In this way 50% of the potential improvement can still be obtained without any complexity increase for the forward recursion.

End puncturing for very short blocks using rate 1/2 codes

Rate 1/2 convolutional coding will be a candidate for very small block sizes (e.g. 3 bits). However, even with rate 1/2 coding, the effective code rate would be 3/(2*(3+8)) = 1/7.33 which may be to low in comparison to other transport channels (depending on their relative quality of service requirements). We therefore have optimised the puncturing pattern for very short blocks as well. We have selected a puncturing pattern of 8 bits, this would result in an effective code rate of 3/(2*(3+8) -8) = 1/4.66 which is still smaller than rate 1/2 or 1/3, but which will be in a reasonable range. As there is less coding gain than for example for turbo codes, the effective rate may possibly have to be lower than 1/3. Lower rates than 1/4.66 can be easily realised by repetition. In this case it is better to start with a good code of 14 bits and employ repetition, rather than start with a worse 22 bit code and hope that puncturing will improve it.

We have conducted simulations and compared the results both to the ordinary rate 1/2 convolutional codes. We have biased the simulation assumptions against the proposed scheme: We have repeated bits after end puncturing to arrive with 22 bits in both cases.

We have also simulated the competing second order Reed Muller coding scheme, which proposes to use the TFCI coding scheme, but then transmits this information not via the TFCI bits, but in an ordinary transport channel. In other words, convolutional coding is replaced by second order Reed Mueller coding. We have assumed all the lately proposed performance improvements for TFCI coding (optimised order of masks, using all the 32 bits without puncturing). This will of course overestimate the performance of the TFCI coding scheme as has already been shown, because this scheme is sensitive to puncturing.

Again we find that the end punctured convolutional scheme has an advantage of 0.2 dB against the conventional convolutional code and an advantage of even 0.5 dB against the standard TFCI coding scheme in AWGN scenarios.

In fading conditions the proposed scheme also shows the best BER performance but there is less difference for FER. It should be noted that the simulation conditions were biased against the proposed scheme, in particular in comparison to

TFCI coding, because for the latter 32 bits are sent compared to 22bits. More bits of course give a better time diversity, which is crucial in fading conditions.

