Analysis and Performance Evaluation of the IEEE 802.16 ARQ Mechanism

Vitaliy Tykhomyrov, Alexander Sayenko, Henrik Martikainen, and Olli Alanen

***Abstract*—The IEEE 802.16 standard defines the ARQ mechanism as a part of the MAC layer. The functioning of the ARQ mechanism depends on a number of parameters. The IEEE 802.16 specification defines them but it does not provide concrete values and solutions. This paper studies the key features and parameters of the 802.16 ARQ mechanism. In particular, we consider a choice for the ARQ feedback type, an algorithm to build block sequences, the ARQ feedback intensity, a scheduling of the ARQ feedbacks and retransmissions, the ARQ block rearrangement, ARQ transmission window and the ARQ block size. We ran simulation scenarios to study these parameters and how they impact the performance of application protocols. The simulation results reveal that the ARQ mechanism and its correct configuration play an important role in achieving reliable data transmission.**

***Index Terms*—IEEE 802.16 WiMAX, ARQ, NS-2**

I. Introduction

IEEE 802.16 is a standard for the wireless broadband access network [1] that can provide a high-speed wireless access to the Internet to home and business subscribers. It supports applications and services with diverse Quality-of-Service (QoS) requirements. The core components of a 802.16 system are a subscriber station (SS) and a base station (BS). The BS and one or more SSs can form a cell with a point-to-multipoint (PMP) structure. In this case, the BS controls the activity within a cell, resource allocations to achieve QoS and admission based on the network security mechanisms. An overview of the key 802.16 features is given in [6].

The automatic repeat request (ARQ) is the mechanism by which a receiving end of a connection can request the retransmission of MAC protocol data unit (PDU), generally as a result of having received it with errors. It is a part of the 802.16 MAC layer and can be enabled on a per-connection basis. The 802.16 specification does not mandate the usage of the ARQ mechanism meaning that it is a provider and a customer specific decision.

The 802.16 ARQ mechanism is controlled by a number of parameters. The specification defines them but it does not provide concrete values and solutions. The 802.16 ARQ configuration parameters have not been studied sufficiently, especially by means of extensive simulations. In [7], an analysis of the ARQ feedback types is presented. However, the UDP traffic, which is not sensitive to packet drops, is considered. Furthermore, no algorithm to select the feedback is presented. In [9], the ARQ mechanism is analyzed in the context of real-time flows of small packets. Authors estimate the bandwidth needed for the ARQ feedback messages. However, a simple simulation environment is used that does not capture any of the ARQ configuration parameters. This paper analyzes ARQ parameters and studies their impact on the performance of the ARQ mechanism. In particular, the following parameters are considered: ARQ feedback type, scheduling of ARQ feedbacks and retransmissions, ARQ feedback intensity, ARQ transmission window size, ARQ block size, ARQ block rearrangement. Though the 802.16 specification defines the Hybrid ARQ mechanism, we focus on ARQ because it is applicable to all the PHY types.

This paper extends our previous research and simulation work on 802.16 networks. In [13], [11], we presented a scheduling solution for the 802.16 BS and extensions for the ARQ aware scheduling. In [10], we analyzed the 802.16 contention resolution mechanism and proposed an adaptive algorithm to adjust the backoff parameters and to allocate a sufficient number of the request transmission opportunities.

The rest of the article is organized as follows. Section II presents key features and parameters of the 802.16 ARQ mechanism. We consider their impact on performance and propose a set of solutions. Next, Section III presents a number of simulation scenarios to study the ARQ performance. This section also analyzes the simulation results. Finally, Section IV concludes the article and outlines further research directions.

II. 802.16 ARQ mechanism

A. Basics of the ARQ mechanism

If ARQ is enabled for a connection, the extended fragmentation subheader (FSH) or the extended packing subheader (PSH) is used, which is indicated by the extended bit in the general MAC header (GMH). Regardless of the subheader type, there is a block sequence number (BSN) in the subheader that indicates the *first* ARQ block number in the PDU. A PDU is considered to comprise a number of ARQ blocks, each of which is of the same constant size except the final block which may be smaller. The ARQ block size is an ARQ connection parameter negotiated between the sender and the receiver upon a connection setup. It is worth mentioning that the ARQ block is a logical entity – the block boundaries are not marked explicitly. The remaining block numbers in a PDU can be derived easily on the basis of the ARQ block size, the overall

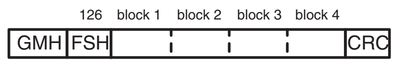
Manuscript received December, 2007 and revised February, 2008.

This paper was presented as part at the Next Generation Teletraffic and Wired/Wireless Advanced Networking (NEW2AN) 2007

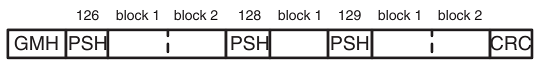
A. Sayenko is with Nokia Research Center, Helsinki, Finland.
(e-mail: alexander.sayenko@nokia.com).

V. Tykhomyrov, H. Martikainen, and O. Alanen are with Telecommunication laboratory, MIT department, University of Jyväskylä, Finland.
(e-mail: {vitykhom,henrik.martikainen,olli.alanen}@jyu.fi)

PDU size, and the first block number. Precisely for these reasons the ARQ block size is a constant parameter. Fig. 1 presents ARQ blocks with the fragmentation and packing mechanisms. Block numbers are given with respect to the BSN stored either in the FSH (see Fig. 1(a)) or PSH (see Fig. 1(b)).



(a) fragmentation



(b) packing

Fig. 1. ARQ blocks with packing and fragmentation mechanisms.

It is important to note that while the 802.16d specification [1] defines an ARQ block size as *any* value ranging from 1 to 2040 bytes, the 802.16e specification [2] has limited it to power of two values ranging from 16 to 1024 bytes, e.g., 16, 32, 64 and so on.

B. ARQ feedback types

To request a retransmission of blocks (NACK) or to indicate a successful reception of blocks (ACK), a connection uses ARQ block sequence numbers. In turn, the sequence numbers are exchanged by means of the ARQ feedback messages. The specification defines the following feedback types: a) selective, b) cumulative, c) cumulative+selective, and d) cumulative+sequence.

The selective feedback type acknowledges ARQ blocks received from a transmitter with a BSN and up to four 16-bit selective ACK maps. The BSN value refers to the first block in the first map. The receiver sets the corresponding bit of the selective ACK map to zero or one according to the reception of blocks with or without errors, respectively. The cumulative type can acknowledge any number of the ARQ blocks. The BSN number in the ARQ feedback means that all ARQ blocks whose sequence number is equal to or less than BSN have been received successfully. The cumulative+selective type just combines the functionality of the cumulative and selective types explained above. The last type, cumulative+sequence, combines the functionality of the cumulative type with the ability to acknowledge reception of ARQ blocks in the form of block sequences. A block sequence, whose members are associated with the same reception status indication, is defined as a set of ARQ blocks with consecutive BSN values. A bit set to one in the sequence ACK map entity indicates that a corresponding block sequence has been received without errors and the sequence length indicates the number of block that are members of the associated sequence.

When the ARQ feature is declared to be supported, a transmitting side, i.e., a receiver of the ARQ feedbacks, must support all the feedback types described by the 802.16 specification. The sender of the ARQ feedbacks has the ability to choose whatever format it will use. The WiMAX Forum recommendations [4] mandate the support of all the types except the selective ACK.

