# An Automatic Re- Transmission Scheme (ARQ) for High Speed Point to Multipoint Networks

**Subir Varma**

*Aperto Networks*
*1637 South Main Street*
*Milpitas, CA 95035*

## 1.0 Current State of the Art

ARQ is a Layer 2 protocol which is commonly used in point-to-point links, to recover from link errors. It involves the re-transmission of errored packets by the link sender, in response to ACK or NACK indications from the link receiver. ARQ enables a link with high error rate to recover from those errors, and thus be able to support higher layer protocols whose functioning requires a reliable link.

In the area of point-to-multipoint networks, ARQ has been proposed as part of the Layer 2 protocol used for transmitting data over narrowband, circuit switched, voice oriented systems, such as the TDMA and CDMA cellular data protocols. However, to our knowledge, there are no current products that provide advanced ARQ services in a broadband, packet oriented, point-to-multipoint network. The proposed invention provides a way in which this can be accomplished, and in the process solves several significant problems that stand in the way.

Two types of ARQ are decribed in this disclosure:

- Downstream ARQ: Data is transmitted downstream, while ACKs are transmitted upstream.
- Upstream ARQ: Data is transmitted upstream, while ACKs are transmitted downstream.

## 2.0 Shortcomings of Current Art

## 3.0 How the Proposed Invention Extends the Current Art

List of issues relating to the use of ARQ in a packet oriented broadband access environment, and how the proposed system deals with them:

- Change in link parameters during re-transmissions (Applies to both Upstream and Downstream ARQ)

  Several Broadband access systems allow link parameters to adapt to current channel conditions. If the parameters change while packets belonging to a flow are undergoing re-transmissions, then current ARQ techniques no longer apply. The proposed system allows the link parameters to change during re-transmissions. It does so by using a byte, rather than packet based, accounting scheme.

- Downstream ARQ: Automatic allocation of Upstream ACK Slots

  Unlike point-to-point systems, the transmission of upstream ACKs in a point-to-multi-point system requires some special considerations. Since the upstream channel is a shares resource, all upstream transmissions have to content for BW before they are allocated tranmission slots. Hoever, if upstream ACKs are also forced to go through this process, then it can significantly impact performance by delaying the reception of ACKs. The proposed system solves this problem by automatically allocating BW for upstream ACK transmissions whenever is are un-acknowledged data in the transmitter.

- Upstream ARQ: Automatic allocation of Upstream Data Slots for Re-transmissions

  If Upstream ARQ re-transmission control is placed in the transmitter, then this will require the CPE to contend for BW for each upstream re-transmission. In order to improve performance, the proposed design places upstream re-transmission control in the receiver, which is in the BSC. As a result, the BSC can allocate upstream slots for re-transmissions automatically, without being prompted by the CPE to do so.

- Integration of ARQ within the framework of a TDMA/TDD based MAC

  In a TDMA/TDD based system, allocations for data transmission are made by the system scheduler in advance of the actual transmission itself. If a packet is dropped due to errors, then the ARQ protocol requires that all subsequent packets also be dropped, until the first errored packet is re-transmitted succesfully. The TDMA/TDD scheduling and the ARQ interact together in a way, such that if no special measures are taken, then the system does not function correctly. The proposed design solves this problem, by 1) Downstream: The transmitter sets a threshold everytime it receives a NACK, and then it ignores all ACKs or NACKs received before the threshold expires. 2) Upstream: The receiver sets a threshold everytime it receives a bad WPDU, and then it ignores all WPDUs received before the threshold expires.

  The values of the thresholds in either direction are set acoording to the parameters used for the TDD/TDMA frame sizes. As a result of this feature, the system is able to recover from errors and continue correct operation.

