Simulation of 3G DCHs Supporting TCP Traffic: Design, Experiments and Insights on Parameter Tuning

Juan J. Alcaraz, Gaspar Pedreño, Fernando Cerdán and Joan García-Haro
Department of Information Technologies and Communications
Technical University of Cartagena
Plaza del Hospital, 1, 30202 Cartagena, SPAIN
34 968326544

{juan.alcaraz, gaspar.pedreno, fernando.cerdan, joang.haro}@upct.es

ABSTRACT

This paper describes a simulator of dedicated channels in 3G radio access networks, with TCP traffic. The design principles of this simulator and the experiments performed in it are explained in depth. We provide specific implementation details of RLC functions in OMNeT++. The main objective of the simulator is RLC parameter configuration. Although many previous works have addressed this issue, our approach is different and more general: we investigate how the conditions of the environment (channel data rate, frame error ratio, Doppler frequency, etc) affect the proper adjustment of each parameter. We disclose the interactions between these conditions and the functionalities of RLC, achieving a methodology for robust RLC configuration in terms of suitable operation under changes in the environment.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless Communication.
D.1.5 [**Object-oriented Programming**]

General Terms

Design

Keywords

Simulation, 3G, RLC, TCP, OMNeT++, parameter setting.

1. INTRODUCTION

In 3G radio access networks, link layer communication between the user terminal and the Radio Network Controller (RNC) is performed by the Radio Link Layer (RLC) protocol, specified by the 3GPP in [1]. Because TCP performance decreases on wireless channels due to frequent packet losses, when a 3G dedicated channel (DCH) supports TCP traffic, the RLC entity of that DCH should operate in Acknowledged Mode (RLC AM). RLC AM comprises an error recovery mechanism based on a relatively complex selective repeat algorithm. This complexity is to some degree related to the great number of configurable parameters required to adjust RLC operation. Interactions between RLC and TCP, as well as the parameter configuration issues of both protocols have been a subject of extensive research in the last years, see [2] and the references therein. Figure 1 shows a diagram of the system under study.

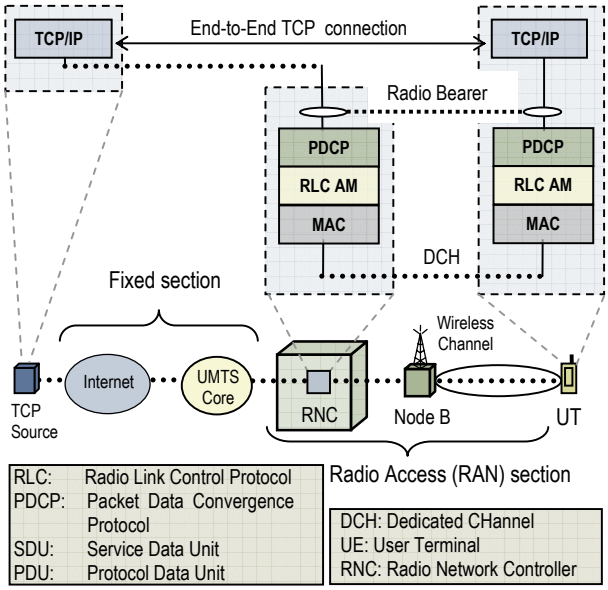


Figure 1. Diagram of the simulated system.

Concerning simulation issues, S. Gurtov and S. Floyd in [3], addressed several aspects of simulator design for the performance evaluation of TCP connections over wireless networks, which are adapted to our simulator. However, this paper addresses more specific aspects related to 3G networks and, in particular, RLC implementation details in OMNeT++ [4] and the design of experiments aimed at accurately adjust RLC parameters.

This latter issue deserves special attention. The work presented in this paper takes the results of previous works as a starting point, and establishes an objective that goes further than these previous works. The goal is not only to configure RLC parameters properly, but to find a valid configuration for a wide range of situations, e.g. frame error ratio (FER), Doppler frequency, radio bearer bit-rate, etc. These characteristics are highly variable in real wireless networks, and to the best of our knowledge, previous

OMNET++ 2008, March 03, Marseille, France

DOI 10.4108/ICST.SIMUTOOLS2008.3057

works have considered relatively static conditions regarding these parameters. Our approach consists of systematically changing these characteristics, which, in addition, allows us to disclose their influence on the adjustment of each RLC parameter.

