Project	**IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>**
Title	**LB4 Comment Support**
Date Submitted	**2002-01-04**
Source(s)	Mika Kasslin Nokia Itämerenkatu 11-13 00180 Helsinki, Finland Voice: +358-718036294 Fax: +358-718036856 [mailto:mika.kasslin@nokia.com]
Re:	LB4 on 802.16a/D1
Abstract	This contribution contains text related to the letter ballot comments provided by the author.
Purpose	Discuss the text when addressing the comments.
Notice	
Release	
Patent Policy and Procedures	

LB4 Comment Support

Mika Kasslin
Nokia

Introduction

This paper contains supporting text for the LB4 comments provided by the author. This document is not a standalone document but should be used in parallel with the related commentary file.

Comment 1 related tables:

Table X1 – Mesh Sub-header Format

Syntax	Size	Notes
Mesh sub-header() {		
TxNbrID	8 bits	
RxNbrID	8 bits	
FSMID	3 bits	
Priority	2 bits	
Reliability	1 bits	
Importance	2 bits	
}		

Table X2 – Mesh Sub-header Fields

Name	Length (bits)	Description
TxNbrID	8	The ID assigned by the transmitter node to the receiver node
RxNbrID	8	The ID assigned by the receiver node to the transmitter node
FSMID	3	Fragmentation state machine ID
Priority	2	Priority field indicates message class
Reliability	1	Message reliability information carried with message from ingress to egress. 0 = no ARQ 1 = ARQ used
Importance	2	Indicates drop precedence

Comment 10 related text and tables:

SchedulingEpochPeriod
Number of frames the reported schedule shall be valid defined as follows:
$$\text{ValidFrames} = 2^{\text{SchedulingEpochPeriod}}$$
SchedulingConfigPeriod
The frequency that this MSH-CSCF message is distributed.
CentSchedXmtsPerFrame
Number of MSH-CSCH or MSH-CRQS transmit opportunities per frame
SchedConfigXmtsPerFrame
Number of MSH-CSCF transmit opportunities per frame
NumberOfChannels
Number of channels available
NumOfNodes
Number of nodes in scheduling tree

Each entry of the scheduling tree shall include all of the following parameters:

NodeID
: Unique node identifier assigned to the node

NumOfChildren
: Number of child nodes for the node

ChildIndex
: Index of the child node

Table X3 – MSH-CSCF Message Format.

Syntax	Size	Notes
MSH-CSCH_Message_Format() {		
Generic_MAC_Header()	48 bits	
Management Message Type = 42	8 bits	
SchedulingEpochPeriod	8 bits	
SchedulingConfigPeriod	8 bits	
CentSchedXmtsPerFrame	3 bits	
SchedConfigXmtsPerFrame	2 bits	
NumberOfChannels	3 bits	
NumOfNodes	8 bits	
for (i=0; i< NumOfNodes; ++i) {		
NodeID	16 bits	Node index for this node is thus i
NumOfChildren	8 bits	
for (j=0; j< NumOfChildren; ++j) {		
ChildIndex	8 bits	Index of jth child node
}		
}		
}		

Comment 14 related new sub-clause:

6.2.7.6.4.5.1 Scheduling next MSH-NCFG transmission

During the current **Xmt Time** of a node (i.e., the time slot when a node transmits its MSH-NCFG packet), the node uses the following procedure to determine its **Next Xmt Time**:

- Order its physical neighbor table by the **Next Xmt Time**.
- For each entry of the neighbor table, add the node's **Next Xmt Time** to the node's **Xmt Holdoff Time** to arrive at the node's **Earliest Subsequent Xmt Time**.
- Set **TempXmtTime** equal to the node's advertised **Xmt Holdoff Time** added to the current **Xmt Time**.
- Set success equal to false
- **While** success equals false **do**:
  - **If TempXmtTime** equals the **Next Xmt Time** of any node in the Physical Neighbor List **Then**
    - Set **TempXmtTime** equal to next MSH-NCFG opportunity.
  - **Else Do**:

- Determine the eligible competing nodes, which is the set of all nodes in the physical neighbor list with an **Earliest Subsequent Xmt Time** equal to or smaller than **TempXmtTime**.
- Hold a *Mesh Election* among this set of eligible competing nodes and the local node using **TempXmtTime** and the list of the MAC addresses of all eligible competing nodes as the input:
  MeshElection (TempXmtTime, MyMacAdr, CompetingMacAdrsList[])
- **If** this node does not win *Mesh election* **Then**
  - Set **TempXmtTime** equal to next MSH-NCFG opportunity.
- **Else Do**
  - Set success equal to true
  - Set the node's **Next Xmt Time** equal to **TempXmtTime**.

The *Mesh Election* procedure determines whether the local node is the winner for a specific **TempXmtTime** among all the competing nodes. It returns TRUE if the local node wins or FALSE otherwise. The algorithm works as follows:

```
boolean MeshElection ( uint32 XmtTime, uint32 MyMacAdr, uint32 MacAdrList[] )
{
   uint32 nbr_smear_val, smear_val1, smear_val2;

   smear_val1 = inline_smear( MyMacAdr ^ XmtTime );
   smear_val2 = inline_smear( MyMacAdr + XmtTime );

   For each mac address nbrsMacAdr in MacAdrList Do
   {
         nbr_smear_val = inline_smear( nbrsMacAdr ^ XmtTime );

         if( nbr_smear_val > smear_val1 )
         {
            return FALSE;  // This node looses.
         }

         else if( nbr_smear_val == smear_val1 )
         {
            // 1st tie-breaker.

            nbr_smear_val = inline_smear( nbrsMacAdr + XmtTime );

            if( nbr_smear_val > smear_val2 )
            {
               return FALSE;  // This looses.
            }

            else if( nbr_smear_val == smear_val2 )
            {
               // If we still collide at this point
               // Break the tie based on MacAdr

               if ( ( XmtTime is even && ( nbrsMacAdr > MyMacAdr ) ) ||
                    ( XmtTime is odd && ( nbrsMacAdr < MyMacAdr ) ) )
               {
                  return FALSE;  // This node looses.
               }
```

```
            }
        }
    }

            // This node won over this competing node

    } // End for all competing nodes

    // This node is winner, it won over all competing nodes.
    return TRUE;
}

// Convert a uniform 32-bit value to an uncorrelated uniform
// 32-bit hash value, uses mixing.

uint32 inline_smear( uint32 val )
{
    val  += (val << 12);
    val  ^= (val >> 22);
    val  += (val << 4);
    val  ^= (val >> 9);
    val  += (val << 10);
    val  ^= (val >> 2);
    val  += (val << 7);
    val  ^= (val >> 12);

    return( val );
}
```

Comment 15 related new sub-clause:

6.2.7.6.4.5.2 Scheduling MSH-NENT messages

NetEntry scheduling protocol described in this section provides the upper layer protocol an unreliable mechanism to access the NetEntry slot(s), which are the very first slot(s) in each super-frame, so that new nodes, which are not yet fully-functional members of the network, can communicate with the fully-functional members of the network.

In the NetEntry slots new nodes shall transmit MSH-NENT messages using one of two methods:

1. In a random, contention-based fashion in a free network entry transmission slot immediately following a MSH-NCFG transmission by a proposed sponsor node, or
2. In a network entry transmission slot in which the new node is polled by its sponsor.

To support the above network entry transmission slot access scheme, the MSH-NCFG packets, transmitted by the normal network nodes, include a "**NetEntry Address**" field, set to one of the following two values:

1. 0x000000000000 – indicating that the next NetEntry transmission slot is free for contention-based access in this node's neighborhood,
2. <MAC Address of New Node> - indicating that the transmitting node is serving as the "sponsor" for the identified new node, and that the new node is polled by the sponsor to transmit in the next NetEntry transmission slot.