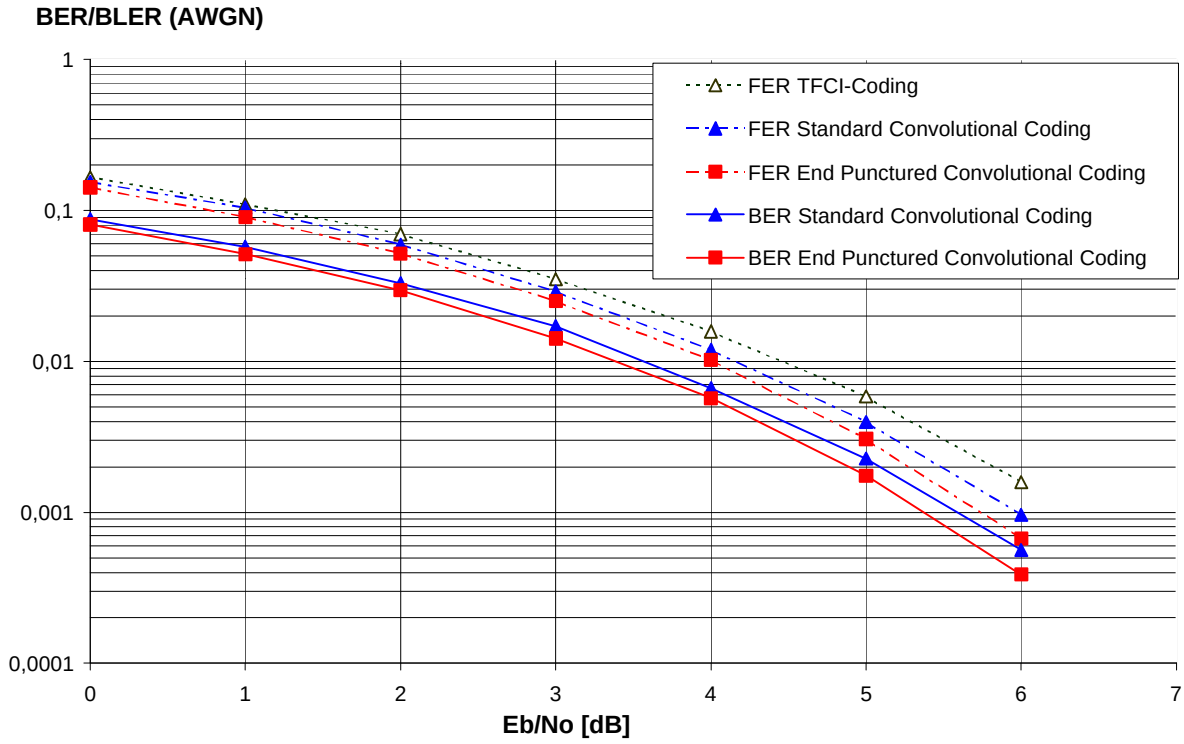

Fig. 7: Performance of end-puncturing for very short block size of 3 bits, compared to standard convolutional coding and to standard TFCI coding, AWGN channel

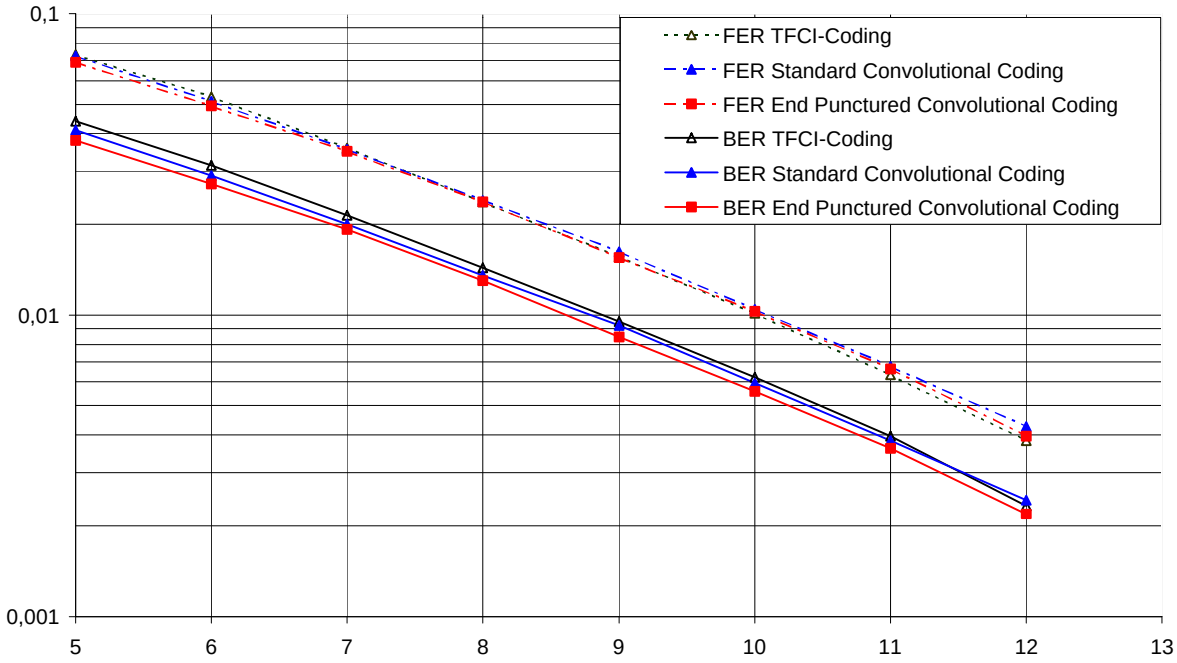

Fig. 8: Performance of end-puncturing for very short block size of 3 bits, compared to standard convolutional coding and to standard TFCI coding, Fading channel, no PC

With short block sizes it is difficult to obtain a large coding gain and therefore the performance in terms of required Eb/N0 must be expected to be worse for small block sizes compared to large ones. This observation is applicable to any coding scheme, the larger the block size the more potential coding gain can be achieved. While this is of course also the case for the proposed scheme, it still maintains the best performance for very short blocks from all the available options.

End puncturing used to aid radio frame equalisation

In the uplink, radio frame size equalisation is performed by padding the input bit sequence with dummy bits in order to ensure that the output can be segmented in F_i data segments of same size where F_i denotes the number of radio frames in one Transmission Time Interval of the TRCH. This operation is necessary for the following interleaver.

Obviously padding is not very efficient, in particular for small blocks, because it adds transmission energy without information content. End puncturing can be tailored to substitute radio frame equalisation without the disadvantage of inserting dummy bits. All that needs to be done is shorting the end puncturing pattern appropriately. This is a very elegant way to achieve a suitable number of bits for interleaving.