The following list summarizes the contributions of this paper:

- We describe how previous recommendations about wireless systems simulation are adapted to the design of an RLC / TCP simulator in OMNeT++ and to the design of experiments and simulation scenarios aimed at RLC configuration.
- The implementation of several functions of RLC in OMNeT++ is addressed in detail: segmentation, concatenation and reassemble. Implementing these functions is challenging because OMNeT++ does not provide any methodology for message segmentation.
- We present detailed results about the configuration of one of the most important RLC features: the control of signalling overhead, showing its influence on performance and its interaction with the Doppler frequency of the channel.
- We summarize the interactions between RLC functions and environmental parameters, obtained by means of our simulation-based configuration methodology.

The rest of the paper is organized as follows: Section 2 contains a brief overview of RLC operation. Section 3 states the main objectives of the simulator and describes its design principles. Section 4 introduces the simulator structure and provides relevant implementation details. Section 5 explains the simulation methodology with emphasis on parameter setting. Section 6 presents the results concerning the control status mechanisms and summarizes the results of the experiments. The paper finishes in Section 7 with the conclusions and future research lines.

2. RLC OVERVIEW

RLC AM is basically a Selective Repeat (SR) ARQ link layer protocol, where the flow control is based on a sliding window mechanism. Packets from higher layers (Service Data Units, SDUs) are segmented and concatenated into link layer frames (Packet Data Units, PDUs) for its transmission over the radio link. The receiver side detects missing PDUs based on the error control field or detecting gaps on the frame sequence numbers. The RLC receiver side requests for retransmissions by sending back a bitmap report, named status PDU, acknowledging PDUs (ACK) and informing about lost PDUs (NACK). Upon the reception of a status message, the sender can advance its transmission window, if one or more in-sequence frames are acknowledged, so new PDUs can be sent. If there are NACKs in the status, the sender retransmits the missing PDUs giving them priority over new ones.

A status message is issued either when the receiver is polled by the sender or self-triggered. At the sender side, a polling request is made by marking the poll bit in the header of a PDU. The poll request can be triggered by seven configurable mechanisms. Two of them (*Timer Based Polling* and *Poll Timer*) are considered recurrent, i.e. are based on timers. *Timer Based Polling* consists of a periodic timer which expiration triggers a poll. *Poll Timer*, also driven by a periodic timer that is only started when a PDU with the poll bit is sent. The aim of this timer is the retransmission of the poll in case of a packet loss in the link, so the timer is stopped upon receiving a status message. The rest of the mechanisms are "one-shot" polling methods, i.e. they are triggered when the sender is in a particular state, e.g. if it is sending the last PDU in retransmission buffer, of if it has sent a certain amount of SDUs.

The receiver side may transmit a status PDU after the reception of a poll bit or when a status trigger is active in the receiver. There are a periodic status trigger and a "one-shot" trigger. Because the status frames make the sender window advance, the receptions of status frames should be very frequent, and therefore several mechanisms should operate simultaneously in both sides of the link. The drawback is the signaling overhead that this may cause. In order to control this overhead, there is an important control scheme in the receiver side, the *Timer Status Prohibit*.

The RLC protocol comprises two methods for SDU discarding. One is based on the waiting time of the SDUs in the transmission buffer, the other is based on the maximum number of retransmissions that a PDU can experience.

3. SPECIFICATION OF THE SIMULATOR

3.1 Objectives of the Simulator

The main objective of the simulator is to evaluate end-to-end performance of TCP connections comprising a 3G link. This performance evaluation has itself the following objectives: 1) to find how to best configure the parameters of the protocols involved, considering cross-layer interaction, 2) to provide support in the developing of cross-layer mechanisms aimed to improve the overall performance of data connections over the radio access system. It must be stated that the introduction of High Speed Downlink Packet Access (HSDPA) does not cause the withdrawal of the RLC layer. In fact, the new MAC-hs layer is placed just below the existing RLC layer in the HSDPA protocol stack. In this new context, it is of special interest to provide insight about the possible cross-layer interactions between RLC and MAC-hs.