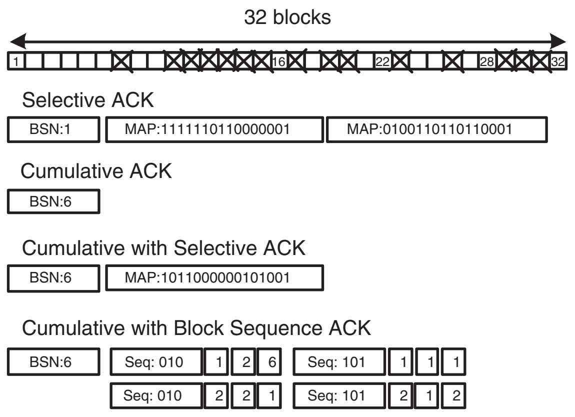


Fig. 2. Example of ARQ feedback types.

Fig. 2 presents an example in which every feedback type is applied to the same set of ARQ blocks. Selective ACK can acknowledge these 32 blocks in two maps. Cumulative ACK cannot acknowledge all the blocks because there are negative acknowledgements. Thus, only six blocks are encoded. Cumulative+selective ACK can send both positive and negative acknowledgements. However, since there should be 16 blocks per one selective map, some blocks remain unacknowledged. For this particular example, cumulative+sequence ACK can acknowledge only 28 blocks; one message can hold four sequence maps at most, whereas each map can have either two or three sequences. This type does not work effectively in this case because the block sequences are very short.

C. Choosing the feedback type

Each feedback type has its advantages depending on the ARQ feedback transmission frequency, the error disturbance patterns, and the computational complexity. From the implementation point of view, the selective feedback type does not require much processing resources because a connection simply puts information on the received blocks into the bitmap. On the other hand, a connection should try to rely upon the cumulative+sequence feedback type if resource utilization is of greater importance. However, it is more complex in implementation because block sequences must be detected. It could form an obstacle for a low power and low capacity mobile device.

In this section, we do not analyze the feedback types from the implementation complexity point of view, but rather propose an algorithm to choose an ARQ feedback type to achieve a good resource utilization. Our algorithm is based on the following assumptions: a) it is always more efficient to send positive acknowledgements by means of the cumulative type, and b) the sequence map can encode more blocks than the selective one. Indeed, the cumulative type can encode *any* number of ARQ blocks by using just one BSN number. Consequently, four sequence maps, each of which can have two sequences of 63 blocks, encode 504 blocks. If a map contains three short sequences, each of which can keep up to 15 blocks, then 180 blocks can be encoded. The proposed algorithm, simplified form of which is shown in Fig. 3, comprises the following three stages:

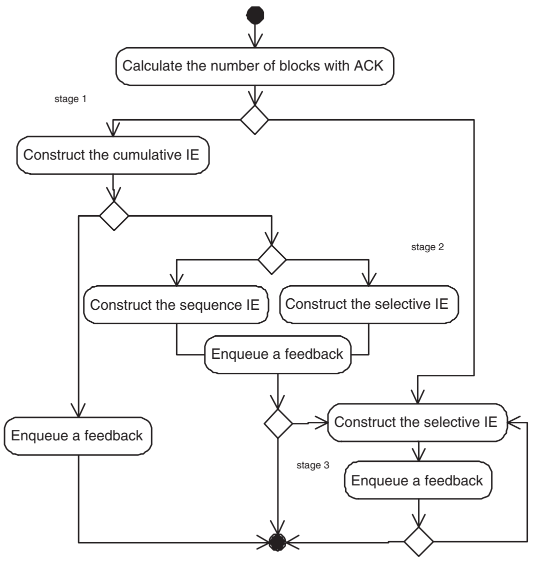

Calculate the number of blocks with ACK
stage 1
Construct the cumulative IE
stage 2
Construct the sequence IE
Construct the selective IE
Enqueue a feedback
Enqueue a feedback
Construct the selective IE
stage 3
Enqueue a feedback

Fig. 3. Algorithm to choose ARQ feedback types.

1) If there are positive acknowledgements in the beginning of the ARQ transmission window, construct the cumulative part. If there are no negative acknowledgements, then a single cumulative feedback message is created.
2) If there are remaining negative acknowledgements (optionally followed by positive and other negative acknowledgements), which we cannot send by using the cumulative part, then we have to choose a map type. To make a decision, we construct the sequence maps and calculate whether the selective maps can acknowledge more blocks. The maximum number of blocks to acknowledge selectively is 64 and it should be a multiple of 16. As for the sequence part, there is a limit for a sequence length and the number of sequences we can send in one message. Eventually, we will have either the cumulative+sequence or cumulative+selective feedback type. As a choice is made, we "attach" map(s) to the cumulative part constructed at the previous stage.
3) Note that we can reach this stage in two cases. The first one is when there are no positive acknowledgements in the beginning of the ARQ transmission window and there is no way to create cumulative, cumulative+selective, or cumulative+sequence types. The second case to reach this stage is when neither cumulative+selective nor cumulative+sequence feedback types encode all the blocks. Though it is a rare case it can happen, because both the cumulative+selective and cumulative+sequence types have technical limitations. It is important to note that we cannot create and send several consecutive cumulative+... feedbacks because the cumulative part of the second message will re-acknowledge positively those ARQ blocks that are acknowledged negatively in the first message. Regardless of the situation, we just create as many selective feedback types as necessary to acknowledge the remaining blocks. As mentioned above, four selective maps can acknowledge up to 64 blocks. It is important to note that due to the clarifications in [3], it is feasible to construct and send the cumulative+sequence feedback type when there are negative acknowledgements in the beginning of the ARQ window. It is possible to put out of the Tx window BSN field in the cumulative part so that it is ignored at the sender (receiver of the ARQ feedback). Such a solution eliminates the need for the selective type when there are errors in the beginning and improves the MAC overhead.

It is worth noting that the presented algorithm scales well to the SS capabilities. If the selective type is not supported, then stage 3 is never executed. If there is no support for one of the cumulative types, then stages 1 and 2 are simplified.

Referring back to stage 2, it is worth mentioning an algorithm to create sequences for the cumulative+sequence ARQ feedback type. The specification does not define it thus allowing alternative implementations. As mentioned before, it is more complex in implementation because block sequences must be detected and correct sequence lengths must be constructed. To simplify this process, the algorithm uses two steps. On the first step, the algorithm parses all blocks and constructs sequences without checking any lengths. On the second step, the algorithm chooses sequence formats and, if necessary, splits large sequences into smaller ones so that they conform to the specification. The algorithm analyzes the current and the next sequence length to decide which sequence format should be used. As the sequence format is chosen, sequences are put into a map. If the sequence length exceeds the technical limit (63 for the format 0 and 15 for the format 1), then it is truncated and the remaining part is written into the input list so that it is processed at the next iteration. The simplified form of this algorithm is presented in Fig. 4. The algorithm stops when either all the sequences are processed or four maps are built. If there are not enough sequences to fill a single map, then zero lengths are put.

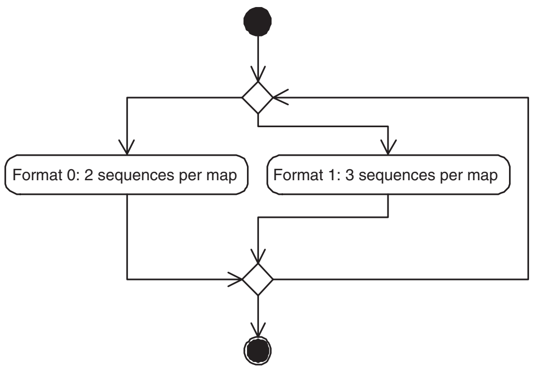


Fig. 4. Algorithm to construct sequence maps.

