TSG-RAN Working Group 1(Radio) meeting #3
Nynäshamn, Sweden 22-26, March 1999

TSGR1#3(99) 203

Agenda Item: Adhoc 4 item 7
or 9.2 New contributions: Multiplexing and channel coding

Source: **Siemens**

Title:

Optimised Rate Matching after interleaving

Document for: **Discussion**

Introduction

The interleaving in the transport multiplexing scheme is performed in two steps. In Nortel, "Discussion on channel interleaver for 3GPP selection", TSG W1 #2 (99)106., Nortel provided some pros and cons for different implementations of this division. In [5] Ericsson highlights the implications of the different solutions have some implications in uplink, and proposes a modified puncturing scheme to remedy this. In [6] Phillips have suggested a further modification, to further improve the performance, which was further elaborated in [7]. This paper will show that the puncturing pattern is still not optimum in all cases and suggests a further modification to arrive at an algorithm which works satisfactory in all cases.

FS-MIL in uplink

In ETSI the assumption has been that puncturing is allowed in both uplink and downlink. When merging the ETSI and ARIB specifications, ARIB's assumption of no puncturing in uplink was put in Editor, "S1.12 v0.0.1, 3GPP FDD, multiplexing, channel coding and interleaving description".. It is believed that puncturing will be useful also in the uplink, for example in order to avoid multicode. There is then a potential problem since if FS-MIL is used in the uplink multiplexing scheme together with the current rate matching algorithm Editor, "S1.12 v0.0.1, 3GPP FDD, multiplexing, channel coding and interleaving description"., the performance could be degraded.

This has been shown in [5] considering, as an example, a case where layer 2 delivers a transport block with 160 bits on a transport channel with transmission time interval 80 ms and assuming that four bits in each frame should be punctured. The result is that 8 adjacent bits will be punctured which is clearly undesirable.

The proposal was to shift the puncturing pattern in each frame. This is equivalent to applying the puncturing before the column shuffling, even if it is actually performed after inter frame interleaving. Indeed, in the above mentioned example, there are no more adjacent bits punctured, as shown in [7].

However, there exist still cases, where adjacent bits are being punctured, depending on the puncturing rate. Consider e.g. the case, where N_i=16, N_c=14, m_1=4, m_2=14, k=1…7, and K=8. For simplicity, only the field before interleaving is shown in Fig. 1. As can be seen, adjacent bits 31-32 and 95-96 are punctured which is clearly undesirable.

0	1	2	3	**4**	5	6	7
8	9	10	11	12	**13**	14	15
16	17	18	19	20	21	**22**	23
24	25	26	27	28	29	30	**31**
32	33	34	35	36	37	38	39
40	**41**	42	43	44	45	46	47
48	49	**50**	51	52	53	54	55
56	57	58	**59**	60	61	62	63
64	65	66	67	**68**	69	70	71
72	73	74	75	76	77	78	79
80	81	82	83	84	85	**86**	87
88	89	90	91	92	93	94	**95**
96	97	98	99	100	101	102	103
104	**105**	106	107	108	109	110	111
112	113	**114**	115	116	117	118	119
120	121	122	**123**	124	125	126	127

Figure 1: 1st interleaving of 80 ms and 1:8 puncturing with improved algorithm proposed in [7].

Principle of optimised algorithm

The goal of a good puncturing algorithm is to spread punctured bits evenly as possible. This was the driving principle for the algorithm in [2] as well. This can best be obtained by puncturing every nth bit (for non integer puncturing rates sometimes every nth and sometimes every n+1st bit). We can try to apply this principle also for puncturing after interleaving, but there is one constraint: We have to distribute punctured bits on all frames evenly. For example, assume 80 ms interleaving and a puncturing rate of 1:6. By puncturing every 6th bit we would only puncture column 0,2,4,6 but not 1,3,5,7 which is of course impossible. To balance puncturing between columns, we have to change the puncturing interval sometimes (here once) to avoid hitting always the same columns. This is shown in Fig. 2. Bold horizontal arrows show puncturing distance of 6 and the thick hollow arrow shows puncturing distance 5 to avoid hitting the first column twice. After having punctured every column once, the pattern can be shifted down by 6 rows to determine the next bits to be punctured (vertical arrows). Obviously this is equivalent to puncturing every 6th bit in each column and shifting puncturing patterns in different columns relative to each other as already proposed in [5, 6, 7], but the amount of column shifting is now determined differently.

