| | |
|---|---|
| Project | **IEEE 802.16 Broadband Wireless Access Working Group <http://ieee802.org/16>** |
| Title | **An ARQ proposal for consideration by TG3/4 MAC group** |
| Date Submitted | **2001-08-28** |
| Source(s) | Yigal Leiba<br>Itzik Kitroser<br>Runcom Technologies LTD.<br>2 Ahoma St. 75655, Rishon Lezion<br>Israel<br><br>Voice: 972-3-9528440<br>Fax: 972-3-9528805<br>yigall@runcom.co.il<br><br>itzikk@runcom.co.il |
| Re: | Call for comments, document IEEE 802.16ab-01/06r1 |
| Abstract | This ARQ proposal attempts to incorporate elements from IEEE802.16abc-01/01 and discussions, comments and contributions that took place within the ARQ Ad-Hoc group. The document attempts to cover the majority of issues related to ARQ incorporation in the MAC. |
| Purpose | For consideration and improvement by the 802.16 TG3/4 ARQ Ad Hoc Group. |
| Notice |  |
| Release |  |
| Patent Policy and Procedures |  |

# An ARQ proposal for consideration by TG3/4 ARQ Ad-Hoc group

*Yigal Leiba*
*Itzik Kitroser*
Runcom Technologies LTD.

## 1 General

The ARQ mechanism is part of the MAC layer. Its implementation is mandatory, yet it's use is optional, on a per-connection basis. The requirement to use ARQ should be specified upon connection creation, but the decision on which connections ARQ should be used is outside the scope of the air interface specification. When a connection is designated to support ARQ, transmission of non-ARQ traffic on the connection is not allowed. The scope of a specific instance of the ARQ state machines, parameters and messages is limited to one unidirectional unicast MAC connection between the BST and an SS. The feedback information required for the ARQ algorithm is sent as a MAC management message on the appropriate basic management connection between that SS and the BST (i.e. it cannot be fragmented), or as piggybacked information on some existing connection.

## 2 Block numbering concept

An ARQ block is a uniquely identifiable entity on which the ARQ algorithm operates. Each ARQ block is identified by an *ARQ block number*, which is assigned to it by the MAC. ARQ block numbers are assigned in increasing order, modulo **ARQ_MAX_IDX**. When the MAC decides to transmit a certain MSDU for the first time, it assigns it block numbers starting from the current block index, and according to the **ARQ_BLOCK_SIZE** parameter, that will determine how many ARQ blocks are contained in the MSDU. Note that the ARQ numbering is merely a numbering scheme that identifies both the MSDU transmission order, and the order of the ARQ blocks comprising each MSDU. Note the ARQ block numbering implies nothing on the order and size of MPDU transmission.

## 3 Packing and fragmentation interoperation with ARQ

### *3.1 Motivation*

As the packing and fragmentation mechanism purpose is to allow the MAC to efficiently use arbitrary bandwidth assignments. The BW assignment for any connections while not completely arbitrary is limited by the granularity imposed by the PHY, namely relatively large FEC block granularity. The current packing and fragmentation mechanism operates on a byte boundary, i.e. it can fully use any BW assignment to the last byte. The ARQ scheme described here is specifically designed for minimal interference with the existing packing and fragmentation mechanism, and preserves its granularity. The proposed ARQ scheme does pose a restriction that an MSDU fragment should not be smaller than the size of an ARQ block.

### *3.2 Connection fragmentation state*

The current 802.16 MAC creates an MPDU from a single MSDU, a fragment of an MSDU, or packing of several MSDUs or fragments of MSDUs. Currently, the range of states introduced by the fragmentation and packing mechanisms is limited by the fact that the connection on which the MSDU is transported is only allowed to be in one fragmentation state at any given instance. When ARQ, whose operation is based on re-transmission of erred data, is introduced into the current 802.16 MAC, the concept of a connection being in a single fragmentation state no longer holds. Instead, once a MSDU is fragmented, it remains fragmented until all

its pieces (i.e. ARQ blocks) have arrived at the receiver. The ARQ blocks should carry enough information in them so the receiver can place them correctly without the assumption of a single fragmentation state.