Fig. 9 shows the relative overhead of the padding bits expressed in dB. The actual gain depends on the TTI, the block size and the coding rate. An overhead of up to 0.2 dB can be observed for block sizes as large as 60 bits.

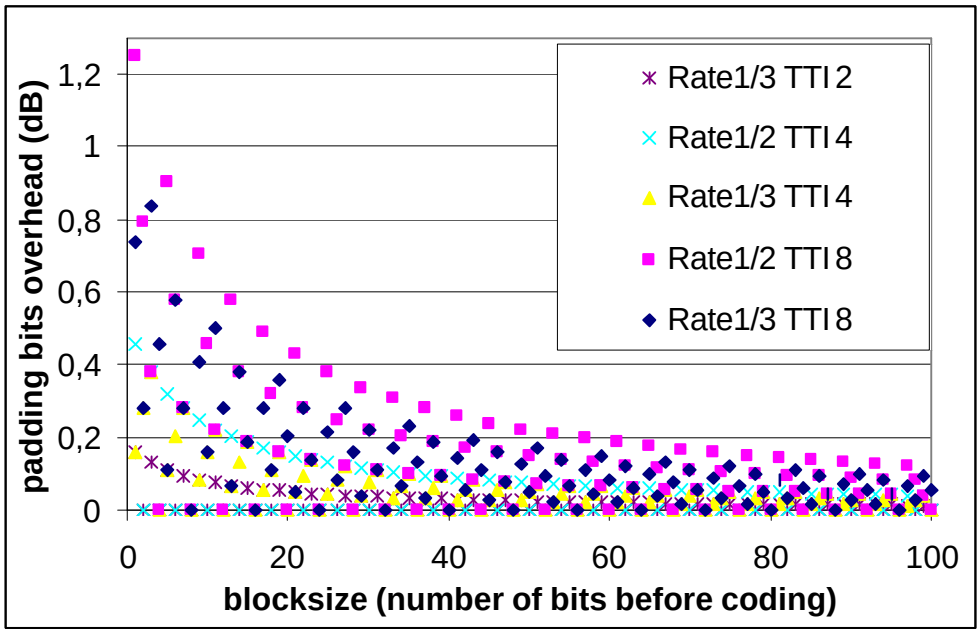

Fig. 9: Overhead caused by padding bits for radio frame equalisation.

Alternate end puncturing implementation, integrated into rate matching

There is one sub optimum aspect with the proposed end puncturing method, that is the fact that it may be followed by additional puncturing or repetition in the conventional rate matching step, depending on the available number of bits for transmission over the radio link. It should be noted that this does however not turn the performance advantage of the end puncturing scheme into a disadvantage, it only means that the full potential advantage cannot be utilised in all cases. This has been shown by the simulations provided in [1].

There would be an advantage, if the end puncturing scheme could be integrated into the normal rate matching scheme. This would require that the rate matching scheme is extended to not only allow uniform puncturing or repetition but also more heavy puncturing at the end of the coded bits and less heavy puncturing in the middle. For the case of repetition this would mean light repetition at the end but more repetition in the middle. If the number of bits after rate matching is very close to the number of input bits, it may even be optimum to do some (light) puncturing at the ends compensated by some repetition in the middle, i.e. applying both puncturing and repetition in the same frame but not one after the other but in different regions of the frame. Of course, the full flexibility of the present rate matching scheme, i.e. the possibility to match any number of input bits in an optimum way to any number of output bits must be retained.

It turns out that the rate matching formula in [3],[4] is versatile enough to be extended for these requirements. The amount of puncturing/repetition is basically determined by the relation of the parameters e_{plus} and e_{minus}. In the present formulation of the algorithm the parameter e_{minus} is fixed for all bits of a transport channel. If e_{minus} is allowed to be varied and takes different values for the bits at the end of a block as compared to the bits in the middle of the block, then different local repetition/puncturing rates can be achieved in the different areas, while the repetition/puncturing patterns within such an area are still evenly distributed. Even at the border between these areas there is no discontinuity (i.e. in particular no repetition/puncturing of adjacent or very close bits and also no extremely long gap).

This simple modification does not allow puncturing and repetition in the same frame, but even this is possible with some modification of the algorithm. In this way we can also avoid having two separate algorithms, one for repetition and the other for puncturing which are so similar that one would expect that they could be easily merged into a single algorithm.