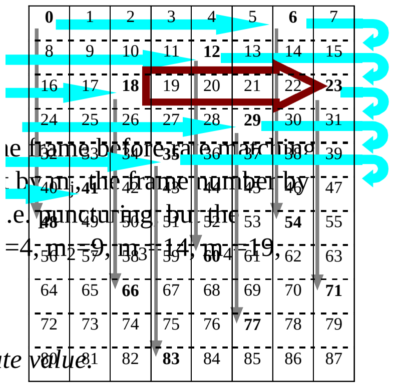

Figure 2: Principle of optimised puncturing

Formulas for optimised algorithm

We now present the formulas for the optimised algorithm: Denote the numbe he frame before rate matching by N_i, the number of bits after rate matching by N_c, the index to the puncture t by m_i, the frame number by k, and the number of interleaved frames by K. We mainly consider the case v .e. puncturing, but the formulas will be applicable for repetition as well. In the example above N_i=2 =4, m=9, m=14, m=19, k=1…7, and K=8. Shifting could then be achieved with the following formula:

-- calculate average puncturing distance
q:= *(ëN_c/(ôN_i-N_cô)û) mod K* -- *where ë û means round downwards and ô ômeans absolute value.*
Q:= *(ëN_c/(ôN_i-N_cô)û) div K*
if q is even -- *avoid hitting the same column twice:*
  then q = q – 1 / lcd(q, K) -- where lcd (q, K) means largest common divisor of q and K
  -- note lcd can be easily computed using bit manipulations, because K is a power of 2.
  -- for the same reason calculations with q can be easily done using binary fixed point
  -- arithmetic (or integer arithmetic and a few shift operations).
endif
– calculate S and T, S represents the shift of the row mod K and T the shifting amount div K

for *i = 0* to *K-1*
  *S(R_K (éi*qù* mod *K)) = (éi*qù* div *K)* -- *where é ù means round upwards.*
  *T(R_K (éi*qù* mod *K)) = i* -- *R_K(k) reverts the interleaver as in [7]*
end for

In a real implementation, these formulas can be implemented as a lookup table as shown below. The table also includes the effect of re mapping the column randomising achieved by *R_K(k)*. Obviously S can also be calculated from T, yet an other implementation option.

S ; T	K	1	2		4				8							
	k	0	0	1	0	1	2	3	0	1	2	3	4	5	6	7
Q	1	0;0	0;0	0;1	0;0	0;2	0;1	0;3	0;0	0;4	0;2	0;6	0;1	0;5	0;3	0;7
	2		0;0	1;1	0;0	0;1	1;3	0;2	0;0	0;2	0;1	0;3	1;5	1;7	1;6	0;4
	3				0;0	1;2	2;3	0;1	0;0	1;4	2;6	0;2	1;3	2;7	0;1	1;5
	4				0;0	1;2	2;3	0;1	0;0	0;1	2;5	1;4	3;7	2;6	1;3	0;2
	5								0;0	3;4	2;2	4;6	4;5	1;1	5;7	2;3
	6								0;0	1;2	2;3	0;1	5;7	3;5	4;6	2;4
	7								0;0	3;4	5;6	1;2	6;7	2;3	4;5	0;1
	8								0;0	3;4	5;6	1;2	6;7	2;3	4;5	0;1

Then, e_{offset} can be calculated as

$$e_{offset}\,(k) = ((2*S + 2*T* Q +1)* y + 1) \bmod 2Nc$$

e_{offset} (k) is then used to pre load *e* in the rate matching formula in [2].