As mentioned above, two sequence formats are available. The algorithm uses a simple, yet powerful, condition to select an appropriate sequence format:

$$\text{Format} = \begin{cases} 0, & (S_i > 15) \text{ OR } (S_{i+1} > 15), \\ 1, & \end{cases} \quad (1)$$

where S_i is the ith sequence length. The idea behind (1) is that it is more resource conserving to switch to the sequence format 0 if either current or the next sequence is a large one. Otherwise, it is more efficient to use format 1 to encode more short sequences. As an example, Fig. 5 presents the input sequences and constructed sequence maps (the first number is the sequence status while a number in the square brackets is a sequence length). As can be seen, the algorithm and proposed conditions split efficiently sequences into the maps according to the 802.16 specification.

```
0[65] 1[17] 0[3] 1[11] 0[4] 1[67] 0[40]

   map #1    |   map #2    |      map #3        |    map #4
  format 0   |  format 0   |     format 1       |   format 0
 0[63] 0[2]  | 1[17] 0[3]  | 1[11] 0[4] 1[15]   | 1[52] 0[40]
```

Fig. 5. Input sequence lengths and built maps

To illustrate that the proposed algorithm selects efficiently the required sequence format, we present two cases for the same initial input sequences when only one particular format is in effect. As can be seen from Fig. 6, both cases fail to encode all the blocks (due to space limitations, only three maps are presented for the sequence format 1).

```
   map #1    |   map #2    |   map #3    |   map #4
  format 0   |  format 0   |  format 0   |  format 0
 0[63] 0[2]  | 1[17] 0[3]  | 1[11] 0[4]  | 1[63] 1[4]
```
(a) format 0 maps only

```
      map #1        |       map #2        |      map #3       |
     format 1       |      format 1       |     format 1      |
0[15] 0[15] 0[15]   | 0[15] 0[5] 1[15]    | 1[2] 0[3] 1[11]   |
```
(b) format 1 maps only

Fig. 6. Built maps (only one particular format).

The resulting computational complexity of the proposed algorithm to construct sequence maps is $O(2N)$. We need to make two passes: the first one is to calculate the initial sequence lengths and the second one is to split sequences between the maps. The computational complexity of the selective map is $O(1)$.

D. Ordering of feedbacks and retransmissions

While sending normal packets, retransmissions, and ARQ feedback messages, a connection should determine their order. Indeed, as a scheduler at the BS allocates resources to a connection, either uplink or downlink, a connection's internal priority mechanism should decide upon which message is of more importance.

We propose to send first the ARQ feedbacks, then retransmissions, and finally the normal user PDUs. The reason we assign the highest priority to the ARQ feedbacks is that they do not require much space and they have a huge impact on the ARQ performance. As a sender receives a feedback, it knows the blocks that were received successfully and the blocks that are to be retransmitted. The successfully transmitted blocks can be removed from the retransmission buffer and the associated resources are cleared (see section II-H). Furthermore, the sender adjusts the ARQ transmission window that, in turn, influences the performance, because a connection cannot send more blocks than the ARQ window allows.

The reason we assign a higher priority to retransmissions is that a receiver can reconstruct a MAC service data unit (SDU) from fragments and forward it to the upper level only once all the fragments are received. Furthermore, if the *ARQ deliver in order* option is turned on,[1] then a receiver is obliged to forward SDUs in the same order in which a sender transmits them. This means that even though a receiver reconstructs successfully an SDU from all the fragments, it has to wait for all the previous SDUs.

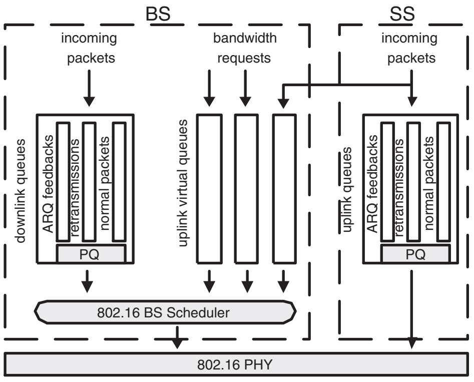


Fig. 7. Queue structure to prioritize feedbacks and retransmissions.

The simplest way to organize these priorities is to introduce several internal subqueues within a connection queue, as Fig. 7 illustrates. It is an extended version of the 802.16 QoS architecture considered in [5], [13]. Every time a PDU arrives to the connection queue, it will be checked and depending on its type it will be placed into an appropriate subqueue. When PDUs are dequeued, the queue can check first the subqueue with the ARQ feedbacks, then the subqueue with retransmissions, and only then the subqueue with normal PDUs. In other words, a connection queue implements internally the strict priority queuing.

An appealing feature of this approach is that it is completely transparent to the BS scheduler. Everything the BS scheduler needs to know to allocate resources is connection QoS requirements, if any, and the queue size [13]. If there are several internal subqueues, then the BS scheduler will be informed about the *aggregated* queue size. It is especially the case for the uplink virtual queues that are maintained through bandwidth requests sent by SSs. An SS cannot inform about the size of each subqueue but rather about the aggregated size. When a connection is allotted slots, first it will send ARQ feedbacks. If there are remaining bytes in a data burst, the connection will send retransmissions, and only then normal PDUs will be sent.

[1]It is anticipated that this option will be turned on for most services. Indeed, there is no sense in turning this option off for the UDP based applications, such as VoIP. The VoIP receiver will just discard packets that arrive in the wrong order unless some sufficiently larger input buffer is utilized, which is not typical for interactive applications. In the case of the TCP based services, an absence of a packet can be treated as a packet drop. It will trigger a retransmission of this packet though it can arrive later.

E. ARQ feedback intensity

Though IEEE 802.16 specification defines ARQ feedback types, it does not specify how often a receiver should send them. We considered the ordering of the ARQ feedbacks and retransmissions in section II-D , however, it does not provide an answer *when* a receiver should construct the ARQ feedback message and place it into the output queue.

Intuitively, it is understandable that the ARQ feedback intensity is a tradeoff between the MAC overhead and the robustness of the ARQ state machine. On the one hand, we may delay sending ARQ feedbacks to decrease the MAC overhead. On the other hand, failing to send the ARQ feedback on time may result in a very bad performance because ARQ blocks will be discarded by the ARQ timers. If the ARQ feedback transmission period T_{feedback} is less than the ARQ retry timer T_{retry}, then the performance starts to decline because a sender will retransmit the same data. If the feedback intensity is even less than the ARQ block lifetime T_{life}, then it may result in a very poor performance due to the discarded ARQ blocks. Based on that it is possible to propose the following inequality:

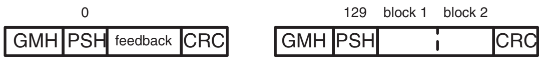
$$T_{\text{feedback}} < T_{\text{retry}} < T_{\text{life}}. \tag{2}$$

Since the ARQ retry and life timers are the connection specific parameters, the receiver can always adapt its ARQ feedback intensity on a per-connection basis. Since it is usually the case that the retry timeout is less than the life timeout, it is enough to analyze the retry timer value to choose a suitable ARQ feedback intensity.

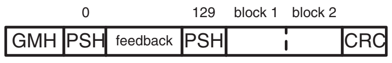
It is worth mentioning that the ARQ feedback intensity should not be very close to the ARQ retry timeout. The reason is that the ARQ feedback message can be dropped due to the failed checksum test, as any other PDU.

F. Standalone and piggy-backed feedbacks

While sending the ARQ feedback message, a connection has an option whether to send it as a standalone message or piggy-back it to a PDU with user data (see Fig. 8). The former approach has somewhat larger MAC overhead of 12 bytes because the ARQ feedback resides in a separate PDU with mandatory GMH and PSH headers, and the CRC field. At the same time, the piggy-backed transmission is less reliable when compared to a standalone message. The reason is that being piggy-backed to a large PDU, the ARQ feedback has a higher probability of being dropped [8] because the whole PDU is discarded when an error is detected. If a sender does not receive any feedback before the ARQ retry timer expires, then correspondent ARQ blocks will be retransmitted. No need to say that a loss of the ARQ feedback message will lead to the retransmission of all ARQ blocks, even of those ones that have been received correctly. As mentioned in section II-E, if a sender does not receive any ARQ feedback before the ARQ block life timeout, then blocks will be discarded completely. Thus, to achieve a more reliable transmission of the ARQ feedbacks, it makes sense to rely upon the standalone feedbacks.