# Pseudocode

# 4.0 Downstream ARQ (BSC Tx, CPE Rx)

## 4.1 Parameters (Control PDU Handler)

```
ARQWindowSize;      // Size of the ARQ window. Set to 2^(n-1) bytes, where n is the
                    //   number of bits in the Sequence Number field
maxAcksLost;        // Maximum number retries for the upstream ACKs, after which
                    //   the CPE is re-ranged
maxReqRetries;      // Maximum number of retries for a REQ packet.
```

## 4.2 BSC Tx (reqWin, scWin, curWin, ackWin)

### 4.2.1 Initialize (Control PDU Handler)

```
reqWinOff = 0;    // Sequence number of next byte to be queued
scWinOff = 0;     // Sequence number of next byte to be transmitted by BSC
curWinOff = 0;    // Sequence number of next byte the CPE expects
ackWinOff = 0;    // Sequence Number of next byte awaiting acknowledgment

// Allocate empty SIDQ_EL and initialize pointers
newSidQEl = AllocateSidQEl();
newSidQEl->EOL = TRUE;
writeElPtr = ackElPtr = curElPtr = scElPtr = newSidQEl;
ackPtr = curPtr = scPtr = 0;

retryCnt = 0;         // Used to decide when to drop a packet
NumAcksLost  = 0;     // Used for link adaptation
```

### 4.2.2 PDU Arrival (Classifier, Policer)

```
// Classify the WPDU
find sidQ (PDU);                                  // Classifier

// Enqueue the WPDU on the overflow section of the sidQ
newSidQEl = AllocateSidQEl();                     // Policer
newSidQEl->EOL = TRUE;                            // Policer
writeElPtr->next = newSidQEl;                     // Policer
writeElPtr->length = PDU.length;                  // Policer
writeElptr->txMsgPtr = PDU.txMsgPtr;              // Policer
writeElPtr->pktPtr = PDU.packet;                  // Policer
writeElPtr = newSidQEl;                           // Policer

// Traffic shaping may be done before the packet is moved out of the overflow section.
// These updates must be done last to avoid timing problems with USG.
reqWinOff = reqWinOff + PDU.size;                 // Policer
writeElPtr->EOL = FALSE;                          // Policer
```

### 4.2.3 MAP Construction (Scheduler)

```
while (space left for data in downstream TDD frame)
{
   sidQCtrl = SID that Scheduler selects;
   bytesInQueueToSchedule = reqWinOff - scWinOff;

   // Always try to schedule bytes for SIDs without ARQ.
   // For SIDs with ARQ, we need to make sure that we have not
```

```
   // exhausted our window before we try to schedule some bytes.
   if ( (sidQCtrl.sidCfgBits.arq = FALSE) OR
        ((scWinOff + bytesScheduled - ackWinOff) < ARQWindowSize) )
   {
      DATA_GRANT_IE.winOff = scWinOff;
      DATA_GRANT_IE.payloadSize = bytesScheduled;   // Includes delimiter bytes
      scWinOff = scWinOff + DATA_GRANT_IE.payloadSize;
      allocate ticks for WPDU in downstream portion of TDD Frame;
      update scElPtr and scPtr to reflect bytes scheduled;

      // Mark SID as needing ACK
      if ( (sidQCtrl.sidCfgBits.ack = TRUE) AND (!sidQCtrl.ackFlag) )
      {
          sidQCtrl.ackFlag = TRUE;
          add to list of downstream SIDs needing ACK;
      }
   }

} // while (space left)

// Schedule only one ACK per SID for a frame.
// We can schedule ACKs for SIDs without ARQ.  This is needed for link adaptation.
for each SID on list of downstream SIDs needing ACK
{
   // If there are bytes remaining to be acked, allocate space for the
   // ACK even if the current frame has no WPDUs scheduled for this SID
   if (scWinOff != ackWinOff)
   {
      Allocate ticks for ACK in the upstream portion of TDD frame;
      ACK_IE.sid = this SID;
   }
   else
   {
      delete from list of SIDs needing ACK;
      sidQCtrl.ackFlag = FALSE;
   }

} // for (each SID on list)
```

### 4.2.4 MAP Arrival (Hardware)

```
if (data grant IE)
{
    // Was a packet dropped or retransmitted?
    if ( (sidQCtrl.sidCfgBits.arq = TRUE) AND (curWinOff != DATA_GRANT_IE.winOff) )
    {
       // if possible, check (ackWinOff == DATA_GRANT_IE.winOff)
       curWinOff = ackWinOff;
       Reset the cur pointers to the ack pointers;
    }

   // Need pseudocode for HW packet fragmentation

    Build a WPDU using the curElPtr and curPtr
    WPDU.winOff = curWinOff;         // Should we use data grant ie not curWin??  JF

    curWinOff = curWinOff + DATA_GRANT_IE.payload
    Update curElPtr and curPtr to reflect bytes transmitted
}
```

