|  |  |
| --- | --- |
| Project | **IEEE 802.21 MIHS****<**[**http://www.ieee802.org/21/**](http://www.ieee802.org/21/)**>** |
| Title | **An Algorithm for Complete Subtree Creation with Subgroup Ranges** |
| DCN | **21-13-0232-00-MuGM** |
| Date Submitted | **December 30, 2013** |
| Source(s) | Yoshihiro Ohba (Toshiba) |  |
| Re: | IEEE 802.21 Session #60 in Los Angeles |
| Abstract | This document describes an algorithm for generating complete subtrees with subgroup ranges for GKB. |
| Purpose | To addresses LB7a Comment #136. |
| Notice | This document has been prepared to assist the IEEE 802.21 Working Group. It is offered as a basis for discussion and is not binding on the contributing individual(s) or organization(s). The material in this document is subject to change in form and content after further study. The contributor(s) reserve(s) the right to add, amend or withdraw material contained herein. |
| Release | The contributor grants a free, irrevocable license to the IEEE to incorporate material contained in this contribution, and any modifications thereof, in the creation of an IEEE Standards publication; to copyright in the IEEE’s name any IEEE Standards publication even though it may include portions of this contribution; and at the IEEE’s sole discretion to permit others to reproduce in whole or in part the resulting IEEE Standards publication. The contributor also acknowledges and accepts that IEEE 802.21 may make this contribution public. |
| Patent Policy | The contributor is familiar with IEEE patent policy, as stated in [Section 6 of the IEEE-SA Standards Board bylaws](http://standards.ieee.org/guides/opman/sect6.html#6.3) <[http://standards.ieee.org/guides/bylaws/sect6-7.html#6](http://127.0.0.1:4664/cache?event_id=757737&schema_id=1&s=5X0vID10lu_E6yrIkWkNd4Wz2H8&q=hancock)> and in *Understanding Patent Issues During IEEE Standards Development* <http://standards.ieee.org/board/pat/faq.pdf> |

# **Proposal**

(Section numbers are based on those in DCN 227r2.)

*[1] Add the following text to 9.4.2*:

When a GKB contains more than *GkbFragmentThreshold* of Complete Subtrees, it is fragmented into multiple GKB fragments such that each GKB fragment contains at most *GkbFragmentThreshold* Complete Subtrees. Each GKB fragment is associated with exactly one Subgroup Range. The Subgroup Ranges for GKB fragments of the same GKB satisfy all of the following conditions:

1. Union of all Subgroup Ranges is equal to the group range defined as the range of the leaf indices of the group management tree.
2. Intersection of any two Subgroup Ranges is empty.
3. For any Complete Subtree whose range of leaf indices is *L* and any Subgroup Range *R*, intersection of *L* and *R* is either empty or equal to *L*.

An algorithm by which Complete Subtrees and Subgroup Ranges that satisfy these conditions is defined as follows.

def CreateCompleteSubtreeFragments(I, T, R, M):

 # Input I: List of indices of leaf nodes to be included in the group

 # Input T: The entire tree that covers all leaf nodes]

 # Input R: Root node of the entire tree

 # Input M: Maximum number of subtrees in per fragment

 # Output O: List of (S, minr, maxr):

 # S: Subtrees covering the group.

 # minr: Lower bound of Subgroup Range

 # maxr: Upper bound of Subgroup Range

 O=[]

 S=[]

 depth=int(math.log(len(T)+1,2)-1)

 def rightmost\_leaf\_number(n):

 # Input n: subtree root node

 # Output y: rightmost leaf number under the subtree

 h=n.index.len # hierarchy level of node n

 x=int(n.index.val, 2) # node index in decimal

 y=(x+1)\*(2\*\*(depth-h))-1

 return y

 def check(n):

 # Input n: subtree root node

 # Output 0, 1

 # 0 : Some node in the subtree is a non-member of the group.

 # 1 : All nodes in the subtree are members of the group.

 global minr

 rv=0

 if n.left==None and n.right==None: # n is leaf

 if n.index.val in I:

 S.append(n)

 return 1

 return 0

 # n is non-leaf

 lval=check(n.left)

 rval=check(n.right)

 if lval\*rval>0:

 S.remove(n.left)

 S.remove(n.right)

 S.append(n)

 rv=1

 elif lval+rval>0:

 if len(S) > M: # one fragment is ready

 maxr=rightmost\_leaf\_number(S[M-1])

 O.append((S[0:M], minr, maxr))

 S[0:M]=[] # Remove the appended subtrees

 minr=maxr+1

 rv=0

 if n==R: # Root node. len(S)>0

 maxr=2\*\*depth-1

 O.append((S, minr, maxr))

 return rv

 check(R)

 return O

*[2] The following examples may be described in an Annex.*

# **Examples**

[Example 1]

Input of CreateCompleteSubtreeFragments():

* I : ['000', '010', '100', '101', '110']
* T: A group management tree in depth 3
* R: The root node of T
* M: 2

Output of CreateCompleteSubtreeFragments():

Subtree: ( 3 , 000 ) ( 3 , 010 )

Subgroup Range: ( 0 , 2 )

Subtree: ( 2 , 10 ) ( 3 , 110 )

Subgroup Range: ( 3 , 7 )

[Example 2]

Input of CreateCompleteSubtreeFragments():

* I : ['000', '010', '100', '101', '110']
* T: A group management tree in depth 3
* R: The root node of T
* M: 3

Output of CreateCompleteSubtreeFragments():

Subtree: ( 3 , 000 ) ( 3 , 010 ) ( 2 , 10 )

Subgroup Range: ( 0 , 5 )

Subtree: ( 3 , 110 )

Subgroup Range: ( 6 , 7 )

Figure 1 Example of Fragmented Complete Subtrees with Subgroup Ranges