A sponsor node is a normal fully-functional member of the network that is selected to communicate with a new node. In order to access the NetEntry transmission slot, MSH-NENT messages should include the address of a target “sponsor” node, which can then decide whether to advertise the new node’s MAC address in its subsequent MSH-NCFG message(s).

When the sponsor expects to receive a MSH-NENT message from the new node, it advertises the new node’s MAC address in its next MSH-NCFG message to allow the new node to send a MSH-NENT message in the next NetEntry slot following the reception of the MSH-NCFG message.

A new node uses the algorithm specified by the following C-like pseudocode to access NetEntry transmission slots:

```
/* Variable Definitions */
Pkt * MSH-NENT_MsgQ = NULL;      // MSH-NENT Message queue
uint SponsorsState = UNAVAILABLE;  // SponsorsState and OthersState record the NetEntry
uint OthersState   = BUSY;
                                   // Address in the MSH-NCFG packet form the sponsor
                                   // or other nodes in the previous supperframe, which
                                   // can be used to determine the availability of the
                                   // NetEntry slot in the current supperframe.
                                   // SponsorsState can be UNAVAILABLE, AVAILABLE and POLLING.
                                   //  OthersState can be AVAILABLE and BUSY.
uint OthersMaxMacAdr = 0xffffffff;
uint OthersMinMacAdr = 0x00000000;

void RecvOutgoingMSH-NENT_Msg (Pkt *MSH-NENT_Msg)
{
        MSH-NENT_MsgQ->enqueue (MSH-NENT_Msg);
}

void RecvIncomingMSH-NCFG_Msg (Pkt * MSH-NCFG_Msg)
{
        if (MSH-NCFG_Msg->sourceMacAdr == sponsorsMacAdr)
        {
                switch (MSH-NCFG_Msg->NetEntryAddress)
                {
                        case 0x000000000000:        SponsorsState = AVAILABLE;
                                                    break;

                        case myMacAdr:      SponsorsState = POLLING;
                                            break;

                        default:        break;
                }
        }
        else
        {
                switch (MSH-NCFG_Msg->NetEntryAddress)
                {
                        case 0x000000000000:        break;

                        default:        OthersState = BUSY;
                                        if (OthersMaxMacAdr < MSH-NCFG_Msg->NetEntryAddress)
                                                OtherMaxMacAdr = MSH-NCFG_Msg->NetEntryAddress;
                                        if (OthersMinMacAdr > MSH-NCFG_Msg->NetEntryAddress)
                                                OtherMinMacAdr = MSH-NCFG_Msg->NetEntryAddress;
                }
        }
```

```
}

void SuperFrameBoundary ()
{
	boolean xmt = FALSE;

	if (MSH-NENT_MsgQ->qLength())
	{
		if (SponsorsState == AVAILABLE)
		{
			if (OthersState != BUSY)
			{
				xmt = TRUE;
			}
		}
		else if (SponsorsState == POLLING)
		{
			if (OthersState != BUSY)
			{
				xmt = TRUE;
			}
			else
			{
				if (((mayMacAdr > OthersMaxMacAdr) && (even supperframe)) ||
				   ((mayMacAdr < OthersMinMacAdr) && (odd supperframe)))
				{
					xmt = TRUE;
				}
			}
		}
	}

	if (xmt)
	{
		Pkt* MSH-NENT_Msg = MSH-NENT_MsgQ->getHead();
		MSH-NENT_MsgQ->dequeue(MSH-NENT_Msg);
		SendOutPkt (MSH-NENT_Msg, nextNetEntryslot);
	}

	SponsorsState = UNAVAILABLE;
	OthersState   = AVAILABLE;

	OthersMaxMacAdr = 0x000000000000;
	OthersMinMacAdr = 0xffffffffffff;
}
```