3.2 Design Principles

In the designing and developing of the simulator, many results and recommendations form previous works were taken into account. In this sense, it is of special relevance the work done by A. Gurtov and S. Floyd in [3], where the authors expose and justify recommendations about how to properly model and simulate transport protocols over radio links. The design principles and simulation guidelines suggested in [3] were adopted in the design of our simulator and in the planning of the simulations. Here, we summarize some of the most relevant ideas, regarding different aspects of the simulation.

- Topology: The most usual scenario in TCP transport over 3G is a terminal connected to a data server located in a wired network. Therefore, only one end of the connection (the one corresponding to the end user) has a wireless 3G link. In terms of multiplexing, a 3G link is expected to multiplex a number of TCP connections ranging from 1 to 4.
- Traffic: The typical connection is asymmetric, with a heavier traffic load in the downlink direction.
- Performance measures: At the transport level, two performance values are of special importance: goodput and delay. The goodput is defined as the amount of data correctly received at the receiver side per unit of time. The goodput is a better measurement than the throughput, because it reflects

the net rate perceived by the user. The delay is the average time it takes a packet to propagate from the sender to the receiver.

- Bandwidth: The most frequent situation is the bandwidth of the radio link to be the lowest of the whole connection, and therefore to constitute the bottleneck of the connection. Bandwidth variations in the 3G link are usual because of radio resource management protocols and should be considered.
- Error model: The classical Gilbert model [4], based on a two state Markov model, is valid to model the bursty nature of errors in the wireless channel when the objective is to evaluate the performance at higher layers.

4. IMPLEMENTATION

4.1 Structure

The topology of the simulated network consists of one or several TCP connections sharing a 3G link (radio bearer), as explained in the previous section, and shown in Figure 1. Each connection is thus divided into two sections, the Radio Access Network (RAN) section and the wired network section. The RAN is the section between the RNC and the mobile terminal, and comprises two logical interfaces: The link between the RNC and the Node B, defined as the Iub interface, and the wireless link between the Node B and the mobile terminal, defined as the Uu interface or air interface. In this latter one is where frame losses take place because of propagation errors. The Iub interface is based either in ATM or IP. The simulator does not implement this subdivision of the radio bearer, because from the RLC point of view, the underlying infrastructure is perceived as a logical link with a round trip time delay (RTT) and a frame error ratio (FER). The data link between two RLC entities is named logical channel and is provided by the MAC layer.

However, it is known that the Iub interface may experience diverse situations (e.g. traffic congestion or path restoration procedures). The Frame Protocol, which is in charge of controlling synchronization related issues at the Iub, has been studied separately from the RLC – TCP interaction. For this task we have also developed Frame Protocol modules in OMNeT++, and some of the results obtained can be seen in [6]. Although the details of that simulator are outside the scope of this paper, the experiments carried out confirm that, as far as RLC is concerned, we can model any situation at the Iub Interface as a variation of the RTT and FER of the logical channel.

The wired network section comprises the UMTS core network and the Internet section required to connect the TCP servers with the UMTS network. Because the radio bearer is generally considered to be the bottleneck of the connection, the wired network section is modeled as a reliable single link with configurable delay and bandwidth (generally higher than the DCH bandwidth).

Figure 2 shows the connections of the modules in our simulator. The following list contains an explanation of each module.

