TSG-RAN Working Group1 meeting #7 *TSGR1#7(99)b58*
Hannover, Germany,
August 30 – September 3, 1999

Source: Samsung

Title: **Discussion on Multiple Scrambling Code**

Document for: Discussion

---

1. Abstract

SAMSUNG proposed the scrambling code generation method with the mask operation[1]. In terms of this issue, there is some E-mail discussion as followings.

2. E-mail discussion

Date: Tue, 20 Jul 1999 17:53:54 +0100
From: "Chambers, Peter" <peter.chambers@ROKE.CO.UK>
Subject: AH10 : R1-99915 "Multiple scrambling code"

I wish to discuss Tdoc 915 (Samsung) which is best discussed by e-mail rather than at a meeting. This paper proposes a non-trivial change to the working assumption on downlink scrambling code generation. I wish to show that the problems discussed by the proponents are not so severe as they write and that it is safest to keep to the working assumption.
Firstly the proponents discuss some "problems with the current method":

- they say that the simplest method is to have multiple generators but with a masking function that complexity is decreased, but
  i) this is an implementational matter and not a standards issue. Manufacturers are free to use variable masking if they wish though the extra gates add complexity.
  ii) the complexity of the current Gold code generator is minimised in terms of the fixed masking function suggested. A general masking function would add complexity especially if it were run time programmable.
- the proponents guess that Tdoc 724 is motivated by a mapping designed to facilitate a mapping function based on a mapping "between number and real code". This does not seem to be the case as the scrambling code number is simply the number used to initialise the scrambling code generator and is a linear function of primary and secondary scrambling code numbers. The complexity of initialisation is very low in hardware and software. No masking is required or seems to be implied.
- the proponents see problems with over provision of secondary scrambling codes by layer 1. If this is problem with higher layer signalling then a limited number only need by used. The proponents suggest 4 secondary codes might be used. For spot beams slightly more may be used. However over provision is no problem. Lack of provision would be a problem.

In the section dealing with the current method two objections are raised:
1. code0 would be a problem for a masking implementation.
2. there is no simple rule to find masking in the current scheme (manufacturers would have to implement one by calculation)

These are really problems with the masking approach which re-use the prime movers of m-sequence generators using linear algebra to provide shifted sequences. This is not a standards issue but the limitation of the hardware architecture solution propounded in the paper. Other

architectures which have been discussed by several companies (both in Fibonacci and Galois forms) do not suffer from this problem. Indeed in low power UEs the current text is more suitable for hardware than it might be (forDSPs) in the base station.
For the above reasons I would suggest that Tdoc 915 not be approved.

[SAMSUNG]

Thank you for having an interest in our proposal. I think there are some misunderstanding in our proposal. Please see below.
Actually, there were supports from several companies like Ericsson and Panasonic during offline discussion. Only comment in the meeting was the number of secondary scrambling code from Ericsson. They want to have 15 as maximum, and we don't have problem with that. Our proposal includes one change and two small suggestions for the current text.
Let me address two small suggetion first.

1.
> - the proponents see problems with over provision of secondary scrambling codes by > layer 1. If this is problem with higher layer signalling then a limited number only need by > used. The proponents suggest 4 secondary codes might be used. For spot beams slightly > more may be used. However over provision is no problem. Lack of provision would be a > problem.
There is an editor's note in the current text about uncertainty of the number of secondary scrambling code.
This issue is supposed to be handeled before we fix release 99 to reduce signaling overhead. So, we suggest 4 secondary scrambling code as maximum, and Ericsson suggested 15. I think 15 is a very safe number, and I am o.k. with that. Some other company can make comment regarding this. I think that deciding the number of secondary scrambling code is a L1 issue not a higher layer because that decision will be based on the real capacity.

2.
> - the proponents guess that Tdoc 724 is motivated by a mapping designed to facilitate a > mapping function based on a mapping "between number and real code". This does not > seem to be the case as the scrambling code number is simply the number used to initialise > the scrambling code generator and is a linear function of primary and secondary > scrambling code numbers. The complexity of initialisation is very low in hardware and > software. No masking is required or seems to be implied.