### 4.2.5 WPDU Transmit (Hardware)

```
transmit built WPDU;
if (sidQCtrl.sidCfgBits.arq == FALSE)
   return any completely transmitted packet;
```

### 4.2.6 ACK Arrival (Scheduler)

```
// Calculate the number of ACKed bytes
NumAcksLost = 0;
ackByteCnt = ACK.winOff - ackWinOff;

// Only free buffers here if ARQ. Otherwise they'd have been freed right after transmit.
if (sidQCtrl.sidCfgBits.arq == TRUE)
{

   // Any bytes ACKed?
   if (ackByteCnt)
   {
      ackWinOff = ackWinOff + ackByteCnt;
      tempElPtr = ackElPtr;
      update ackPtr and ackElPtr to account for the bytes ACKed;

      if (tempElPtr != ackElPtr)
         free SIDQ_ELs between ackElPtr and tempElPtr;
      if (ACK.nakFlag clear)
         retryCnt = 0;
   }

   // Any bytes NACKed?
   if (ACK.nakFlag set)
   {
       if ((ackByteCnt == 0) && (time > threshold))
           threshold = time at which the last (partially) allocated TDD frame ends;
           retryCnt = retryCnt + 1;

       // When the retry count expires, drop only the first packet in the list.
       if (retryCnt > sidQCtrl.maxRetry)
       {
           // pktPtr points to the first byte in the packet, and ackPtr is the offset from
           // pktPtr to the next byte to ack
           dropBytes = ackElPtr->length - ackPtr;
           tempElPtr = ackElPtr;
           update ackElPtr to next packet in list;
           ackPtr = 0;
           free (tempElPtr);

           // Account for any bytes that need to be retransmitted
           reqWinOff -= dropBytes; // Scheduler asks Policer to do this and does not
                                   // schedule any more bytes for this SID until it is
                                   // done.

           inform link adaptation task that we dropped an EPDU
       }

       // We have to reschedule some bytes for retransmission
       scWinOff = ackWinOff;
       update sc pointers to ack pointers;

   } // if nakByteCnt

} // if ARQ
```

### 4.2.7 ACK Lost (Scheduler)

```
NumAcksLost = NumAcksLost + 1;
if (NumAcksLost > maxAcksLost)
   ReRange CPE;

Note: ACK may be lost if the corresponding MAP was lost. However it is not clear how a
      lost MAP event may be detected by the BSC.
Note: If a CPE cannot be ReRanged, the Link Adaptation Task needs to send a message
```

to the Control PDU Handler to flush the sidQ.

## 4.3 CPE Rx (curWin)

### 4.3.1 Initialize (Control PDU Handler)

```
// CPE S/W does not care about winOffs
curWinOff    = 0;     // Sequence number of the next WPDU to transmit/receive
cur pointers = NULL;
```

### 4.3.2 WPDU Arrival (Hardware)

```
// Never keep bad wpdus
if (crc error)
{
   Set NAK flag;
   Discard(WPDU);
   discard any packet currently being reassembled;
}

else if (no energy detected)
   Set NAK flag;

// If an out of sequence wpdu arrives and this SID has ARQ, discard the
// wpdu until we receive the next sequence number we are expecting.
else if ( (sidQCtrl.sidCfgBits.arq = TRUE) AND (WPDU.winOff != curWinOff) )
   Discard(WPDU);

// Receive the WPDU. Either it's in correct sequence, or the SID has no ARQ and
// doesn't care about the sequence.
else
{
   curWinOff = WPDU.winOff + WPDU.payloadSize;

   // Need pseudocode for HW packet reassembly

   // if a new packet arrives and we were previously assembling a packet,
   // we discard the old packet and accept the new.
   if ((WPDU.catPtr == 0) and (curPtr != 0))
   {
      Discard(Partial assembled packet);
      curPtr = 0;
      curElPtr = NULL;
   }

   // if possible, check the new packet for incorrect length, cuz if it's wrong, and
   // we don't find it here, it'll be a real bugger to track down
   if (curElPtr.length != curPtr??)
      discard packet;
}
```