### *3.3 Block numbering operation*

To demonstrate how the block numbering works we might look at the allowed formats of MPDUs, and how ARQ block numbering would work in each. ARQ block numbering is done in the MSDU level, and ARQ blocks may be split between different MPDUs. As the ARQ algorithm in the receiver has enough information in each ARQ block to place it correctly regardless of the fragmentation, the split will at most cause the yet un-transmitted portion of a block to be transmitted in vain. If this situation is to be avoided, the MAC may choose to always fragment on an ARQ block boundary. Refer to section 7.1 for the format of the ARQ sub-headers.

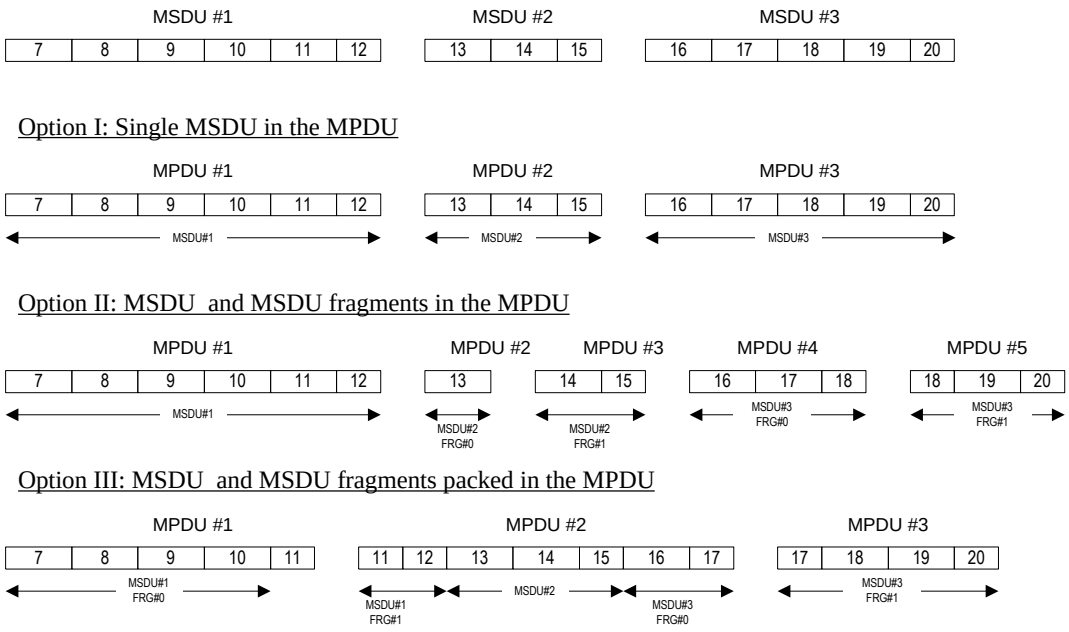


Figure 1: Packing Fragmentation and ARQ block numbering

## 4 ARQ state machine

The ARQ scheme is one of type selective repeat, and it uses an independent state machine on each ARQ-enabled connection. The ARQ scheme utilizes the existing MPDU CRC-32 checksum for detecting erred MPDUs. The proposed ARQ scheme is specifically designed to be simple and include minimal explicit messaging between transmitter and receiver.

### *4.1 Transmitter state machine*

At the transmitter the MSDUs are segmented ARQ blocks, and each block is numbered as explained in section 2 above. An ARQ block may be in one of the following four states, *not-sent*, *outstanding*, *discarded* and *not-acknowledged*. Any ARQ block begins as *not-sent*. When it is sent it becomes *outstanding* for a period of time termed **ACK_WINDOW_DURATION**. After that period of time it either is acknowledged and is *discarded*, or becomes *not-acknowledged*. An ARQ block can become *not-acknowledged* before the **ACK_WINDOW_DURATION** period expires if it is implicitly negatively acknowledged (for instance an ACK for a block having a higher sequence number has been received). An ARQ block may also change from

*not-acknowledged* to *discarded* when an ACK message for it is received or after a timeout **ARQ_BLOCK_MAX_DELAY**, but setting this timeout is considered outside the scope of the ARQ specification.
In order to simplify implementation, a parameter MAX_TX_WINDOW is defined. This parameter specifies the maximum difference between lowest and highest numbered outstanding ARQ blocks at any given time. This parameter can be varied to trade buffering requirements at the transmitter (and the receiver) for ARQ algorithm performance (throughput).
The transmitter policy is that if any *not-acknowledged* ARQ blocks exist, they should be given precedence over *not-sent* packets. ARQ blocks that are *outstanding* or *discarded* should never be transmitted.
The ARQ block state sequence is graphically shown below.