The reason behind our guess is following. When we use secondary scrambling code, both BS and UE must have at least two generators at the same time. One for primary, the others for secondary For example, one for BCH, one for DCH. Initialization is of course easy, but we don't want to have separate generators, so initialization for second generator is not necessary. Using maskng function is pretty well known technic for this case. We can have second generator with almost no complexity increase if a masking function is simple whatever implementation is done with a hardware or a software. Furthermore, calculating masking function by two initial state is not possible. So, we suggest a rephrase of the current text.

3. Therefore, we proposed to change the current way to assign primary and secondary scrambling codes.
Option 2 seems to be the simplest and the best way to minimize complexity.

> they say that the simplest method is to have multiple generators but with a masking > function that complexity is decreased, but ii) this is an implementational matter and not a > standards issue. Manufacturers are free to use variable masking if they wish though the > extra gates add complexity.
> iii) the complexity of the current Gold code generator is minimised in terms of the fixed > masking function suggested. A general masking function would add complexity especially > if it were run time programmable.

Regarding first comment, Some issue is related with implementation complexity. Are you saying implemetation complexity is not important? Then, I like to hear about your opinion on Nokia's contribution Tdoc 588. That is also about suggestion to decrease complexity and I think it is also valuable.
With the current method, we can not avoid some complexity increase whatever the method is (Hardware, software, Fibonacci or Galois) We need a rule to assign codes anyway in the text, then we should have
a nice rule to minimize a complexity. Regarding second comment, I think you assumed seperate secondary scrambling code gerneerator which is not our subject.

> In the section dealing with the current method two objections are raised:
> 1. code0 would be a problem for a masking implementation.
> 2. there is no simple rule to find masking in the current scheme (manufacturers would have > to implement one by calculation)
> These are really problems with the masking approach which re-use the prime movers of m- > sequence generators using linear algebra to provide shifted sequences. This is not a > standards issue but the limitation of the hardware architecture solution propounded in the > paper. Other architectures which have been discussed by several companies (both in > Fibonacci and Galois forms) do not suffer from this problem. Indeed in low power UEs the > current text is more suitable for hardware than it might be (for DSPs) in the base station.
Actually, I don't understand whole pharagraph because the current description of the text is in Fibonacci form. Moreover, regardless of the expression, all problems we mentioned are valid. Fibonacci form is different from Galois form in terms of H/W logic. But two different forms are specific representation forms of LFSR sequence, and they are equivalent each other in terms of LFSR Theory. (See "Shift Register", - Golomb, Holiday) Mask problem depend on a base finite field of sequence, not, H/W logic.
That is, the masking operation is based on the fact that a shift version of LFSR sequence can be represented by a sum of some shift version. The reason is as follows.
Actually, LFSR sequence is based on the finite field, and there is a one-to-one mapping of each shifted version onto exactly one element in finite fields. Therefore, a finite field is equivalent to a set of all shifted versions of LFSR sequence.
Since the finite field is a vector space, there is a basis element of a finite field, and the corresponding shifted version is a basis element of the set of all shifted versions of LFSR sequence. Hence, all shifted versions of LFSR sequence are represented by a linear combination of some basis element. This proposal recommend simply to change the current assignment rule for primary and secondary scrambling code. This proposal does not harm anything, but provide simple and nice implementation.
If you want implement several seperate generators, then you can do that with our proposal. No extra cost. However, if you want to share most of logic, then you will have a benefit from our proposal. I think most of misunderstanding comes from the assumption about secondary scrambling code. Regardless of software or hardware implementation, having several seperate generators is not desirable at all. With option 2, you can only change input from the primary code generator for the generation of secondary scrambling code. No other operation is necessary from the generator of primary scrambling code.