We present here the proposed algorithm and will then discuss it further:

$e = e_{ini} = 1$	-- initial error between current and desired puncturing ratio -- this is now proposed to be set to 1, see [5] and the CRs [6],[7]
$m = 1$	-- index of current bit

```
do while m <= N
    e = e – e_minus(m)              -- update error
    do while e <= 0                 -- check if and how often bit number m should be sent
        select bit x_m for transmission
        e = e + e_plus              -- update error
    end do
    m = m + 1                       -- next bit
end do
```

In the inner loop a bit is selected for transmission, if e <= 0. In this way puncturing, ordinary transmission and any number of repetitions can be realised, depending on the value of e when entering the inner loop:

- If $e > 0$ the bit is not selected at all, in other words, the bit is punctured.
- If $-e_{plus} < e <= 0$ the bit is selected exactly once, in other words, the bit is neither punctured nor repeated.
- If $e <= -e_{plus}$ the bit is selected more than once, in other words, the bit is repeated.

The parameter e_{minus} or more specifically its relation to e_{plus} determines how often a bit is transmitted. It will be transmitted between $\lceil e_{minus}/e_{plus} \rceil$ and $\lfloor e_{minus}/e_{plus} \rfloor$ times. If e_{minus}/e_{plus} is not an integer the algorithm will maintain an smooth distribution of bits that are transmitted above and below their nominal "quota" so that the total transmitted energy, when averaged over a few consecutive bits, will be as close as possible to the desired value.

For practical reasons we would suggest to only consider integer values for e_{minus} and e_{plus}. We further introduce the relative importance or weight of every individual bit and call it w(m), again we suggest integer values for simplicity of the implementation. As in [3] and [4], N_i denotes the number of bits before rate matching and N_c the number of bits after rate matching. Then individual weights w(m) can be selected as appropriate for the coded block. For the rate matching step the parameters $e_{minus}(m)$ and e_{plus} are then computed as:

$$e_{minus}(m) = w(m)\, N_c$$

$$e_{plus} = \sum_{m=1}^{Ni} w(m)$$

With this selection the number of transmitted bits is N_c, as required:

$$\sum_{m=1}^{Ni} e_{minus}(m) \Big/ e_{plus} = N_c \sum_{m=1}^{Ni} w(m) \Big/ \sum_{m=1}^{Ni} w(m) = N_c$$

To obtain the effect of end puncturing, the weights w(m) of the bits at both ends of the blocks can be assigned a predefined higher value than the bits in the middle which would all receive the same value. In this way all the regions can be transmitted with optimum energy without the risk that there is too heavy or too light puncturing or repetition locally due to the interaction of two uncoordinated puncturing/repetition operations.

Conclusion

This paper has shown, that end puncturing has very appealing properties regarding performance. This advantage applies both for rate 1/2 and rate 1/3 codes. Further more, the performance for very short codes (down to e.g. 3 bits only) is also enhanced.

The end puncturing operation can be implemented very simply right after convolutional coding.

End puncturing can be applied for fixed positions as well, but only the bits in the beginning (not in the end) should be punctured.

We therefore propose to implement the end puncturing scheme (possibly under control of higher layers) immediately after convolutional coding and before interleaving or rate matching. This principle is also intended to supplement the toolbox to support Unequal Error Protection within layer 1 under higher layer control.

We have also shown, that end puncturing can be combined with the standard rate matching, however, given the time pressure to finalise release 99, we would propose to study this scheme for release 2000.

References

[1] Siemens, "New puncturing scheme for convolutional codes"; TSG-RAN Working Group 1 meeting No. 8, Tdoc R1-99G51, October 12-15, New York, USA

[2] Nortel Networks, "Proposal for channel coding scheme for small block sizes"; TSG-RAN Working Group 1 meeting No. 8, Tdoc R1-99G46, October 12-15, New York, USA

[3] 3GPP TS 25.212: "Multiplexing and channel coding (FDD)" (1999-10)

[4] 3GPP TS 25.222: "Multiplexing and channel coding (TDD)" (1999-10)

[5] LGIC, "Simulation Results of Convolutional Code Puncturing with Initial Offset of '1'"; TSG-RAN Working Group 1 meeting No. 8, Tdoc R1-99f48, October 12-15, New York, USA

[6] LGIC "Revised CR to 25.212 for initial offset value change for convolutional code rate matching"; TSG-RAN Working Group 1(Radio) meeting #9, Tdoc R1-99j11, 30 November – 3 December 1999, Dresden, Germany

[7] LGIC "Revised CR to 25.222 for initial offset value change for convolutional code rate matching"; TSG-RAN Working Group 1(Radio) meeting #9, Tdoc R1-99j12, 30 November – 3 December 1999, Dresden, Germany

Appendix, Text proposal for TS25.212

In the following pages we describe the changes that are needed for the introduction of this scheme.