- TCP modules: For this simulator specific TCP modules were implemented for the sender and receiver sides. For each side, the base class is TCP Reno, and two newer versions of TCP are sub classed from it: TCP New Reno and TCP SACK.
- PDCP: This module implements the Packet Data Convergence Protocol (PDCP), located just above RLC in the protocol stack. PDCP is defined in the TS 25.323 specification [7], and multiplexes data flows that are carried over a 3G link. PDCP is responsible of packet header compression for TCP packets.
- RLC: The main features of RLC are described in Section 2.
- MAC: In dedicated channels, MAC assembles the transport blocks (TB) from the frames received form an RLC entity. MAC is in charge of controlling the instantaneous bandwidth of RLC by setting the maximum number of RLC frames that can be gathered together into a TB. These TBs are transferred to lower layers in synchronized time slots named Transmission Time Instants (TTI).
- Radio Channel: The wireless channel generates error bursts according to the Gilbert channel model developed by Zorzi in [8]. This model considers the normalized Doppler frequency, $f_D \cdot t_{TTI}$, where f_D is the Doppler frequency in Hertzs and t_{TTI} is the duration of the radio time slot in seconds. The model uses $f_D \cdot t_{TTI}$ and the average Frame Error Ratio (FER) to compute the transition probabilities of the Markov chain. These transition probabilities determine the average length of error bursts. Lower $f_D \cdot t_{TTI}$ causes longer bursts of errors.

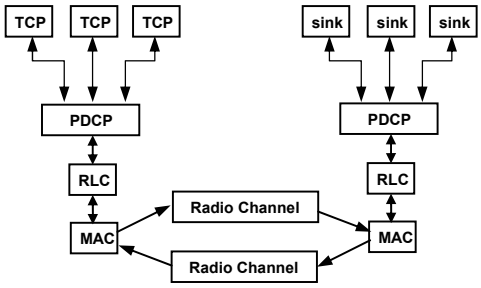


Figure 2. Modules and topology of the simulator.

4.2 Programming Issues

Considering the RLC functions, three of them are of special interest regarding its implementation in C++ within the OMNeT++ environment: the segmentation and concatenation of upper layer packets (SDU, Service Data Units) at the RLC sender side and, in the RLC receiver side, the reassembly of SDUs. The interest of these functions comes from the fact that, although OMNeT++ provides a method to encapsulate several messages into a single message, there are no pre-existing methods to segment a message so that it can be reassembled into the original *cMessage* object by a module receiving these segments. The implementation of these functions relies on the data structures defined for the RLC module. These structures are also necessary for other mechanisms like sliding window flow control, selective retransmission, packet discarding and in-sequence delivery.

4.2.1 Data structures

PDU_Packet: is implemented in a class derived from *cMessage* by means of a message definition file, shown (in a simplified way) in Figure 3, with customized functions for encapsulating messages (see [4] for further details on this procedure). The total number of SDU objects (*SDU_packet*) stored in the PDU is indicated in the attribute *ending_SDUs*. For each SDU or SDU segment encapsulated, the sequence number and the segment length are stored in the corresponding arrays (*SDU_SN* and *Seg_sizes*). The total lengths of the SDUs from which the segments are extracted, are included in the *SDU_lengths* array.

```
PDU_Packet
{properties:
        customize=true;
fields:
        unsigned int SN;
        int ending_SDUs;
        unsigned int SDU_SN[];
        int Seg_sizes[];
        int SDU_lengths[];
}
```

Figure 3. Definition of the PDU packet.

PDU_Array: PDUs are stored in an array until they are acknowledged by the receiver side. The arrival of SDUs from the PDCP module generates the creation of new PDU objects that are stored in the PDU array. When the RLC module has to send new or old PDUs, it gets them form the PDU array.

SDU_Reg_s (at the sender side): When an SDU is segmented into one or several PDUs, the sequence number of these PDUs is stored into the registry of this SDU. When all the PDUs are acknowledged, the SDU is fully acknowledged and is deleted. The SDU is created at the PDCP module, and can encapsulate any packet coming from upper layers. The PDCP module at the receiver side extracts the packet from the SDU object. This eases the integration of the simulator with any upper layer module.

SDU_Reg_r (at the receiver side): This structure stores the number of bits that the receiver side has correctly received for a single SDU. The SDU is extracted from the PDUs encapsulating the last segment of the SDU, and stored in the SDU registry, but the SDU is only available to be passed to the upper layer when all the bits expected for that SDU are received.