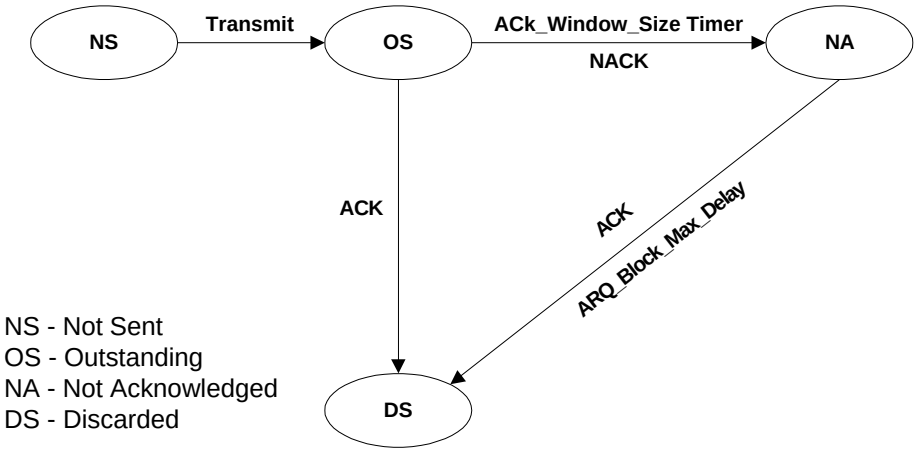


Figure 2: ARQ block states

## *4.2 Receiver state machine*

When an MPDU is received, its integrity is determined based on its CRC-32 checksum. If the MPDU is not erred, it is unpacked and de-fragmented. The receiver maintains a sliding-window defined by *minimum arq block number* state variable and the **MAX_TX_WINDOW** parameter. When an ARQ block with a number that falls in the range defined by sliding window is received, the receiver will accept it. ARQ block numbers outside the sliding window will be rejected as out of order. The receiver should identify duplicate ARQ blocks (i.e. ARQ blocks that where already received correctly) and discard them.
The sliding window is maintained such that the *minimum arq block number* variable always points to the lowest numbered ARQ block that has not been received or has been received with errors. When an ARQ block with a number corresponding to the minimum ARQ block number is received, the window is advanced (i.e. *minimum arq block number* is increased modulo **ARQ_MAX_IDX**) such that the *minimum arq block number* variable points to the next lowest numbered ARQ block that has not been received or has been received with errors.
If ARQ blocks are being received, yet the *minimum arq block number* variable has pointed to a specific ARQ block longer than **ARQ_SYNC_LOSS_DELAY**, the ARQ algorithm synchronization will be considered lost. In such a case the window will slide until an ARQ block that has been correctly received is found. The *minimum arq block number* variable will point to the next lowest numbered ARQ block that has not been received or has been received with errors. (Optionally a sync-loss handshake protocol between the transmitter and the receiver can be added if people believe it is important).

For each ARQ block accepted fully and without errors (including duplicates), an acknowledgment message is sent to the sender. Acknowledgments may be either for specific ARQ blocks (i.e. contain information on the acknowledged ARQ block numbers), or cumulative (i.e. contain the highest ARQ block number below which all ARQ blocks have been received correctly). Acknowledgments should be sent in order of the ARQ block numbers they acknowledge.
An MSDU is handed to the upper layers when all the ARQ blocks between the lowest numbered ARQ block in a fragment marked as starting-fragment and the first highest numbered ARQ block in the a fragment marked as last-fragment have been received without errors.

# 5 ARQ and scheduling interoperation

ARQ and scheduling interact in the sense that ARQ consumes part of the bandwidth for re-transmissions and for control messages (ACK). BW requests from SS supporting connections with ARQ will always include the entire overhead required for the ARQ.
To support GPC terminals a new UIUC value is defined, that identifies bandwidth the scheduler has set aside for transmission of uplink ARQ information. Here is the addition to table 108 of document IEEE P802.16/D4-2001.

| IE Name | Uplink Interval Usage Code (UIUC) | Connection ID | Description |
|---|---|---|---|
| ACK-data | 12 | unicast | Starting offset of burst for ACK data assignment. |

Table 1: ACK-data UIUC

Note that as this UIUC also describes the modulation parameters to use for sending the data, it might not be an adequate solution. A cleaner solution for GPC terminals would be to define an explicit connection for ACK data.