Date: Mon, 9 Aug 1999 14:26:10 +0200
From: Fredrik Ovesjo <Fredrik.Ovesjo@ERA-T.ERICSSON.SE>
Subject: Re: AH10 : R1-99915 "Multiple scrambling code"
Comments: To: "Chambers, Peter" <peter.chambers@ROKE.CO.UK>

As I have understood this paper, option 2 is only a redefinition of what code numbers that belong to the primary and its associated secondary scrambling code group.
Hence, in order to introduce option 2 in the spec, only a very minor change needs to be done (definition of the groups). I do not see that the generation of the codes needs to be redefined in any way. However, I do not want this masking function to become a part of the specification text, since it is an implementation matter.

I indicated to Samsung offline that I would be willing to accept option 2 with the change that we have at least 15 secondary scrambling codes per primary code, and that the text proposal is such that only
the grouping of the code numbers was changed in the spec.

Date: Tue, 10 Aug 1999 12:53:26 +0200
From: Fredrik Ovesjo <Fredrik.Ovesjo@ERA-T.ERICSSON.SE>
Subject: Re: AH10 : R1-99915 "Multiple scrambling code"

I have reread (99)915, and I am no longer sure that I can support option 2 in that document.
It seems that in (99)915 another generation of the code of number K is used than is currently assumed in the specs. To generate code number K currently, one only loads the upper register with the binary equivalent of K. Very simple.
However, in (99)915 it seems (please confirm), that the initial state of the upper register is calculated as a phase shift of the sequence initialised by 0000000000...0001. This would mean that some shifting algorithm would be needed, as previously proposed by Nokia.
I think we have already had that discussion about the shifting algorithm, and that several companies were not too happy with that.
Hence, with this new information (please forgive me for misunderstanding before), I would agree with Peter that we should not accept the proposed scheme in (99)915. Too me it seems that the potential benefits are too small to justify the necessary modifications of our current assumptions.

[SAMSUNG]

Thank you for your comment. As you understand, we proposed shifting scheme before our proposal. There are several reasons to do that. Please think about and give us your comment.
Even though initial state loading looks simple, but there are several problems. Of course, it is o.k. if you want to generate only one scrambling code. However, we need several generators are needed at the same time in real situation. One for primary or one for secondary or one for other cells... Followings are problems of initial state loading method.

1. Calculating masking function from initial state is almost impossible (discrete logarithm problem)
2. Complexity of masking is not desirable.

Of course, it is fine too if you want to have multiple independent generators. I think that requires only one solution and does not provide a flexible and an efficient implementation. we are not happy too.
However, if we simply change a description, then we can have many way to implement. If you want to implement with independent genrator, it is fine. If you want to implement with masking function, it is also easy to do that.
I realized this is more fundamental problem. With current description, I think we have some problem.

Date: Wed, 11 Aug 1999 10:50:41 +0200
From: Fredrik Ovesjo <Fredrik.Ovesjo@ERA-T.ERICSSON.SE>
Subject: Re: AH10 : R1-99915 "Multiple scrambling code"
Comments: To: jaeyoel Kim <kimjy@METRO.TELECOM.SAMSUNG.CO.KR>

I understand now that we need big changes to 25.213 to incorporate your proposal. Earlier I thought that there very only small changes, and since the cost was zero one could go for a solution that could give some (very minor) implementation benefit. However, now understanding that we change the entire scrambling code generation method, I do not like your proposal. For me the deadline regarding the
initialisation method (simple loading vs. algorithm) was at the Cheju meeting. Bringing this issue up once again will lead to implementation delays I am afraid.

Where do you expect implementation benefits by your scheme, in the UTRAN or in the UE? Too me the benefit seems very small indeed. Hence, I do not think there is any problem with the current description,
since that alternative will be just as implementable as the masking alternative.