### 4.3.3 ACK Transmission (Hardware)

```
// When wpdus are scheduled for SIDs with ACK, the Scheduler
// will create an IE in the same MAP or in the following MAP
// for the ACK.
if (sidQCtrl.sidCfgBits.ack == TRUE)
{
    ACK.status    = ACK or NAK;
    ACK.winOff    = curWinOff;
    ACK.linkParms = modemStatus;
    Transmit ACK;
}
```

# 5.0 Upstream ARQ (CPE Tx, BSC Rx)

## 5.1 CPE Tx (reqWin, curWin, ackWin)

### 5.1.1 Initialize (Control PDU Handler)

```
reqWinOff = 0;  // Sequence Number for the number of the next byte awaiting
                //   transmission.
curWinOff = 0;  // Sequence Number of the next byte that the CPE expects to tx. The
                //   sequence number in the MAP may be less than this, in case of
                //   re-transmissions.
ackWinOff = 0;  // Sequence Number of the next byte awaiting acknowledgment.

// Allocate empty SIDQ_EL and initialize pointers
newSidQEl = AllocateSidQEl();
newSidQEl->EOL = TRUE;
writeElPtr = ackElPtr = curElPtr = readElPtr = newSidQEl;
ackPtr = curPtr = 0;
```

### 5.1.2 PDU Arrival (Classifier, Policer)

```
// Classify the WPDU
find sidQCtrl (PDU);                            // Classifier

// Create new empty SidQEl to terminate list
newSidQEl = AllocateSidQEl();                   // Policer
newSidQEl->EOL = TRUE;                          // Policer

// Enqueue the WPDU on the overflow section of the sidQ. EOL bit should already be set.
writeElPtr->next = newSidQEl;                   // Policer
writeElPtr->length = PDU.length;                // Policer
writeElptr->txMsgPtr = PDU.txMsgPtr;            // Policer
writeElPtr->pktPtr = PDU.packet;                // Policer
writeElPtr = newSidQEl;                         // Policer

if (sidQCtrl->flushFlag not set)
{
    wait till activeFlowFifo has room;
    activeFlowFifo = PDU.sidNumber;             // Policer notifies Hw
}

// Traffic shaping may be done before the packet is moved out of the overflow section
reqWinOff = reqWinOff + PDU.size;               // Policer
writeElPtr->EOL = FALSE;                        // Policer
```

### 5.1.3 REQ Transmission (HW)

```
if (state = Idle)
   PDU arrival
   Compute Defer
   state = Deferring;

else if (state = Deferring)
   map arrives with req IE opportunity
      REQ.winOff = curWinOff;
      REQ.reqWinOff = reqWinOff;
      Tx REQ;
      state = GrantPending;

else if (state = GrantPending)
   // The BSC received our REQ packet
   map arrives with upstream data IE
```

```
      transmit WPDU;

      // Any more bytes left in SID queue?
      if (reqWin - curWin)
         state = GrantPending;

      // SID queue is empty
      else
         numReqRetries = 0;
         state = Idle;

   // Our REQ packet did not get to the BSC
   map arrives with no grant IE or grant pending IE
      numReqRetries = numReqRetries + 1;
      if (numReqRetries > maxReqRetries)
         HW writes SID num plus flush flag in fifo;
         HW does not tx anymore pdus until sw writes to ACTIVE_SID_FIFO;
         HW sets sidQCtrl->flushFlag;
         numReqRetries = 0;
         state = Idle;
      else
         state = Deferring;
```