# 6 ARQ parameters and connection setup

Connections are set and defined either statically through the configuration file, or dynamically through the DSA/DSC class of messages. All the ARQ parameters for an ARQ supporting connection are set when the connection is set up. The transmitter and receiver windows are reset on connection setup.
This section describes the TLV fields for the ARQ algorithm, that are required for both static and dynamic connection creation methods.

## *6.1 ARQ_BLOCK_SIZE*

Fixing this number considerably simplifies the definition of related messages and bit field and makes implementation easier. The suggested fixed value is 64 bytes.

## *6.2 ARQ_MAX_IDX*

Sets the size of the group of ARQ block numbers, should be expressed as 2^N – 1 where N is an integer. The group should be large enough such that no loss of an ARQ block or an ACK will cause any ambiguity. The number would depend on the **ACK_WINDOW_DURATION**, the **ARQ_BLOCK_SIZE**, and the transmission rate. Taking as a maximum a PHY data rate of 120Mb/S with frame duration of 1mS, and allowing 16mS for the **ACK-WINDOW_DURATION**, implies we need N as high as 13 here.

## 6.3 MAX_TX_WINDOW

From the point of view of performance, it would be ideal if this parameter were set such that the transmitter is never blocked by transmission and processing delays. It is also required for correct ARQ algorithm operation that this number is less than half of the **ARQ_MAX_IDX** parameter. From the point of view of implementation complexity, this parameter eases implementation by placing a limit on the transmitter (and receiver) buffering requirements. This parameter is negotiated between the transmitter and the receiver at the connection set-up. The DSA/DSC message will contain a suggested value for this parameter The DSA-RSP/DSC-RSP message will contain an acknowledgement of this parameter, or a different suggested value. The lower of the two values will be used.

| Type | Length | Value | Scope |
|---|---|---|---|
| [24/25].? | 2 | 1 – 4095 | DSx-REQ<br>DSx-RSP<br>Configuration file |

Table 2: MAX_TX_WINDOW TLV

## 6.4 ACK_WINDOW_DURATION

The ACK window duration should account for the transmission and processing time for the forward going message and for the scheduling, transmission and processing delays for the ACK message. The combination of all these delays is partly predictable, partly implementation dependent and partly scenario dependant (e.g. the scheduling part).
The minimal **ACK_WINDOW_DURATION** is the sum of the times durations listed below,
*Tmsg-tx-time*: The time it takes to send an ARQ block over the PHY. Depends on the channel and modulation parameters. Known by the BS at connection setup.
*Tair-delay*: The time it takes the RF wave to spread, depends on the cell size. The BS at connection setup knows the maximum value.
*Treceiver-process*: Implementation dependent receiver processing delay. This is a capability of the receiver.
*Tack-scheduling*: Scenario or policy dependant scheduling delay for the ACK message in the receiver. This value could be set by the BS (for a GPC terminal) or by the SS (for a GPT terminal). The longer value should prevail.
*Tack-tx-time*: The time it takes to send an ACK message over the PHY. Depends on the format of the ACK message as well as on channel and modulation parameters. Known by the BS at connection setup.
*Tair-delay*: The time it takes the RF wave to spread, depends on the cell size. The BS at connection setup knows the maximum value.
*Ttransmitter-process*: Implementation dependent transmitter processing delay of the ACK message. This is a capability of the transmitter.
From the above analysis we may conclude that there are three types of parameters here, 1. Parameters set by the BS (namely *Tmsg-tx-time, Tair-delay, Tack-tx-time*). The DSA/DSC or DSA-RSP/DSC-RSP message will contain the values for these parameters, set by the BS.
2. Parameters negotiated between the BS and the SS (namely *Tack-scheduling*). The DSA/DSC message will contain a suggested value for this parameter The DSA-RSP/DSC-RSP message will contain an acknowledgement of this parameter, or a different suggested value. The larger of the two values will be used.
3. Parameters that are capabilities (namely *Treceiver-process, Ttransmitter-process*). The DSA/DSC or DSA-RSP/DSC-RSP message will contain the values for these parameters where the receiver and transmitter each declare its capability.
When the DSA/DSC handshake is over each party should know all the parameters listed above and each can calculate the value for the **ACK_WINDOW_DURATION** as their sum.