This algorithm will obtain the perfect puncturing as if puncturing using the rate matching algorithm was applied directly before interleaving, if the puncturing rate is an odd fraction i.e. 1:5 or 1:9. For other cases, adjacent bits will never be punctured, but one distance between punctured bits may be larger by up to lcd(q,K)+1 than the other ones. Note that this algorithm should be applied to bit repetition as well as already suggested in [7]. While repeating adjacent bits is not as bad as puncturing them, it is still advantageous to distribute repeated bits as evenly as possible.

The basic intention of these formulas is to try to achieve equidistant spacing of the punctured bits in the original order, but taking into account the constraint, that the bits have to be punctured equally in different frames. This may make it necessary to reduce the puncturing distance by 1 sometimes. The presented algorithm is optimum in the sense, that it will never reduce the distance by more than 1, and will reduce it only as often as necessary. This gives the best possible puncturing pattern under the above mentioned constraints.

The following is an example using the first set of parameters i.e. puncturing by 1:5 (Fig. 3, left). Obviously the optimised algorithm not only completely avoids puncturing adjacent bits, it also distributes punctured bits with equal spacing in the original sequence. In fact the same properties are achieved, as if the puncturing had been done directly after coding before interleaving.

0	1	2	**3**	4	5	6	7
8	9	**10**	11	12	13	14	15
16	**17**	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	**39**
40	41	42	43	44	45	**46**	47
48	49	50	51	52	**53**	54	55
56	57	58	59	**60**	61	62	63
64	65	66	**67**	68	69	70	71
72	73	**74**	75	76	77	78	79
80	**81**	82	83	84	**85**	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	103
104	105	106	107	108	109	110	111
112	113	114	115	116	**117**	118	119
120	121	122	123	**124**	125		

Figure 4: Unequal number of bits per frame

0	1	**2**	3	4	5	6	**7**
8	9	10	11	**12**	13	14	15
16	**17**	18	19	20	21	**22**	23
24	25	26	**27**	28	29	30	31
32	33	34	35	36	**37**	38	39
40	41	**42**	43	44	45	46	**47**
48	49	50	51	**52**	53	54	55
56	**57**	58	59	60	61	**62**	63
64	65	66	**67**	68	69	70	71
72	73	74	75	76	77	78	79
80	81	**82**	83	84	85	86	**87**
88	89	90	91	**92**	93	94	95
96	**97**	98	99	100	101	**102**	103
104	105	106	**107**	108	109	110	111
112	113	114	115	116	**117**	118	119
120	121	**122**	123	124	125	126	**127**
128	129	130	131	**132**	133	134	135
136	**137**	138	139	140	141	**142**	143
144	145	146	**147**	48	149	150	151
152	153	154	155	156	**157**	158	159

0	1	2	**3**	4	5	6	7
8	9	**10**	11	12	13	14	15
16	**17**	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	**39**
40	41	42	43	44	45	**46**	47
48	49	50	51	52	**53**	54	55
56	57	58	59	**60**	61	62	63
64	65	66	**67**	68	69	70	71
72	73	**74**	75	76	77	78	79
80	**81**	82	83	84	85	86	87
88	89	90	91	92	93	94	95
96	97	98	99	100	101	102	**103**
104	105	106	107	108	109	**110**	111
112	113	114	115	116	**117**	118	119
120	121	122	123	**124**	125	126	127

Figure 3: 1st interleaving of 80 ms and 1:5 puncturing (left) and 1:8 puncturing (right) with proposed algorithm.

Let us now investigate the next case i.e. puncturing by 1:8 (Fig. 3, right). Again puncturing of adjacent bits is avoided. In this case it is not possible to obtain an equidistant puncturing because then all bits of one single frame would be punctured, which is totally unacceptable. In this case most of the distances between adjacent bits are 7 (only one less than would be the case with an optimum distribution). Some distances are larger (every eighth) in exchange.