(a) standalone PDU



(b) piggy-backed to a PDU with user data

Fig. 8. ARQ feedback transmission.

G. ARQ block rearrangement

While retransmitting a PDU, a connection may face a problem that an allocated data burst is smaller than the PDU size to be retransmitted. This may happen if the BS scheduler allocates data bursts of different sizes, which is usually the case for real-time Polling Service (rtPS), non-real-time Polling Service (nrtPS), and Best Effort (BE) connections. Suppose, that the BS allocates a data burst of three slots for the BE connection and the latter sends a PDU that spans the whole data burst. If this PDU encounters an error, the connection will retransmit it. However, if the BS scheduler allocates later a data bursts of two slots, there is no way to retransmit the original PDU. Fortunately, the connection may rely upon *the retransmission with rearrangement* that allows for fragmenting the retransmitted PDU on the ARQ block size boundaries. If there is a sufficiently small ARQ block size, then the connection may construct a smaller PDU. As an example, Fig. 9 shows the rearranged PDU presented in Fig. 1(a). There are two PDUs with two blocks per each PDU.

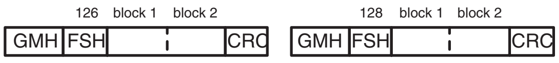


Fig. 9. Rearranged PDU.

In this subsection we do not focus on the optimal ARQ block size, but rather consider a solution for a case where a sender retransmission policy is not to use the ARQ block rearrangement. The reason this functionality can be absent is the fact that rearrangements involve much more complicated actions with PDUs in the retransmission buffer when compared to the PDU construction. A sender must keep a set of the ARQ timers for each ARQ block. If the retransmission with rearrangement is not implemented, then eventually a sender can associate all those timers with a PDU, which requires much less resources.[2] Furthermore, the rearrangement requires a sender to analyze a PDU and to search for block boundaries on which that PDU can be fragmented.

It is important to note that this problem concerns merely the uplink connections, because having the bandwidth request size, the BS does not know whether it is one big PDU or several smaller ones. In the case of the downlink transmission, the BS can always look inside the queue. Besides, this problem

[2]Practically, a sender can associate a timer with a whole PDU even if the ARQ block rearrangement is turned on. However, then it has to perform quite complicated actions with ARQ timers when the retransmitted PDU is partitioned into several PDUs because certain ARQ blocks are retransmitted while the other ones remain in the output buffer.

would not be so critical if the BS knew that a connection does not support rearrangements. However, there is no such QoS parameter that would indicate it. On the one hand, the BS can guess that a connection does not rearrange PDUs by monitoring bandwidth request sizes and the number of received bytes. On the other hand, a connection should not rely much upon this functionality because it is not mandated by the specification. Thus, the only safe way is to control the maximum size of transmitted PDUs. It is not a complicated task for the rtPS and nrtPS connections that should be always allocated such a number of slots that their minimum bandwidth requirements are ensured [13]. Thus, the maximum PDU size can be limited by the minimum data burst size allocated by the BS scheduler. The BE scheduling class is a more challenging task since the BS scheduler can allocate a data burst of any size. A connection may monitor allocated data burst sizes to control the maximum PDU size. Another possible solution is to send as large PDU as the size of one slot. However, such an approach may be unacceptable due to the increased MAC overhead and very small slot size of robust MCSs. As a result, regardless of an approach taken, the BE connection, which does not support retransmissions with rearrangements, should avoid sending large PDUs.

H. ARQ transmission window and ARQ block size

At any time a sender may have a number of outstanding and awaiting acknowledgements ARQ blocks. This number is limited by the ARQ transmission window that is negotiated between an SS and the BS during a connection set-up. A sufficiently large ARQ window allows for a continuous transmission of data. A connection can continue to send ARQ blocks without waiting for each block to be acknowledged. Conversely, a smaller ARQ window causes a sender to pause a transmission of new ARQ blocks until a timeout or the ARQ feedback is received. Though it may seem that a large transmission window is always the best choice, it is worth noting that a large transmission window leads to increased memory consumption and processing load. Every ARQ block must be stored in the retransmission buffer until a positive feedback is received. Taking into account the largest ARQ block size of 1024 bytes and the maximum ARQ transmission window of 1024 blocks, it is possible to arrive at the conclusion that some mobile and portable devices will not have enough resources to handle this amount of data for each frame.

If we assume a continuous errorless data transmission, then the maximum throughout a connection can achieve is limited by the following expression:

$$\frac{S^{\mathrm{ARQ}}\, W\, \mathrm{FPS}}{\mathrm{DF}}, \tag{3}$$

where S^{ARQ} is the ARQ block size, W is the ARQ transmission window size, FPS is the number of frames per second and DF is the delay factor. In the case of the downlink transmission, the delay factor is always 1 because the BS can allocate a downlink data burst whenever it wants. In the case of the uplink transmission, the delay factor depends on PHY and whether a polling is in effect. If the BS polls a connection in *every* frame, then the delay factor is also 1. Otherwise, like in the case of the BE connections, the delay factor is 2 for OFDM and 3 for OFDMa PHY. The reason is that in OFDM PHY, the uplink bandwidth request carries the request size, while in the OFDMa PHY, special CDMA codes are used that do not carry any request size. As a result, once the BS receives the CDMA code, it puts a special uplink CDMA allocation where an SS can transmit the request size.

The ARQ transmission window and the ARQ block size parameters depend one on each other. On the one hand, a connection may prefer to work with a small ARQ transmission window that will result in a necessity of choosing a larger ARQ block size because the throughput may be limited by the transmission window size. A large block size requires less resources because a set of the ARQ timers must be associated with a single ARQ block at the sender and the receiver. At the same time, a connection supporting the retransmission with rearrangement may wish to work with a smaller ARQ block size because that will provide a greater flexibility in splitting large PDUs into several smaller ones. Furthermore, the choice for the ARQ block size can be dictated by the device peculiarities, such as the memory page size. These various requirements introduce a cyclic dependency between these two parameters.

We anticipate that the ARQ block size should be the governing parameter, while the ARQ transmission window size should be adapted. The reason is that the ARQ block size has a set of discrete values, while the ARQ transmission window can accept any value within the specified range.

III. Simulation

This section presents a simulation analysis of the 802.16 ARQ mechanism. To run simulations, we have implemented the 802.16 MAC and PHY levels in the NS-2 simulator. The implementation is called WINSE (WiMAX NS-2 Extension). The MAC implementation contains the main features of the 802.16 standard, such as frames, bursts, downlink and uplink transmission, connections, MAC PDUs, packing and fragmentation, the contention and ranging periods, the MAC level management messages, dynamic size of the MAP messages, and the ARQ mechanism. The ARQ implementation supports the ARQ blocks, the ARQ transmission window, retransmission with rearrangement, all the ARQ feedback types, and the ARQ timers. The ARQ implementation also includes the prioritization of the feedbacks and retransmissions, and the algorithm to select the feedback type and to build block sequences. The implemented PHY is OFDMa. The simulation results for the OFDM PHY can be found in [13].