Date: Wed, 11 Aug 1999 17:21:00 +0200
From: "(Esko Nieminen)" <esko.nieminen@NOKIA.COM>
Subject: AH10: 915 vs 724 II

Option 2 of Tdoc 915 does not change much the current code generation HW provided that a manufactor does not want take advantage of it. Then the same HW as now is just fine. Option 2 and the working assumption differ in a way an initial state of a long code is defined. Option 2 need an extra HW or SW to derive an intial state for a given long code but after that Option 2 is able to use the same long code generator as the current scheme.
Formally Option 2 is nothing more than a book keeping method to allocate initial states of long codes. The idea is that the initial state of the 1st secondary code is obtained by one forward shift of the initial state of its primary code, the initial state of the 2nd secondary is obtained by one shift of the initial state of the 1st secondary code, and so on. This way it is possible to achive some HW benefits at least for Fibonacci-type registers.
Where this book keeping is going to happen ? A natural candidate is a BS. We can apply Option 2 such a way that a BS does a preprosessing needed to map a long code number N to its initial state n, then the initial state information is sent over air to a UE, just like in the current scheme. However, in this way we have Option 2 to use if desired.
But we need to rewrite some parts of Specs in order to have Option 2 in any case but L1 would stay as it is now up to some details on grouping rules for long codes.
Our time schedule is a problem for us anyway.

Date: Wed, 11 Aug 1999 17:46:43 +0100
From: "Chambers, Peter" <peter.chambers@ROKE.CO.UK>
Subject: Re: AH10: 915 vs 724 II - "Multiple scrambling codes"

Now we have had a chance to examine the problems and concerns raised in Tdoc 915. We can take stock of what we can do and the problems (including timescale that face us).

1. as Esko points out the initial load values must be chosen carefully. This must occur along the m-sequence.
2. we face strict timescale pressures as Fredrik points out so cannot re-design the generator radically.
3. the current scheme is optimised for separate generators. Samsung point out that there is another way and want the standard to be more technology neutral.

In the spirit of compromise I offer the enclosed draft paper. I hope I have made the terminology clear in this. If there are ambiguous statements I will be happy to clarify. This is a topic in which terminology has confused many of us in the discussion!
In the paper I hope that I have outlined a definition for an (equivalent) generator which can produce one
primary and 16 secondary codes, is a minor modification of the current scheme (when only one code is needed), can be optimised for hardware complexity, has the same DSP (initialisation)complexity as at present and can (in a standardised way) support the "bookkeeping" mentioned below.
No doubt my esteemed colleagues will point out errors in my paper during the discussion. As you will see I go into some detail. I do not think we can have consensus by changing the description without estimates of impact on at least a reference implementation.
As far as changes to the standards are concerned I wish the minimum impact to timescale. If the present scheme can be made to work that is good.
This is offered to provide something quickly, if needed.

[SAMSUNG]

To all ;
It seems that everybody is back from vacation and many people get involved in this issue now.
Let me speak my opinion briefly.
I think people start to think the possibility for generating mutiple scrambling code simultaneously. To me, simple loading is not that simple when we consider to generaton of multiple codes at the same time.
As I said before, initial state loading cannot be implemented by a masking function because calculation is almost impossible. So, we need several independent generators for primary, secondary or other primary scrambling code. This complexity burden goes to UE rather than Utran. I think this gives us only one way to implement and make impossible to implement by software.
What I don't understand is why people think we need a big change. We just need to change a description for initial state a little bit. Furthermore, we can also have the same implementation method with the current one.(several independent generators with rather simple calculation as esko mentioned before) We don't have to change a figure in the text.

To Esko ;
I think you understand most of issue correctly. There is small error in your second paragraph. I think you did not intend. Option 2 is not idea of shift of initial state but shift of sequence since this option assume shifting scheme. I think there are several possible ways to keep the current implementation (initial state loading) with new descrption.
1. Utran and UE get initial state with simple calculation since offset distance is known and constant.
2. Utran and UE get initial state by shifting upper generator, it will delay 8192 cycle for maximum since we only use 16*512 =8192 codes.
3.Utran calculate state and let UE know as you mentioned. In this case we need 18 bits to broadcast and UE may have no idea if it does not have an information for other cell.

To Peter ;
I appricate your effort to compromise. I think your figure 3 looks like option 3 of our proposal. Would you elaborate this little bit more? I think whatever we have a initial state loading method or shifting method, we have no uncertainty about distinction. We can define 2^18 distinct codes in both methods.
Would you explain this too?