The table below lists the relevant TLVs.

| Name | Type | Length | Value | Scope |
|---|---|---|---|---|
| *Tmsg-tx-time* | [24/25].?.? | 2 | 0 – 4095<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |
| *Tair-delay* | [24/25].?.? | 1 | 0 – 255<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |
| *Treceiver-process* | [24/25].?.? | 2 | 0 – 65535<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |
| *Tack-scheduling* | [24/25].?.? | 2 | 0 – 65535<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |
| *Tack-tx-time* | [24/25].?.? | 2 | 0 – 4095<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |
| *Ttransmitter-process* | [24/25].?.? | 2 | 0 – 65535<br>(In microseconds) | DSx-REQ<br>DSx-RSP<br>Configuration file |

Table 3: ACK_WINDOW_DURATION releated TLVs

## *6.5 ARQ_BLOCK_MAX_DELAY*

This parameter is useful for cases where delay is bounded, yet ARQ can be helpful (e.g. VOIP). It allows limiting the delay that ARQ introduces and guarantees that transmission of ARQ blocks that are no longer useful to the upper layers is avoided.
The BS should set this parameter. The DSA/DSC or DSA-RSP/DSC-RSP message will contain the value for this parameter as set by the BS.

| Type | Length | Value | Scope |
|---|---|---|---|
| [24/25].? | 2 | 1 – 4095 | DSx-REQ<br>DSx-RSP<br>Configuration file |

Table 4: ARQ_BLOCK_MAX_DELAY TLV

## *6.6 ARQ_SYNC_LOSS_DELAY*

This parameter guarantees that the ARQ algorithm is does not get stuck indefinitely. The only limitation on this parameter is that it should be long enough such that a reasonable number of retransmissions are allowed before this timeout occurs and data is lost in the corrective action taken.
The BS should set this parameter. The DSA/DSC or DSA-RSP/DSC-RSP message will contain the value for this parameter as set by the BS.

| Type | Length | Value | Scope |
|---|---|---|---|
| [24/25].? | 2 | 1 – 4095 | DSx-REQ<br>DSx-RSP<br>Configuration file |

Table 5: ARQ_SYNC_LOSS_DELAY TLV

# 7 Formats of ARQ related MAC message

## *7.1 ARQ sub-header*

The information conveyed by the ARQ sub-header is a BSN, which stands for Block Sequential Number referencing the block number of the first ARQ block in the MPDU. As the size of the ARQ block is known, and it is possible to detect a partial ARQ block (i.e. an ARQ block that was last in the MSDU and therefore its size is smaller than **ARQ_BLOCK_SIZE**), this information should suffice to extract all ARQ block numbers in the MPDU.
When an ARQ sub-header is present with other sub-headers, it is always inserted between the generic MAC header and the MPDU payload after the grant-management and packing or fragmentation sub-headers.

## *7.2 ARQ feedback sub-header*

This sub-header enables piggybacking of ARQ feedback information elements. When ARQ-feedback sub-header is present with other sub-headers, it is always the last to be inserted between the generic MAC header and the MPDU payload. Refer to section 7.4.2 for details.

## *7.3 Generic MAC header update*

The Generic MAC header *TYPE* field needs to be updated to indicate the presence of the ARQ sub-header. Here are the updated table that should replace table 4 and table 5 of document IEEE P802.16/D4-2001.

| Type | Description |
|---|---|
| 0x00 | No sub-headers present |
| 0x01 | Reserved |
| 0x02 | Packing sub-header present |
| 0x03 | Reserved |
| 0x04 | Fragmentation sub-header present |
| 0x05 | Reserved |
| 0x06 | Reserved |
| 0x07 | Reserved |
| 0x08 | ARQ sub-header present |
| 0x09 | Reserved |
| 0x0A | ARQ and packing sub-headers present |
| 0x0B | Reserved |
| 0x0C | ARQ and fragmentation sub-headers present |
| 0x0D – 0xF | Reserved |
| 0x10 | ARQ-feedback sub-header present |
| 0x11 | Reserved |
| 0x12 | ARQ-feedback and packing sub-header present |
| 0x13 | Reserved |
| 0x14 | ARQ-feedback and fragmentation sub-header present |
| 0x15 | Reserved |
| 0x16 | Reserved |
| 0x17 | Reserved |
| 0x18 | ARQ-feedback and ARQ sub-header present |
| 0x19 | Reserved |
| 0x1A | ARQ-feedback and ARQ and packing sub-headers present |
| 0x1B | Reserved |
| 0x1C | ARQ-feedback and ARQ and fragmentation sub-headers present |
| 0x1D – 0x3F | Reserved |