### 5.1.4 MAP Arrival (Hardware)

```
if (MAP missing)
{
   scream at Mike for kicking it into the creek;
   try to get away from the Blair Witch;

   calculate time of next MAP;
   assume largest MAP size;
   program Broadcom to receive next MAP;
}

if (Data Grant IE)
{
   // If ARQ, don't do anything until a grant gives us the expected offset
   if ( (sidQCtrl.sidCfgBits.arq == FALSE) OR (DATA_GRANT_IE.winOff == curWinOff) )
   {
      WPDU.payloadSize = DATA_GRANT_IE.payloadSize;
      Confirm that allocated ticks are sufficient to accommodate WPDU;
      WPDU.req = reqWinOff;
      WPDU.winOff = curWinOff;
   }
}

if ((MAP ACK IE) OR (MAP NAK IE))
{
   if (sidQCtrl.sidCfgBits.arq == TRUE)                    // THIS HAS CHANGED! 5/4/00
   {
      ackByteCnt = ACK.winOff - ackWinOff;                 // THIS HAS CHANGED! 5/4/00
      ackWinOff = ACK.winOff;

      // Any bytes ACKed?
      if (ackByteCnt)
      {
         update ackElPtr to account for the ackByteCnt;
         ackPtr = 0;
      }

      if (MAP NAK IE)
      {
         reset cur pointers and winOff to ack pointers and winOff;
      }
```

```
      // Notify SW of ACK, so it can free buffers.
      write SID number and set ACK flag in the WM_TX_PKT_FIFO;

   } // if ARQ
} // if ACK or NAK IE

if (MAP FLUSH IE)
{
   // The Scheduler decided it was time to give up on the packet, so drop the
   // EOL or End Of List packet.

   write SID number and set flush flag in WM_TX_PKT_FIFO;
   set sidQCtrl->flushFlag;

   // Force data transmission on this SID to halt.  This gives us time to
   // update the reqWinOff.
   HW does not tx anymore pdus until SW writes to ACTIVE_SID_FIFO;
   go to req state Idle;
}

if (Toilet FLUSH IE)
{
    flush toilet;
    if ((you are a guy) AND (you're married))
        put toilet seat down;
}
```

### 5.1.5 Process Tx Pkt Fifo (WMAC Driver)

```
read SID number from WM_TX_PKT_FIFO;
if (ACK flag)
{
    free SIDQ_ELs from readElPtr to ackElPtr;
    readElPtr = ackElptr;
}
if (flush flag)
{
    send Flush msg to Policer;
}
```

### 5.1.6 Flush Packet (Policer)

```
// Software temporarily has write access to all sidQCtrl fields.
drop EOL PDU;
update ackElPtr to skip remainder of dropped PDU;
ackPtr = 0;
curWinOff = ackWinOff;
reqWinOff = reqWinOff - remainder of dropped PDU;
update curPtr and curElPtr to ackPtr and ackElPtr;
clear sidQCtrl->flushFlag;

// Kick off another REQ if there are any bytes still on the queue.
if (reqWinOff - curWinOff)
   write SID number to ACTIVE_SID_FIFO;
```

### 5.1.7 WPDU Transmission (Hardware)

```
extract WPDU.payloadSize bytes from position curWinOff in SID queue;
advance curElPtr and curPtr by WPDU.payloadSize bytes;
curWinOff = curWinOff + WPDU.payloadSize;
if (sidQCtrl.sidCfgBits.arq == FALSE)
   return any completely transmitted packet;
```

## 5.2 BSC Rx (reqWin, scWin, curWin)

### 5.2.1 Initialize (Control PDU Handler)

```
scWinOff = 0;     // Sequence Number of next byte to be transmitted by CPE
curWinOff = 0;    // Sequence Number of the next byte that the BSC expects
reqWinOff = 0;    // Cumulative count of number of bytes received at CPE
retryCnt = 0;     // Number of times we have sent the packet unsuccessfully.
```

### 5.2.2 REQ Arrival (Scheduler, Hardware)

```
if (sidQCtrl.sidCfgBits.arq == FALSE)
    scWinOff = REQ.winOff;              // Scheduler  * this is new *

reqWinOff = REQ.reqWinOff;              // Scheduler
```

### 5.2.3 MAP Construction (Scheduler)

```
// Clear ErrorRecovery state for each new frame
state = normal;

while (Space left in current Upstream TDD frame)
{
   sidQCtrl = SID that Scheduler selects;
   bytesInQueueToSchedule = reqWinOff - scWinOff;

   // Always try to schedule bytes for SIDs without ARQ.
   // For SIDs with ARQ, we need to make sure that we have not
   // exhausted our window before we try to schedule some bytes.
   if ( (sidQCtrl.sidCfgBits.arq == FALSE) OR
        ((scWinOff + BytesScheduled - curWinOff) < ARQWindowSize) )
   {
       Allocate ticks for WPDU in upstream portion of TDD frame;
       DATA_GRANT_IE.payloadSize = BytesScheduled;
       DATA_GRANT_IE.winOff = scWinOff;
       scWinOff = scWinOff + DATA_GRANT_IE.payloadSize;
   }
}
```