SDU_Buffer (at the receiver side): The SDU registries are stored in an array according to the sequence number of the SDUs. The SDUs are assigned a sequence number by the PDCP module. If the delivery is not in-sequence, the RLC receiver side checks if the next expected SDU has all its bits received to extract it from its SDU registry and send it to the upper layer.

4.2.2 Segmentation and concatenation of SDUs

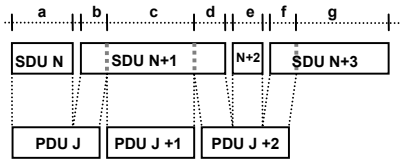
These processes are implemented in a single function: *Segment_Concat(SDU_Packet *sdu)*. The RLC module has two attributes useful for this task, the sequence number of the PDU which is currently being filled with SDU segments (*PDU_SN*) and the free space in bits of this PDU (*PDU_freespace*). The simplified code of the function is shown in Figure 4. Many details of the implementation are hidden for the sake of clarity.

The function *Encapsulate_SDU* stores the length of the encapsulated segment (first argument) and the total length of the segmented SDU (second argument), into the corresponding arrays of the *PDU_packet*. The function *addToPaquetes* stores the *SDU_Packet* object into an internal array named *Packets*. The function *Increase_Ending_SDUs* increases the attribute *ending_SDUs* of the current *PDU_Packet* object. The function *Update_PDU_freespace* updates the amount of remaining unused payload bits in the current *PDU_Packet* after inserting a segment.

The underlying idea in the function *Segment_Concat* is that each SDU segment is represented by two integers: the SDU sequence number and the amount of bits of the segment. When the segment is the last one, or the whole SDU fits inside the PDU, then the real SDU object is stored in the PDU, into the *Packets* array implemented in the *PDU_Packet* class. The remaining bits of the PDU payload (*PDU_freespace*) are used to store further segments in successive calls to *Segment_Concat*, except if the current *PDU_Packet* is sent to the lower layer. Figure 5 illustrates with a diagram the operation of *Segment_Concat*.

```
void RLC_AM::Segment_Concat(SDU_Packet*sdu){
int Seg_length = SDU_length = sdu->length();
if (PDU_freespace>=Seg_length){
   ((PDU_Packet*)PDU_Array[PDU_SN])
   ->addToPackets(sdu);
   Encapsulate_SDU(Seg_length, SDU_length);
   Increase_Ending_SDUs();
   Update_PDU_freespace(Seg_length);
   }
else{
   Encapsulate_SDU(PDU_freespace,
   SDU_length);
   Seg_length-=PDU_freespace;
   PDU_SN+=1;
   Create_PDU(PDU_SN);
   while (Seg_length>PDU_freespace) {
      Encapsulate_SDU(PDU_freespace,
      SDU_length);
      Seg_length-=PDU_freespace;
      PDU_SN+=1;
      Create_PDU(PDU_SN);
      }
   ((PDU_Packet*)PDU_Array[PDU_SN])
   ->addToPaquetes(sdu);
   Encapsulate_SDU(Seg_length, SDU_length);
   }
   Increase_Ending_SDUs();
}
```

Figure 4. C++ code of the *Segment_Concat* function.



PDU J	
SDU_SN	[N, N+1]
Seg_sizes	[a, b]
SDU_lengths	[a, b+c+d]
Ending SDUs	1
Objects stored	SDU N

PDU J + 1	
SDU_SN	[N+1]
Seg_sizes	[c]
SDU_lengths	[b+c+d]
Ending SDUs	0
Objects stored	-

PDU J + 2	
SDU_SN	[N+1, N+2, N+3]
Seg_sizes	[d, e, f]
SDU_lengths	[b+c+d, e, f+g]
Ending SDUs	2
Objects stored	SDU N+1, SDU N+2

Figure 5. Example of segmentation and concatenation.