Table 6: Downlink type encoding

| Type | Description |
|---|---|
| 0x00 | No sub-headers present |
| 0x01 | Grant management sub-header present |
| 0x02 | Packing sub-header present |
| 0x03 | Grant management and packing sub-headers present |
| 0x04 | Fragmentation sub-header present |
| 0x05 | Grant management and fragmentation sub-headers present |
| 0x06 | Reserved |
| 0x07 | Reserved |
| 0x08 | ARQ sub-header present |
| 0x09 | ARQ and grant management sub-headers present |
| 0x0A | ARQ and packing sub-headers present |
| 0x0B | ARQ and packing and grant management sub-headers present |
| 0x0C | ARQ and fragmentation sub-headers present |
| 0x0D | ARQ and fragmentation and grant management sub-headers present |
| 0x0E – 0x0F | Reserved |
| 0x10 | ARQ-feedback sub-header present |
| 0x11 | ARQ-feedback and grant management sub-headers present |
| 0x12 | ARQ-feedback and packing sub-headers present |
| 0x13 | ARQ-feedback and grant management and packing sub-headers present |
| 0x14 | ARQ-feedback and fragmentation sub-headers present |
| 0x15 | ARQ-feedback and grant management and fragmentation sub-headers present |
| 0x16 | Reserved |
| 0x17 | Reserved |
| 0x18 | ARQ-feedback and ARQ sub-headers present |
| 0x19 | ARQ-feedback and ARQ and grant management sub-headers present |
| 0x1A | ARQ-feedback and ARQ and packing sub-headers present |
| 0x1B | ARQ-feedback and ARQ and packing and grant management sub-headers present |
| 0x1C | ARQ-feedback and ARQ and fragmentation sub-headers present |
| 0x1D | ARQ-feedback and ARQ and fragmentation and grant management sub-headers present |
| 0x1E – 0x3F | Reserved |

Table 7: Uplink type encoding

### 7.3.1 ARQ with no fragmentation and no packing

In this case each MPDU will contain a single MSDU. Knowledge of the BSN, the length of the MSDU (conveyed in the MAC header) and the **ARQ_BLOCK_SIZE** parameter enables the calculation of the range of ARQ blocks contained in the message. The ARQ sub-header position and its contents are shown below.

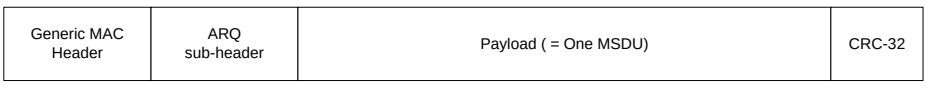


Figure 3: ARQ sub-header with no fragmentation and no packing

| Syntax | Size | Notes |
|---|---|---|
| ARQ_Sub_Header_Format() { | | |
| **Reserved** | 3 bits | |
| **BSN** | 13 bits | Block sequential number for the first ARQ block in the MSDU |
| } | | |

Table 8: ARQ sub-header format with no fragmentation and no packing

### 7.3.2 ARQ with fragmentation only

In this case each MPDU will contain a single fragment of a MSDU. Fragmentation is NOT guaranteed to be on an ARQ block boundary, and therefore either the first ARQ block or the last ARQ block (or both) could be partial blocks. Note that part of the information carried in the fragmentation sub-header is no longer very useful. We still require the FC field to tell us if this is the first, last or a middle fragment, but the FSN field is redundant, as we must use the ARQ block numbers for correct fragment placement. We will reuse the FSN bits carry information about the size of the first ARQ block (that may be partial) in the fragment. Note that a partial first ARQ block in a fragment will always carry the last bytes of the ARQ block while a partial last ARQ block in the fragment will always carry the first bytes of the ARQ block (there is an example later).
Knowledge of the BSN of the first ARQ block, the size of the first ARQ block, the length of the MSDU (conveyed in the MAC header) and the **ARQ_BLOCK_SIZE** parameter enables the calculation of the range of ARQ blocks contained in the message. The ARQ with fragmentation sub-header position and its contents is shown below.