Fig. 10 shows the network structure we use in the simulation scenarios. There is the BS controlling the 802.16 network, the parameters of which are presented in Table I.[3] To compare results fairly, we run somewhat simplified PHY model with a fixed signal to noise ratio of 2 dB, which corresponds to QPSK3/4 MCS, and forward error correction (FEC) block error rate of 1%. The downlink broadcast messages, such as DL-MAP and UL-MAP, use a more robust QPSK1/2 MCS;

[3]These parameters conform the WiMAX Forum mobile system profile [4].

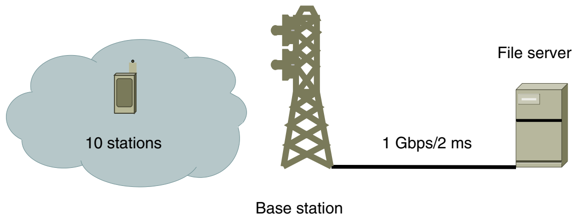


Fig. 10. Network structure.

TABLE I
802.16 NETWORK PARAMETERS.

Parameter	Value
PHY	OFDMa
Bandwidth	10 MHz
FFT	1024
Cyclic prefix length	1/8
TTG+RTG	464 PS
Duplexing mode	TDD
Frames per second	200 (5 ms per frame)
OFDM symbols	47
DL/UL symbols	26/21
DL/UL subcarrier alloc.	DL FUSC/UL PUSC
DL/UL slots	416/245
MAP MCS	QPSK1/2 (6 B/slot)
MCS	QPSK 3/4 (9 B/slot)
FEC BLER	1%
Ranging transm. opport.	2
Ranging backoff start/end	2/15
Request transm. opport.	8
Request backoff start/end	4/15
CDMA codes	256
ranging+periodic ranging	64
bandwidth request	192
handover	–
Fragmentation/packing	ON
PDU size	unlimited
CRC/ARQ	ON
ARQ feedback	standalone
ARQ feedback types	all
ARQ feedback intensity	5 ms
ARQ block size	16 B
ARQ window	1024
ARQ discard	ON
ARQ block rearrangement	ON
ARQ deliver in order	ON
ARQ timers	
retry	50 ms
block lifetime	200 ms
Rx purge	200 ms

they are never dropped in our simulations. The BS runs the scheduling algorithm, details of which are presented in [13], [12]. In a few words, if there are only the BE connections, then the BS allocates resources fairly between the SSs based on their bandwidth request sizes. In addition, the ARQ aware scheduling is deployed to the BS station scheduler [11].

The BS scheduler also reserves two transmission opportunities for the initial ranging purposes (as in real life, an SS has to join the network in our simulator) and eight transmission opportunities for the bandwidth request contention resolution. The backoff parameters are given in Table I. The distribution of the CDMA contention codes is also given in Table I (since we do not simulate any mobility, there are no CDMA handover codes).

The simulation environment includes one wired node and ten SSs. Each SS establishes the basic management connection to exchange the management messages with the BS. In addition, to exchange user data, an SS establishes one uplink and downlink BE connection. An SS hosts exactly one FTP-like application that downloads data from a wired node over the TCP protocol. The reason we choose such an application type is that it tries to send as much data as possible thus utilizing all the network resources. At the same time, the TCP protocol is very sensible to packet drops that can occur in the wireless part. Each simulation run lasts for 10 seconds. The actual data transmission starts at the 1.5th second of the simulation run because first SSs has to enter the cell and register at the BS.

A. General ARQ results

In this simulation scenario we present general results concerning the ARQ performance. Fig. 11 presents the downlink throughput when neither ARQ nor errors are enabled. The throughput is calculated at the upper MAC level of the SS wireless interface, i.e., when the SS reconstructs original packets from received PDUs. As can be seen, all the BE connections have almost identical throughput. Since there are no QoS requirements, the BS scheduler allocates resources fairly between them.

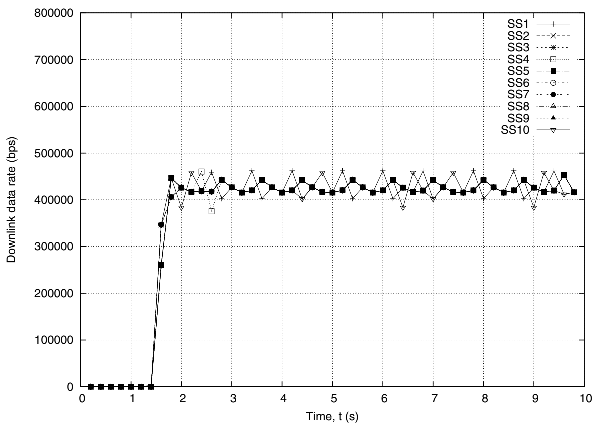


Fig. 11. Downlink throughput (no errors, no ARQ).

If we enable errors at the PHY level but keep the ARQ mechanism disabled for the transport connections, then there will be no smooth transmission anymore. As Fig. 12 illustrates, there is quite a bursty downlink data transmission. Some SSs even do not send data for some periods of time. Such a behavior is explained by the fact that the receiver does not test whether there is an erroneous PDU or not – it passes all the reconstructed SDUs to the wired node. Thus, the error detection and retransmission occurs at the transport layer which affects greatly the throughput.[4] It is worth mentioning that Fig. 12 presents even somewhat optimistic results because there is a small round-trip delay between the source subscriber stations and the destination wired node. As it becomes larger, the throughput would decline appropriately.

[4]Practically, a connection may include the CRC field into the PDU without enabling the ARQ mechanism. It will prevent a receiver from forwarding erroneous PDUs. However, a retransmission will still occur at the transport level.

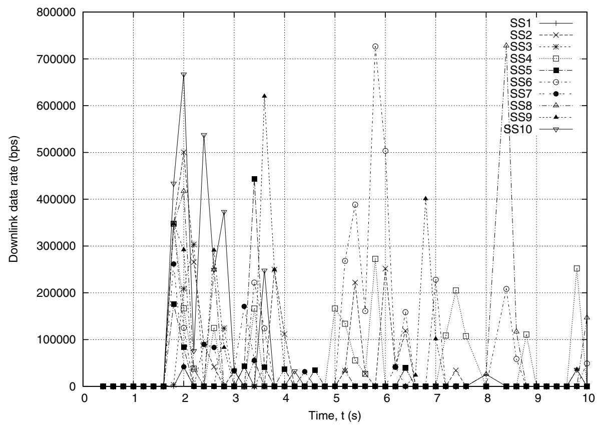


Fig. 12. Downlink throughput (errors, no ARQ).

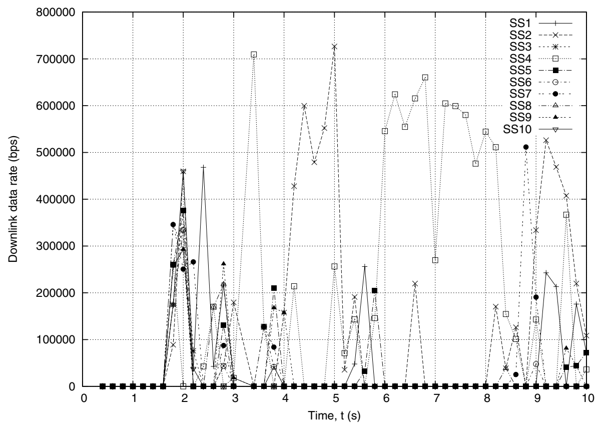


Fig. 14. Downlink throughput (errors, ARQ, no ARQ priority).

Fig. 13 shows the connection throughput when errors at the PHY level and the MAC ARQ mechanism are enabled. As follows from the figure, each BE connection achieves a smooth data transmission. Since there are errors in the PHY channel, the mean connection throughput is less than in Fig. 11. Nevertheless, the ARQ mechanism ensures extremely good resource utilization. The fluctuations are explained by the fact that PDUs are dropped and retransmitted.