### 5.2.4 WPDU Arrival (Hardware)

```
// Discard any bad wpdus
if (CRC Error)
{
   discard(WPDU);
   discard any packet currently being reassembled;
}

// If ARQ, discard any out of sequence wpdus
else if (sidQCtrl.sidCfgBits.arq == TRUE) AND (WPDU.winOff != curWinOff) )
{
   Discard(WPDU);
   Hw writes burst status to Fifo;
   send bad or dropped status to Scheduler;
}

// Good wpdu
else
{
   curWinOff = WPDU.winOff + WPDU.payloadSize;

   // Discard packet cases -
   //   if a new packet arrives and we were previously assembling a packet
```

```
   //   if the packet arrives with an incorrect length

   if ((WPDU.catPtr == 0) and (curPtr != curElPtr.pktPtr))
   {
      Discard(Partial assembled packet);
      curPtr = 0;
      curElPtr = NULL;
   }

   // if possible, check the new packet for incorrect length
   if (curElPtr.length != (curPktPtr - curElPtr.pktPtr))
       discard packet;

   send good status to Scheduler;
}
```

### 5.2.5 WPDU Status Arrives (Scheduler)

```
if (wpdu good) AND ((sidQCtrl.sidCfgBits.arq == FALSE) OR (ackWinOff == WPDU.winOff))
{
   retryCnt = 0;
   reqWinOff = WPDU.reqWinOff;
   ackWinOff = WPDU.winOff + WPDU.length;

   // WMAC Driver needs to calculate this and send the new winOff to the Scheduler
   if (sidQCtrl.sidCfgBits.arq == FALSE)
      scWinOff = WPDU.winOff + WPDU.length;
}

else if ( ((wpdu lost) OR (wpdu bad)) AND (sidQCtrl.sidCfgBits.arq == TRUE) )
{
   // By checking the state for ErrorRecovery, this means that we just
   // reset the Scheduler's window for the first bad WPDU in the frame.
   // The state is reset to normal during upstream map construction.
   //
   // If a MAP is lost, then the wpdus will be lost.
   if ( (time > ErrorRecoveryTime) AND (retryCnt <= sidQCtrl.maxRetry) )
   {
      nakFlag = TRUE;
      ErrorRecoveryTime = Tick Count at end of last scheduled upstream frame;
      scWinOff = curWinOff;
      // Should ackWinOff = curWinOff? Verify.
      update sc pointers to cur pointers;
      retryCnt = retryCnt + 1;
   }
}
```

### 5.2.6 Flush Packet (Scheduler, Policer)

```
// When the retry count expires, drop the packet being assembled.
if (retryCnt > maxRetry)                          // Scheduler
{
   // When a packet is dropped, the CPE must make a new request.
   scWinOff = curWinOff;                           // Scheduler
   Update sc pointers to cur pointers;             // Scheduler
   retryCnt = 0;                                   // Scheduler
   flushFlag = 1;                                  // Scheduler
   Send msg to Link Adaptation Routine;            // Scheduler
   send msg to Policer with sidNum;                // Scheduler
   reqWinOff = curWinOff;
}
```

### 5.2.7 Build MAP ACK IE Types (Scheduler)

```
if (sidQCtrl.sidCfgBits.arq == TRUE)
```

```
{
   if (nakFlag)
   {
      NACK.sidNumber = sid;
      NACK.winOff    = curWinOff;
      Put NACK in MAP;
      nakflag = FALSE;
   }
   else if (flushFlag)
   {
      FLUSH.sidNumber = sid;
      FLUSH.winOff    = curWinOff;
      Put FLUSH in MAP;
      flushflag = 0;
   }
   else
   {
      ACK.sidNumber = sid;
      ACK.winOff    = ackWinOff;
      Put ACK in MAP;
   }
}
```