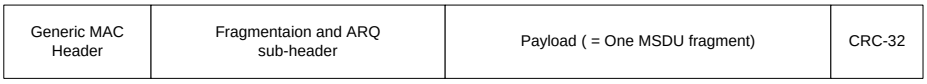


Figure 4: ARQ sub-header with fragmentation only

| Syntax | Size | Notes |
|---|---|---|
| Fragmentation_and_ARQ_Sub_Header_Format() { | | |
| **FC** | 2 bits | Fragmentation Control<br>Indicates the fragmentation state of the payload:<br>00 = no fragmentation<br>01 = last fragment<br>10 = first fragment<br>11 = continuing (middle) fragment |
| **Reserved** | 3 bits | |
| **First block length in bytes, minus one** | 6 bits | **ARQ_BLOCK_SIZE** = 64 bytes |
| **BSN** | 13 bits | Block sequential number for the first ARQ block in the MSDU fragment |
| } | | |

Table 9: ARQ with fragmentation sub-header format

### 7.3.3 ARQ with fragmentation and packing

In this case each MPDU may contain multiple MSDU or fragments thereof. In this case as well, fragmentation is NOT guaranteed to be on an ARQ block boundary, and therefore either the first ARQ block or the last ARQ block (or both) could be partial blocks. Here as well, part of the information carried in the fragmentation sub-header is no longer very useful. We still require the FC field but not the FSN field. We will reuse the FSN bits carry information about the size of the first ARQ block (that may be partial) in the fragment. Note that a partial first ARQ block in a fragment will always carry the last bytes of the ARQ block while a partial last ARQ block in the fragment will always carry the first bytes of the ARQ block (see example below).
Each of the packed MSDU or MSDU fragments requires its own ARQ sub-header, as some of them may be transmissions while other are re-transmissions. Knowledge of the BSN of the first ARQ block, the size of the first ARQ block, the length of the each MSDU fragment (conveyed in the packing sub-header) and the **ARQ_BLOCK_SIZE** parameter enables the calculation of the range of ARQ blocks contained in each part of the packed message. The ARQ with packing sub-headers position and the contents of an ARQ with packing and fragmentation sub-header is shown below.

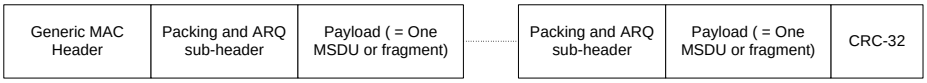


Figure 5: ARQ sub-header with fragmentation and packing

| Syntax | Size | Notes |
|---|---|---|
| Packing_and_ARQ_Sub_Header_Format() { | | |
| **FC** | 2 bits | Fragmentation Control<br>Indicates the fragmentation state of the payload:<br>00 = no fragmentation<br>01 = last fragment<br>10 = first fragment<br>11 = continuing (middle) fragment |
| **Length** | 11 bits | The length in bytes of the MSDU or MSDU fragment, including the four-byte Packing_and_ARQ sub-header |
| **First block length in bytes, minus one** | 6 bits | **ARQ_BLOCK_SIZE** = 64 bytes |
| **BSN** | 13 bits | Block sequential number for the first ARQ block in the MSDU or MSDU fragment |
| } | | |

Table 10: ARQ with fragmentation and packing sub-header format

As this case is the most complicated, an example is given here to show how information on the ARQ block numbers is extracted from the data in the MPDU in a dynamic situation.