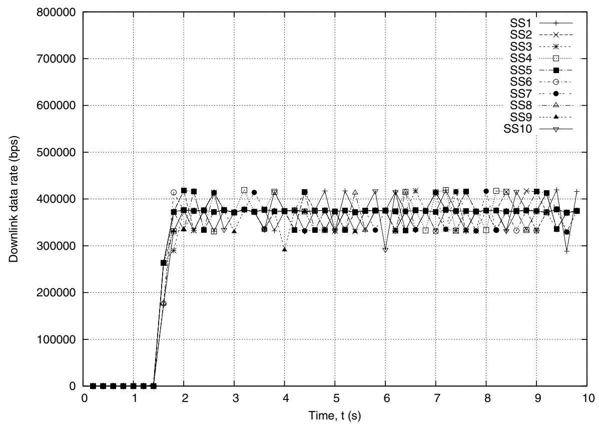


Fig. 13. Downlink throughput (errors, ARQ).

Fig. 14 presents the simulation results when the ARQ *priority* is absent, i.e., the ARQ feedbacks and retransmissions are transmitted in exactly the same order as they are put into the connection output queue. As can be seen from the figure, there are bursty changes in the uplink connection throughput, similar to the case when the ARQ mechanism is completely disabled. As considered in II-D, failing to prioritize ARQ feedbacks and retransmissions leads to a situation when the sender does not receive immediately information on ARQ blocks to retransmit thus resulting in a low performance.

Table II provides a comparison of these subcases by using another criterion, the total amount of downlink data. The amount of uplink data is much less and, due to the TCP behavior, is proportional to the downlink data. As follows from the results, an absence of the ARQ mechanism when there are errors in the transmission channel (which is usually a case for the wireless networks) results in a very low resource utilization. Table II also presents an interesting subcase when the ARQ is turned on, but errors are turned off. Its purpose is to show that ARQ introduces some overhead to the MAC level. Finally, absence of the priority for the ARQ retransmissions and the ARQ feedback messages decreases significantly the overall performance.

TABLE II
AMOUNT OF TRANSFERRED DATA.

ARQ	ARQ priority	errors	Downlink data (MB)
–	–	–	4.296
√	√	–	4.097
–	–	√	0.392
√	√	√	3.718
√	–	√	0.592

B. ARQ block rearrangement

In this subsection, we study the ARQ retransmission with rearrangement. The network parameters are the same as presented in Table I. There are ten SSs that download from the wired node through the BS. To demonstrate the ARQ block rearrangement importance, we turn on/off this feature and adjust the PDU size.

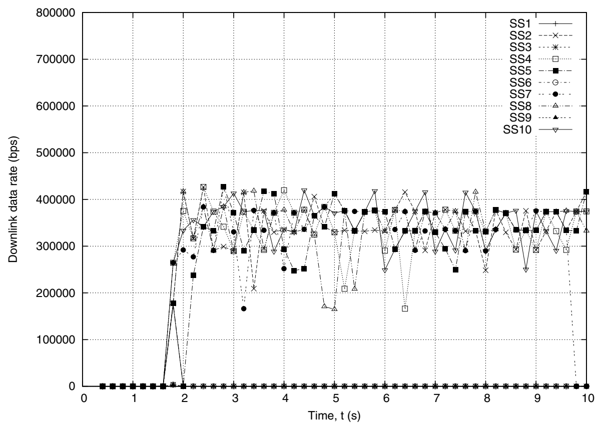


Fig. 15. Downlink throughput (no ARQ block rearrangement, large PDU).

Fig. 15 shows the throughput of downlink transmission when the ARQ block rearrangement is *turned off*. As can be seen from the figure, uplink connection throughputs are not smooth but rather change drastically. This is a result of the insufficient size of the uplink data burst when a connection retransmits a PDU. As explained earlier, while a connection may transmit a large PDU, an attempt to retransmit the same PDU may fail if the BS allocates later a data burst of a smaller size.

If a connection does not support the ARQ block rearrangement, then a possible solution is to use a smaller PDU size. Fig. 16 shows the downlink throughput for exactly the same case, but now all the connections have the maximum PDU size of 108 bytes, the ARQ block rearrangement is turned off.

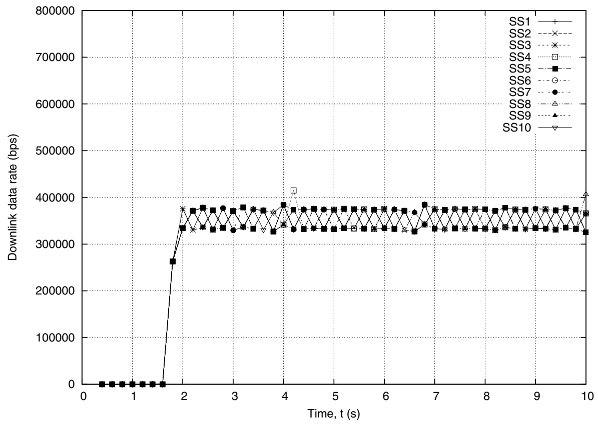


Fig. 16. Downlink throughput (no ARQ block rearrangement, PDU size is 108 bytes).

If we compare Fig. 16 (small PDU, no ARQ block rearrangement) and Fig. 13 (unlimited PDU size, ARQ block rearrangement), we may notice that the ARQ block rearrangement has an impact on the performance. Connections can use large PDUs of any size thus decreasing the MAC level overhead. At the same time, all the connection achieve a smooth data transmission. It is noticeable that an average connection throughput in Fig. 16 is less than in Fig. 13, which is explained by the MAC overhead caused by the small PDU size.

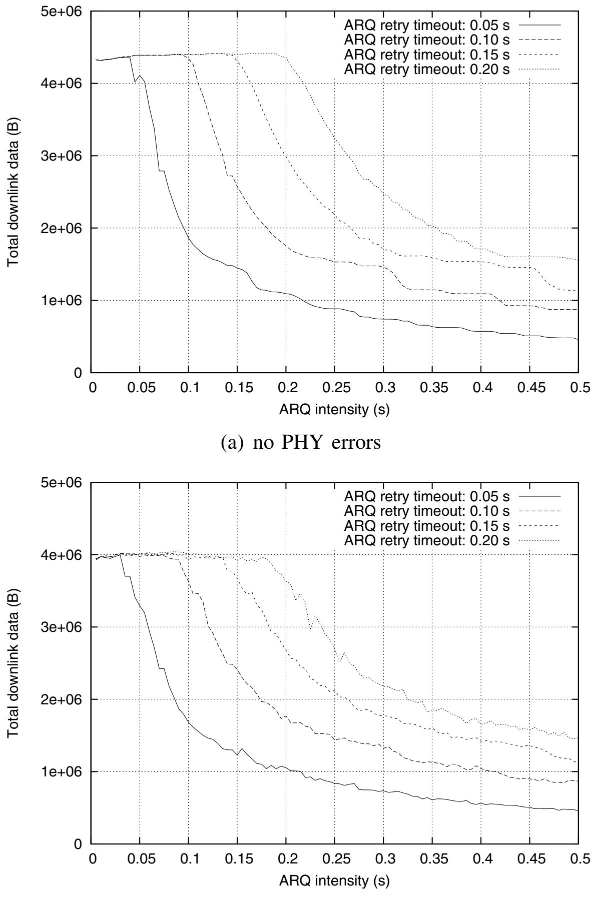
TABLE III
AMOUNT OF TRANSFERRED DATA.