4.2.3 Reassembly

The function for SDU reassembly, shown in Figure 6, is only invoked when a PDU is received for the first time, i.e. with a sequence number that has not been previously received. In the

code presented many details are omitted again, in order to focus only in the part of the function dealing with SDU reassembling.

```
int RLC_AM::Reassemble_SDU(PDU_Packet *PDU)
unsigned int N_seg = PDU-
>getSDU_SNArraySize();
int SDU_Ending = PDU->getEnding_SDUs();
for(unsigned int i=0; i< N_seg; i++){
   SDU_seq = PDU->getSDU_SN(i);
   Seg_size = PDU->getSeg_sizes (i);
   SDU_length = PDU->getSDU_length(i);
   if (SDU_Buffer.exist(SDU_seq)){
      int bits =
      ((SDU_Reg_r*)SDU_Buffer[SDU_seq])
      ->getBits();
      int bits_left = bits - Seg_size;
      ((SDU_Reg*)SDU_Buffer[SDU_seq])
      ->setBits(bits_left);
      }
   else{
      SDU_Reg* SDU = new SDU_Reg_r;
      SDU->setSDU_SN(SDU_seq);
      SDU->setLength(SDU_length);
      int bits_left = SDU_length - Seg_size;
      SDU->setBits(bits_left);
      SDU_Buffer.addAt(SDU_seq,SDU);
      }
if (SDU_Ending>0) {
   SDU_Packet* sdu= (SDU_Packet*)PDU-
   >dropFront();
   ((SDU_Reg*)SDU_Buffer[SDU_seq])
   ->encapsulate(sdu);
   SDU_Ending--;
   }
}
```

Figure 6. C++ code of the *Reassemble* function.

The *Reassemble_SDU* function operates in the following way. For each PDU received, the function obtains the number of segments and SDU objects (*SDU_Packet*) stored in this PDU (*N_seg*). Each segment contains the sequence number (*SDU_seq*), the segment size (*Seg_size*), and the total size (*SDU_length*) of the SDU. The function processes each segment separately. If a previous PDU containing segments of the same SDU has been already received, the corresponding *SDU_Reg_r* is updated, reducing the number of pending bits to be received for this SDU (*bits_left*). If the segment is the first to be received of this particular SDU, a new *SDU_Reg_r* object is created. If there are SDUs objects actually encapsulated in the PDU, it means that the processed segment is the last one or corresponds to a whole SDU (see Figure 5). Therefore the SDU object is extracted with *dropFront* from the internal array (*Packets*) of the PDU.

5. DESIGN OF EXPERIMENTS

5.1 Parameter Setting

As explained in Section 1, our approach to RLC parameter adjustment consists of evaluating the impact of each environmental parameter on each RLC function. The parameters that can potentially affect configuration decisions on RLC are:

- The nominal rate of the radio bearer
- The RTT of the radio access section
- The RTT of the fixed section
- The FER of the radio access section
- The Doppler frequency of the radio channel.

In order to consider any situation in a real network, the simulations are done for several values of these parameters. These values were chosen by means of an extensive study of previous works, so that the range defined includes both typical and extreme values for each parameter. In the following list, we show some of the references where the authors assign values to these environmental parameters.

- The RTT of the radio access section: M. Meyer [11] considers 80 ms, Pentikousis [12] 200 ms, Mutter [13] and Cano-García [14] 45 ms, Bestak [9], Chen [15] and Fukuda [16] 60 ms, and Rossi [17] 100 ms.
- The RTT of the fixed section: Chakravorty1 [18] considers this RTT variable from 200 ms up to 1 s, Meyer [11] from 25 ms up to 1 s, Pentikousis [12] from 50 ms up to 1 s.
- The FER of the radio access section: Meyer [11] uses FER = 10%, Fukuda [16], from 0,1% up to 10 %, Bestak [9] and Xu [19] from 0% up to 30 %,and Chen [15] from 0% to 20%.
- The Doppler frequency of the radio channel. This parameter is usually expressed as normalized Doppler frequency, $f_D \cdot t_{TTI}$. As an example of works using $f_D \cdot t_{TTI}$, we have the works of Chockalingam [20] and Rossi [17] where $f_D \cdot t_{TTI}$ ranges from 0.01 up to 1, Chiasserini [21] from 0.02 up to 0.2. In the work of Mukhtar [22], f_D ranges from 0.1 up to 1 Hz.