Change of rate matching during the transmission time interval

There are two cases, where the rate matching can change during the transmission time interval:

a) The number of input bits is not divisible by K. Then the last frames will carry one input bit less than the first ones and therefore also have a slightly lower puncturing rate. Note that it is not clear, whether this case will be allowed or whether the coding will be expected to deliver a suitable number.

b) Due to fluctuations in other services which are multiplexed on the same connection the puncturing must be changed in later frames.

In these cases the balanced puncturing scheme could still suffer. Due to the unpredictable nature of case b) it seems unlikely, that any scheme can be found, which could lead to a near perfect puncturing pattern, so here we may have to live with some unpredictable behaviour anyhow. In case a) however, we propose not to change the puncturing pattern in the last rows. Instead we suggest to use the same puncturing algorithm as for the first columns, but simply omit the last puncture.

Consider as an example that 125 input bits are to be punctured to give 104 output bits, interleaved over 8 frames. Then the puncturing pattern would look like shown in Fig. 4. The last columns have one less input bit than the first ones, by omitting the last puncture, the columns all have 13 bits.

Optimised 1st interleaver in uplink

There also is an alternative proposal to use an optimised 1st interleaver, and use a simple 2nd interleaver and a simple puncturing scheme. This relies on the expectation, that an optimised interleaver will distribute bits in a way that puncturing blocks of bits after interleaving will spread these punctured bits evenly before interleaving. However, the experience with puncturing after a simple 1st interleaver tells, that this is not an easy task. As the single interleaver can not be optimised for all puncturing rates it is next to impossible, that good properties can be achieved: The reason is as follows: The puncturing patterns for n+1 bits must be identical to the puncturing pattern for n bits, but one additional bit can be selected for puncturing. If the puncturing pattern for n bits is good (see firs row in the table below), then which ever bit is punctured to get n+1 bits (second row), it is impossible to come close to an optimum distribution of n+1 bits (last row).

		■					■					■					■			Best solution to puncture n bits
		■					■			🟥		■					■			Puncture n+1 bits as above plus one extra bit
	■				■				■				■				■			puncture n+1 bits with optimised algorithm

Further more such an interleaver would have to be a compromise between good puncturing properties for block puncturing and good general interleaving properties at the same time. Finding a scheme that optimally satisfies both constraints seems impossible.

Concluding we think that such a optimised 1st interleaver will unfortunately not exist, so we have to use the other alternative i.e. puncturing after a simple 1st interleaver followed by a second interleaver with optimised interleaving properties.

Summary

This paper has shown that near optimum puncturing (or repetition) patterns are possible when applying rate matching after first interleaving. The necessary algorithm is not very complex, it is similar to the puncturing algorithm itself but has to be executed once per frame only, not once per bit.

References

[1] Nortel, “Discussion on channel interleaver for 3GPP selection”, TSG W1 #2 (99)106.

[2] Editor, “S1.12 v0.0.1, 3GPP FDD, multiplexing, channel coding and interleaving description”.

[3] Siemens, “Proposal for Combined Static- and Dynamic Rate Matching”, Tdoc SMG2 UMTS-L1 430/98.

[4] Philips, “Service Multiplexing”, Tdoc SMG2 UMTS-L1 229/98.

[5] Ericsson, “Two step interleaving” 3GPP TSG RAN W1 AdHoc 4, Transport channel multiplexing, March 10, 1999

[6] Phillips, “ Re: Ad Hoc 4: Item 7”, 3GPP_TSG_RAN_WG1@LIST.ETSI.FR, sent March 10 1999 18:20

[7] NTT DoCoMo "Modified Rate Matching Algorithm in uplink" 3GPP RAN TSG WG1 Ad Hoc 4; March 15th, 1999