Rearrangement	PDU size (B)	Downlink data (MB)
√	unlimited	3.718
–	unlimited	2.062
–	108	3.581

Table III also shows the amount of downlink data for this simulation scenario. As follows from the results, a connection should consider smaller PDU sizes if the ARQ block rearrangement functionality is not supported.

C. ARQ feedback intensity

In this simulation subcase, we study the impact of the ARQ feedback intensity on the network resource utilization. The network parameters are the same as in the previous simulations scenarios, the only difference is that we use the ARQ block size of 128 bytes. Otherwise, with a low ARQ feedback intensity the ARQ transmission window may get full and the transmission will stall. Also, the ARQ block lifetime timer is increased to 1.3 seconds to prevent ARQ blocks from being discarded when the low ARQ feedback intensity is in effect.



(a) no PHY errors

(b) PHY errors

Fig. 17. Total downlink data

Fig. 17 presents the simulations results for different ARQ feedback intensity and ARQ retry timeout values. According to Fig. 17(a), less frequent ARQ feedback messages allow connections to achieve better throughput due to the decreased MAC overhead. However, as the ARQ feedback transmission interval value is close to the ARQ retry timeout value, the MAC utilization starts to decline because a sender retransmits the same data. If there are errors in the wireless channel, then a lower ARQ feedback intensity results only in a marginal improvement, as Fig. 17(b) shows. At the same time, the MAC utilization starts to degrade even earlier than the ARQ retry timeout value. The reason for this is that ARQ feedback messages can be dropped, as any other PDU. Thus, a higher ARQ feedback intensity introduces a redundancy into the ARQ feedback mechanism – even if one feedback message is lost, the next one will duplicate the information.

Based on the presented simulation results, it is possible to arrive at the conclusion that the ARQ feedback transmission interval must be at least two times less than the ARQ retry timer. A higher ARQ feedback intensity results only in a slightly increased MAC overhead. At the same time, it improves the robustness of the ARQ feedback mechanism. More results including the ARQ feedback intensity over the HARQ enabled connections can be found in [14]

D. ARQ feedback types

In this simulation subcase, we study the ARQ feedback types. The network parameters are the same as in the previous simulations scenarios, the only difference is that we use different ARQ block size values and the ARQ feedback transmission interval is set to 40 ms.

TABLE IV
THE ARQ FEEDBACK TYPES STATISTICS.

ARQ block (B)	Feedback type percentage (%)				Num. of msg.	Downlink data (MB)
	Sel	Cum	Cum+ sel	Cum+ seq		
16	0	92.025	0	7.975	3586	2,769
32	0	91.910	0	8.091	3535	2,773
64	0	92.125	0	7.875	3543	2,769
128	0	91.556	0	8.444	3541	2,759
256	0	92.033	0	7.967	3477	2,817
512	0	92.312	0	7.688	3278	2,757
1024	0	90.547	0	9.453	2941	2,635

Table IV shows the results for these simulation runs. The total number of ARQ feedback messages sent in each simulation run and the percentage of each ARQ feedback type are presented. As can be seen, neither selective nor cumulative+selective feedback messages are sent during the simulation runs. As explained earlier, it is almost always more efficient to send acknowledgments by means of the cumulative+sequence type that can encode more blocks than the cumulative+selective. If there are only positive acknowledgments, then the cumulative feedback type is used. As follows from the table, the majority of the ARQ feedback messages are of this type. As explained earlier, due to the clarifications in [3], it is feasible to construct and send the cumulative+sequence feedback type when there are negative acknowledgements in the beginning of the ARQ window. It is possible to put out of the Tx window BSN field in the cumulative part so that it is ignored at the sender (receiver of the ARQ feedback). Such a solution eliminates the need for the selective type and improves the MAC level utilization.

It was anticipated that as we increase the ARQ block size, the number of the ARQ feedback messages should decline. As follows from Table IV, it is indeed so. It is also important to note that the best performance is achieved for the ARQ block size of 256 bytes. Smaller ARQ block sizes create a larger MAC overhead, while larger ARQ block sizes result in a higher PDU error rate [8] because the minimum PDU size should be large enough to carry at least one ARQ block [11]. Besides, as mentioned earlier, large ARQ block size values may prevent a connection from utilizing all the burst size because the PDU is fragmented and retransmitted on the ARQ block boundaries. If a connection uses a large ARQ block size then it is less flexible in retransmitting PDUs. Thus, the optimal ARQ block size is a tradeoff between the PDU error rate and the number of the ARQ feedback messages, which cause the ARQ overhead at the MAC level.

TABLE V
THE ARQ FEEDBACK TYPES STATISTICS.

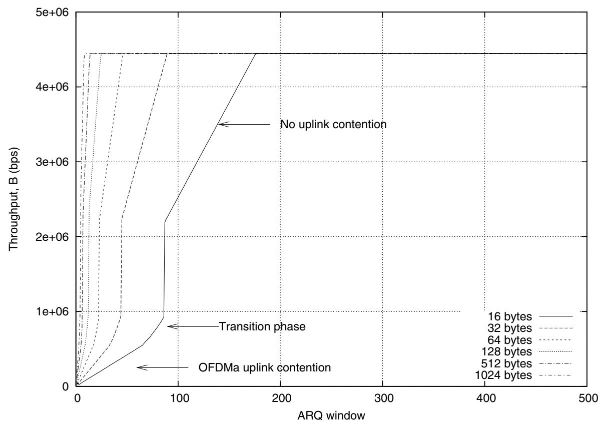
ARQ block (B)	Feedback type percentage (%)				Num. of msg.	Downlink data (MB)
	Sel	Cum	Cum+ sel	Cum+ seq		
16	100	0	0	0	5639	2,680
32	100	0	0	0	3621	2,644
64	100	0	0	0	3382	2,356
128	100	0	0	0	3351	2,394
256	100	0	0	0	3411	2,758
512	100	0	0	0	3221	2,697
1024	100	0	0	0	2853	2,596

Table V shows the results where only the selective ARQ feedback type is enabled. As expected, there are more ARQ feedback messages, especially for small ARQ block sizes, such as 16 bytes. If we compare the amount of transferred data in Table V and Table IV, then we can arrive at the conclusion that the selective ARQ feedback type does not result in a severe performance degradation. Thus, being combined with larger ARQ block sizes, it can be a valid choice for certain mobile devices with limited computational resources.

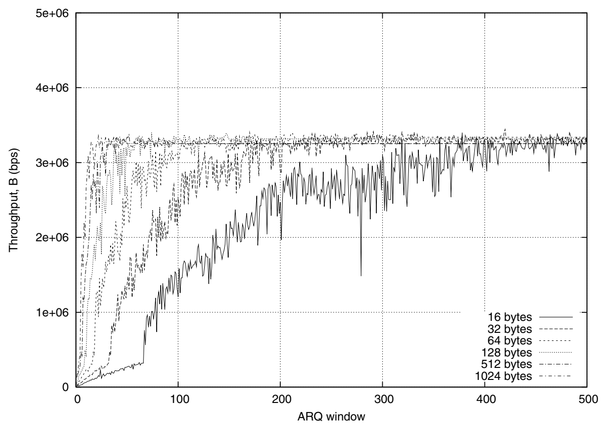
E. ARQ transmission window

In this subsection we study the impact of the ARQ transmission window on the throughput. The network parameters are the same as in the previous simulations scenarios, the only difference is that we vary the ARQ transmission window and block sizes. There is only one SS, otherwise it would be difficult to present an analysis of the throughput of all the SSs. We run a separate simulation for each ARQ transmission window value and ARQ block size. Since an SS throughput fluctuates during a simulation run, it is averaged by using the exponentially weighted moving average algorithm.