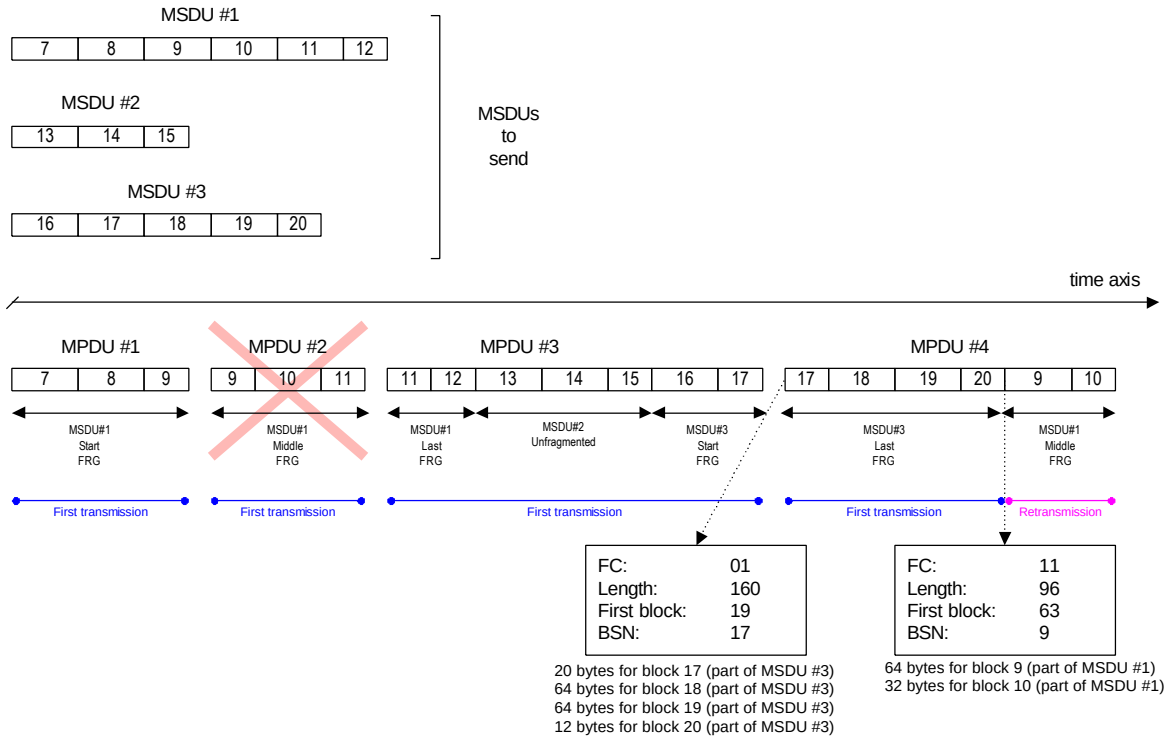


Figure 6: ARQ with packing and fragmentation dynamic example

## *7.4 ACK messages*

In order to function the ARQ algorithm in the transmitter needs feedback from in the receiver in the form acknowledgment messages for the ARQ blocks that have been received without errors. The feedback information is sent on the appropriate basic management connection between that SS and the BST (there is one such connection in each direction). Note that MAC management messages sent on the basic management connection should not be packed or fragmented.

### 7.4.1 Stand-alone ACK message

This ACK message may take the format of a stand-alone MAC message. It can signal a cumulative ACK or several selective ACKs, and it can also hint on not acknowledged ARQ block numbers (NACKs). The format is shown below.

| Syntax | Size | Notes |
|---|---|---|
| ACK_Message_Format() { | | |
| **Management Message Type = ?** | 8 bits | |
| for (*i* =1; *I* <*n* ; *i* ++) { | | Repeat as many times as required |
| ARQ_feedback_IE () | 16 bits | The connection ID being referenced |
| } | | |
| } | | |

Table 11: ACK message

| Syntax | Size | Notes |
|---|---|---|
| ARQ_feedback_IE () { | | |
| **Last_flag** | 1 bit | 0 = More ARQ feedback IE in the list<br>1 = Last ARQ feedback IE in the list |
| **Cum_Sel_flag** | 1 bit | 0 = Selective ACK entry<br>1 = Cumulative ACK entry |
| **BM_flag** | 1 bit | 0 = Bit-map field does not exist<br>1 = Bit-map field exists |
| If (Cum_Sel_flag == 0) { | | |
| **BSN** | 13 bits | Block sequential number for the acknowledged ARQ block |
| } | | |
| else { | | |
| **BSN** | 13 bits | Block sequential number for the ARQ block below which all blocks are acknowledged |
| } | | |
| If (BM_flag == 1) { | | |
| **ACK MAP** | 16 bits | Each bit set to one means the corresponding ARQ block has been received without errors. The LSB corresponds to the ARQ block whose number is the BSN above, an the MSB relates to the ARQ block whose number is (BSN + 15) modulo 2^13 |
| } | | |
| **CID** | 16 bits | The ID of the connection being referenced |
| } | | |

Table 12: ARQ feedback IE definition

### 7.4.2 Piggyback ACK

The ARQ feedback can be piggybacked on any connection taking the format of a sub-header.