Regarding the nominal rate of the radio bearer, it is usual to consider a nominal rate of 384 kbit/s in 3G access networks. However some works e.g. Meyer [11] considers a rate of 128 kbit/s. Considering the the bandwidth changes caused by RRC algorithms, we have used several specified rates. Table 1 shows the values assigned to each parameter in simulations.

Table 1. Parameters of the simulated environment

Paramter	Values
DCH rate (Kbit/s)	64, 128, 256, 384
$f_D \cdot t_{TTI}$	0.01, 0.1, 1.0
FER (%)	10, 15, 20
RTT radio access (ms)	50, 100, 200
RTT fixed section (ms)	50, 200, 400, 800

Although the downlink buffer size of the RLC is not technically an RLC parameter, its size has a great impact on the performance of TCP flows, highlighted also in [2], [3] and [9], because the recovery mechanisms at RLC may cause overbuffering and even buffer overflow. Chakravorty in [18] and [23] uses 120 Kbytes, Mutter [13] uses 50 kBytes, Fukuda [16] from 5 to 70 packets of 1500 bytes, Hu [24] uses a buffer of 64 packets of 1500 bytes, Bestak [9] from 20 to 40 packets of 600 bytes, and Chan [25] from 5 to 45 packets of 1500 bytes. We evaluate buffer sizes ranging from 20 to 150 Kbytes in steps of 10 Kbytes.

Many previous works have dealt with issues concerning RLC configuration. These works were taken as a basis to obtain a reference RLC configuration. For example, regarding the polling function, which is addressed in Section 6, it is generally recommended to combine “one-shot” triggers with “recurrent” triggers, e.g. [2], [13], [17], [19]. Using these guidelines, we obtained, by means of successive simulation tests, the RLC configuration summarized in Table 2, for a reference scenario (DCH rate = 384 Kbit/s, $f_D \cdot t_{TTI}$ = 0.01, FER = 10 %, RTT RAN section = 50 ms and RTT fixed section = 200 ms). The configuration of TCP sources is also shown in Table 2.

5.2 Simulation Scenario

The simulated scenario is based in the topology shown in Figure 1 and Figure 2, where the user downloads a long file from one or several TCP servers, so that every TCP source in the system has always available data to transfer (long-lived TCP flows). In a mobile network, it is very usual for users to download files like mp3 songs, images, videos or games. In addition, many data applications may eventually have an FTP-like behaviour, e.g. accessing to large web pages, like those containing flash contents, or receiving large e-mail messages. In this kind of scenario, two situations are considered: a single TCP connection or several TCP connections multiplexed over a DCH.

Table 2. Initial configuration of RLC and TCP protocols

3G link parameters	Setting
PDU payload size	320 bits
TTI (Transm. Time Interval)	10 ms
Transmission window	2047 PDUs
Timer based discard	off
maxDAT discard	on
maxDAT	20
In-order-delivery	true
Buffer Size	150 Bytes
One-shot poll triggers	
Last PDU in buffer Poll	on
Last retransmitted PDU Poll	on
Poll window	50 %
Recurrent poll triggers	
Timer Based Polling	off
Poll Timer	60 ms
Status control in receiver side	
Status Prohibit Timer	200 ms
Missing PDU detection	on

TCP Reno parameters	
Maximum TCP/IP packet size	1500 bytes
Maximum allowed window	64 kbytes
Initial window	1
Wired Network Round Trip Delay	200 ms

5.3 Experiments and Measurements

The experiments consist of changing the value of an RLC parameter, taking Table 2 as the starting point, over the whole range of possible values according to the RRC [26] for each configuration of the simulation scenario. This configuration is defined by a given combination of environmental parameters, so that each experiment is repeated for each value of Table 1. RLC parameters are classified into the following functions:

- Polling function. The most important parameter for this function is the value of the *Status Prohibit Timer*.
- Flow control, set by the Window Size parameter.
- Persistence, regulated by the maximum number of retransmissions prior a packet discard (*MaxDAT*).
- Buffering, obviously related to the buffer size.
- In-Sequence delivery, determined by a single Boolean parameter which turns this option on or off.