Date: Fri, 13 Aug 1999 16:30:00 +0200
From: "(Esko Nieminen)" <esko.nieminen@NOKIA.COM>
Subject: AH10: 915 vs 724 - multiple DL codes

To keep discussion less virus dependent I use ascii tools for communication. I am away until 6-Sep-99.
When it comes to a shift of an initial load value or a shift of an m-sequence, they are more or less the same thing, it just depends on a point of view. Nevertheless, definitions for long codes are good to write using the latter for clarity.
The idea to send initial load values over air during initial synchronization does not work. This can be solved by a lookup table of 512 allowed initial load values, after initial synchronization a UE has a number from 0 to 511, the UE uses the number as an index for the lookup table.
Or the UE uses Nokia or Samsung algorithms or does something else.

Here are some details for further discussion.

1. UTRAN
1.1 Parametrization for long scrambling codes in DL
Denote an m-sequence of x18+x7+1 by A. Initial conditions:
A(17)=A(16)=...=A(1)=0, A(0)=1.

A number of a long scrambling code (long scrambling code = codeword) is denoted by N, it takes values 0,1,2,..., 2^18 - 2. Hence the total number of codewords is 2^18 - 1.
An 18-bit initial load value n=n17n16...n0 of an N-th codeword is given by the Relationship :
n17= A(17+N), n16=A(16+N),..., n0=A(0+N). (1.1.1)

2. UE

2.1 Use of initial load values at UE

A UE uses an initial load value n of an N:th codeword to initialize the m-sequence xn of x18+x7 +1 as follows:
xn(17)=n17, xn(16)=n16, xn(15)=n15,..., xn(0)=n0.

3. Group rules for codewords

Each group consists of 512 primary codes and 15*512 secondary codes, 15 per primary code. The total number of groups is 31, labeled by 0,1,2,...,30 and denoted by G. The primary code parameter PC takes values 0,1,2,...,511 and the secondary code parameter SC from 1 to 15.

The codeword number N(PC,G) for a primary code PC in a group G, G=0,1,2,...,30, is defined as N(PC,G) = G*16*512 + 16*PC, for PC=0,1,...,511, and the codeword numbers of its secondary codes are defined:
N(SC,PC,G) = N(PC,G) + SC, for SC=1,2,...,15.
UTRAN maps N(PC,G) and N(SC,PC,G) to initial load values by 1.1.1, and then sends them to UEs.
Initial synchronization uses the group 11 (I just picked one group!) , hence a UE must be able to apply 1.1.1 for 512 primary codes. OK, by the above definitions we can generate 1+15 codewords, a primary code and its secondary codes, by three shift registers: two for x18+x7+1 and one lower register (Fibonacci feedback).
Moreover, a UE knows initial values of all secondary codes by knowing an initial load value of a primary code (due to definition). I think the current scheme does as well.
Initial load values for initial synchronization can be precomputed by 1.1.1 into a lookup table. This is the only change for a UE due to the new parametrization if minimal amount of changes is desired.
The current definition for initial load values does not walk along an m-sequence with a regular step but jumps with variable random steps, that is why you can not generate 1+15 codes by three LFSRs.
By playing with 1.1.1 it is possible to generate all options of 915 and a lot of new ones.
The current scheme uses N(PC,G) and N(SC,PC,G) directly as initial load values.

[SAMSUNG]

I think you understand most of issue correctly. There is small error in your second paragraph. I think you did not intend. Option 2 is not idea of shift of initial state but shift of sequence since this option assume shifting scheme. I think there are several possible ways to keep the current implementation (initial state loading) with new descrption.
1. Utran and UE get initial state with simple calculation since offset distance is known and constant.
2. Utran and UE get initial state by shifting upper generator, it will delay 8192 cycle for maximum since we only use 16*512 =8192 codes.
3. Utran calculate state and let UE know as you mentioned. In this case we need 18 bits to broadcast and UE may have no idea if it does not have an information for other cell.

3. Reference

[1] TSGR1#6(99)915, ” Multiple Scrambling Code” - Samsung