4.2.3 Channel coding

Code blocks are delivered to the channel coding block. They are denoted by $o_{ir1}, o_{ir2}, o_{ir3}, \ldots, o_{irK_i}$, where i is the TrCH number, r is the code block number, and K_i is the number of bits in each code block. The number of code blocks on TrCH i is denoted by C_i. After encoding the bits are denoted by $y_{ir1}, y_{ir2}, y_{ir3}, \ldots, y_{irY_i}$. The encoded blocks are serially multiplexed so that the block with lowest index r is output first from the channel coding block. The bits output are denoted by $c_{i1}, c_{i2}, c_{i3}, \ldots, c_{iE_i}$, where i is the TrCH number and $E_i = C_i Y_i$. The output bits are defined by the following relations:

$c_{ik} = y_{i1k}$ $k = 1, 2, \ldots, Y_i$

$c_{ik} = y_{i,2,(k-Y_i)}$ $k = Y_i + 1, Y_i + 2, \ldots, 2Y_i$

$c_{ik} = y_{i,3,(k-2Y_i)}$ $k = 2Y_i + 1, 2Y_i + 2, \ldots, 3Y_i$

…

$c_{ik} = y_{i,C_i,(k-(C_i-1)Y_i)}$ $k = (C_i - 1)Y_i + 1, (C_i - 1)Y_i + 2, \ldots, C_i Y_i$

The relation between o_{irk} and y_{irk} and between K_i and Y_i is dependent on the channel coding scheme.

The following channel coding schemes can be applied to TrCHs:

- Convolutional coding
- Turbo coding
- No channel coding

The values of Y_i in connection with each coding scheme:

- Convolutional coding, ½ rate: $Y_i = 2*K_i + 16$ - N_{EP}; *1/3 rate:* $Y_i = 3*K_i + 24$ - N_{EP}
  N_{EP} is defined in section 4.1.3.1.1.
- Turbo coding, 1/3 rate: $Y_i = 3*K_i + 12$
- No channel coding, $Y_i = K_i$

Table 1: Error Correction Coding Parameters

Transport channel type	Coding scheme	Coding rate
BCH	Convolutional code	1/2
PCH		
FACH		
RACH		
CPCH		1/3, 1/2 or no coding
DCH		
CPCH	Turbo Code	1/3 or no coding
DCH		

4.1.3.1 Convolutional coding

4.1.3.1.1 Convolutional coder

- Constraint length K=9. Coding rate 1/3 and ½.

- The configuration of the convolutional coder is presented in figure 3.

- The output from the convolutional coder shall be done in the order output0, output1, output2, output0, output1, …,output2. (When coding rate is 1/2, output is done up to output 1).

- K-1 tail bits (value 0) shall be added to the end of the code block before encoding.

- The initial value of the shift register of the coder shall be "all 0".

- If end puncturing is applied, the number of bits to be punctured (N_{EP}) is calculated as indicated in table 2:

- The N_{EP} bits on the first N_{EP} positions listed in table 3 (counting from 0 for the first bit from output(0) of the resulting outputstream after coding are punctured:

Table 2: Number of Bits N_{EP} to be punctured from End Puncturing Patterns

		Uplink	Downlink
Rate 1/2	Fixed positions of the TrCHs	Not applicable	4
	Flexible positions of the TrCHs	$(2*K_i + 15) \bmod F_i + 9 - F_i$	8
Rate 1/3	Fixed positions of the TrCHs	Not applicable	8
	Flexible positions of the TrCHs and $K_i > 3$	$(3*K_i + 23) \bmod F_i + 17 - F_i$	16
	Flexible positions of the TrCHs and $K_i < 4$	$(3*K_i + 19) \bmod F_i + 13 - F_i$	12

Table 3: End Puncturing Patterns

Rate 1/2	Fixed positions of the TrCHs	2, 4, 8, 9,
	Flexible positions of the TrCHs	2, $2*K_i$ +13, 4, $2*K_i$ +10, 8, $2*K_i$ +9, 9, $2*K_i$ + 7
Rate 1/3	Fixed positions of the TrCHs	0, 1, 3, 5, 7, 10, 13, 16
	Flexible positions of the TrCHs	0, $3*K_i$ +23, 1, $3*K_i$ +22, 3, $3*K_i$ +20, 5, $3*K_i$ +18, 7, $3*K_i$ +16, 10, $3*K_i$ +13, 13, $3*K_i$ +10, 16, $3*K_i$ + 7