Fig. 18 presents the simulations results for this case with the PHY errors turned off and on. The figure indicates that large ARQ block sizes allow a connection to achieve its maximum throughput even for small ARQ transmission window values. Conversely, a small ARQ block value needs a large ARQ transmission window to achieve a high throughput. In the case of the errorless transmission, as the ARQ transmission window grows, the throughput increases linearly regardless of the ARQ block size. Of course, it grows faster for larger ARQ block sizes. When the ARQ transmission window reaches a certain value, its further growth does not have an impact on the throughput because the latter is limited by the overall network capacity. It is noticeable that regardless of the ARQ block size value, there are several phases in how the throughput increases. (see Fig. 18(a)). In the beginning, it grows very slow due to the fact the stations have to take part in the uplink connection resolution to send to the BS TCP acknowledgements and the ARQ feedback messages. In this case, the throughput is approximated accurately by (3) with the delay factor of 3. When a certain point is reached, there is a continuous uplink transmission due to the increased downlink traffic. Stations do not take part in the uplink contention resolution anymore as



(a) no PHY errors



(b) PHY errors

Fig. 18. Downlink throughput and ARQ window.

they piggy-back their bandwidth requests to user data. In this case, the throughput is approximated accurately by (3) with the delay factor of 1. A similar behavior is observed when there are PHY errors (see Fig. 18(b)). The only difference is that throughput increases much slower due to the PDU retransmissions.

Fig. 18 illustrates clearly that small ARQ transmission window values may prevent a connection from sending data even if it has slots allocated by the BS scheduler. Though it is not a huge problem for the BE connections, one should account for it if there is a QoS connection with the minimum bandwidth requirements.

IV. Conclusions

In this paper, we have analyzed the performance of the 802.16 ARQ mechanism. We have shown that the ARQ mechanism can improve significantly a performance of the TCP based applications. Since a probability for an erroneous transmission in the wireless channel is much higher when compared to the wired medium, the ARQ mechanism should be enabled for the TCP connections if a provider wants to ensure better QoS and to maximize the network utilization. Though we did not present simulation results for the UDP protocol, it is clear that its performance would not be affected by the absence of the ARQ mechanism because the UDP transmission does not depend on packet drops.

We have proposed a solution on how to prioritize normal PDUs, ARQ feedbacks, and retransmissions. The simulation results have also revealed the importance of the ARQ block rearrangement functionality. If an SS does not support it, then an additional care must be taken. An SS should choose smaller PDU sizes to achieve a smooth data transmission. We have also demonstrated that a connection must choose a sufficiently large ARQ transmission window size to utilize the allocated resources. While large ARQ blocks can utilize resources even with a small ARQ window, small ARQ blocks, such as those of 16 and 32 bytes, require much larger ARQ window. We proposed lightweight, yet efficient, algorithms to select the ARQ feedback type and to build block sequences for the cumulative+sequence feedback type. Besides, the selective ARQ feedback type does not result in a severe performance degradation; mobile devices with scarce computational resources may rely safely upon it. If a receiver can adjust the ARQ feedback intensity, then it better to rely upon a higher ARQ feedback intensity to avoid retransmissions activated by the ARQ retry timer. In any case, the ARQ feedback transmission interval must not be less than the ARQ retry timer.

Our future research will aim at studying the optimal parameters of the ARQ mechanism, which is especially the case for the ARQ-enabled QoS connections. It is also important to compare the results provided by the ARQ mechanism and the HARQ mechanism available in the OFDMa PHY.

Acknowledgments

We would like to express our acknowledgments to Jani Lakkakorpi (Nokia) and Jani Moilanen (Nokia Siemens Networks) for their valuable discussions on the IEEE 802.16 ARQ mechanism functioning. Special thanks to Jan Suumäki (Nokia) for providing a valuable insight on the ARQ mechanism implementation and reviewing the paper.

References

[1] Air interface for fixed broadband wireless access systems. IEEE Standard 802.16, Jun 2004.

[2] Air interface for fixed broadband wireless access systems - amendment for physical and medium access control layers for combined fixed and mobile operation in licensed bands. IEEE Standard 802.16e, Dec 2005.

[3] Air interface for fixed and mobile broadband wireless access systems. IEEE Standard 802.16 (Corrigendum 2/D4), May 2007.

[4] WiMAX Forum Mobile System Profile, Release 1.0 Approved Specification, Nov 2007. Revision 1.5.0.

[5] C. Cicconatti, L. Lenzini, and E. Mingozi. Quality of service support in IEEE 802.16 networks. *IEEE Networks*, 20(2):50–55, Mar/Apr 2006.

[6] C. Eklund, R. Marks, K. Stenwood, and S. Wang. IEEE standard 802.16: a technical overview of the Wireless MAN air interface for broadband wireless access. *IEEE Communications*, 40(6):98–107, Jun 2002.

[7] Min-Seok Kang and Jaeshin Jang. Performance evaluation of IEEE 802.16d ARQ algorithms with NS-2 simulator. In *IEEE Asia-Pacific Conference on Communications*, pages 1–5, Aug 2006.

[8] H. Martikainen, A. Sayenko, O. Alanen, and V. Tykhomyrov. Optimal MAC PDU size in IEEE 802.16. In *4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks*, pages 66–71, Feb 2008.

[9] S. Perera and H. Sirisena. Contention based negative feedback ARQ for VoIP services in IEEE 802.16 networks. In *4th IEEE International Conference on Networks*, volume 2, pages 1–6, Sep 2006.

[10] A. Sayenko, O. Alanen, and T. Hämäläinen. Adaptive contention resolution parameters for the IEEE 802.16 networks. In *International Conference on Heterogeneous Networking for Quality, Reliability, Security and Robustness*, Aug 2007.

[11] A. Sayenko, O. Alanen, and T. Hämäläinen. ARQ aware scheduling for the IEEE 802.16 base station. In *IEEE International Conference on Communication*, 2008. accepted for publication.

[12] A. Sayenko, O. Alanen, and T. Hämäläinen. Scheduling solution for the IEEE 802.16 base station. *Computer Networks*, 52:96–115, 2008.

[13] A. Sayenko, O. Alanen, J. Karhula, and T. Hämäläinen. Ensuring the QoS requirements in 802.16 scheduling. In *The 9th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 108–117, Oct 2006.

[14] V. Tykhomyrov, A. Sayenko, H. Martikainen, O. Alanen, and T. Hämäläinen. On ARQ feedback intensity of the IEEE 802.16 ARQ mechanism. In *International Conference on Telecommunications*, Jun 2008. accepted for publication.

Vitaliy Tykhomyrov has obtained the M.Sc. degree from Kharkov National University of RadioElectronics(Ukraine) in 2006. Currently, he is a Ph.D. student at the University of Jyväskylä. His current postgraduate research interests include Quality-of-Service in wireless networks, and in particular IEEE 802.16.

Alexander Sayenko has obtained the B.Sc. degree from the Kharkov State University of RadioElectronics (Ukraine) in 2001. He has obtained the M.Sc. degree from the University of Jyväskylä (Finland) and the Ph.D. degree from the same university in 2002 and 2005, respectively. Currently, he works for the Nokia Research Center, where he is responsible for the resource and power management solutions. His research interests are QoS, resource management and scheduling in the wireless networks.

Henrik Martikainen is a Ph.D. student at University of Jyväskylä, Finland. He received his M.Sc. in Computer Sciences from University of Jyväskylä in 2006. Since that he has been studying IEEE 802.16 MAC level performance and optimization.

Olli Alanen is a researcher working in University of Jyväskylä, Finland. He has been studying QoS issues in IEEE 802.16 and IP based networks for the past years. He received his M.Sc. in Computer Sciences from University of Jyväskylä in 2004 and the Ph.D. degree from the same university in 2007.