The performance figures obtained depend on the function evaluated. In general three measures are always obtained, the goodput, the end-to-end delay and the packet loss ratio at the RLC layer. Some particular functions need additional performance measures. An important performance measure for the polling function is the signalling ratio (*status/PDU*), defined as the number of status frames sent divided by the number of PDU data frames sent. S*tatus/PDU* , obtained at the RLC level, gives an idea of the efficiency of the configuration in terms of signalling load. When studying the buffering process it is also helpful to know both the average buffer occupancy and the maximum buffer occupancy in order to accurately configure the buffer size. When studying the effects of delivering packets in sequence or out of sequence to upper layer some measures done at the TCP level are also of importance, like the number of timeouts or the number of fast retransmit processes experienced by the sender side. Each performance figure is obtained averaging the results of 20 simulation runs of 120 simulated seconds. The confidence intervals have been computed for a 95 % confidence degree.

6. RESULTS

6.1 Status Control

The status control is part of the polling function, which pursues three goals: to increase throughput, to reduce end-to-end delay and to reduce energy consumption. These three objectives were previously identified in [13], [17], [21] and [27]. While a frequent generation of status messages make the sender window advance faster (except in some situations described later), which helps to fulfil the first two objectives, if the rate of status generation is higher than needed, the terminal wastes radio resources as well as the energy of its battery.

The control of excessive status generation is handled by the *Status Prohibit Timer* at the receiver side. If the polling and status generation mechanisms generate status signalling at a sufficiently high rate, which is recommended, this mechanism is the one which ultimately controls the advance of the sender window. The question arises about how to best configure the timer duration, and how the characteristics of the environment should be taken into account to adjust this parameter.

Figure 7 shows the performance figures of the possible timer values. For each performance measure (Goodput, Delay and *Status/PDU*) we can see three curves. Each one corresponds to a different normalized Doppler frequency value.

Several conclusions can be drawn from Figure 7. Obviously, if the timer is too large, the general performance decays, due to the slow advance of the RLC sender window. If the timer is very small, the signalling overhead is higher. The most interesting thing, however, is the fact that, at lower levels of the timer, the goodput also decays. This is due to redundant transmissions at the RLC layer. Clearly, if the receiver side generates more than a status message in an RTT period (50 ms), more than one retransmission of the same PDU can be generated. This obviously reduces the goodput at higher layers. Moreover, what we can see in Figure 7 (a) is that, for lower Doppler frequencies, the maximum goodput is achieved at higher timer values. With $f_D \cdot t_{TTI} = 0.01$, the goodput is maximum between 200 ms and 350 ms. The reason is that, at lower Doppler frequencies, the error bursts last longer. According to the Zorzi model [8], for $f_D \cdot t_{TTI} = 0.01$ and FER = 10%, the average burst duration ($E[T_B]$) is 13,6 time slots (136 ms), which is longer than the RTT period. In this case, the retransmission periods will last, al least, as much as the error bursts that caused the frame losses. Therefore, if the timer duration is smaller than $E[T_B]$, a status message can be generated during the retransmission period, asking for PDUs which

retransmission is already scheduled, thus causing redundant retransmissions.

The simulation results show that an appropriate configuration of the *Status Prohibit Time* is the average duration of the error burst in the worst expected situation (e.g. $f_D \cdot t_{TTI} = 0.01$) adding the RTT of the RAN section, which should be periodically measured.

Focusing in the signalling efficiency, Figure 7 (b) shows that there is a point where the curve reaches its minimum, from which the *Status/PDU* ratio reaches higher values for longer timer durations. The reason is that the goodput decays so rapidly when the timer increases, that the number of PDUs decreases faster than the number of status messages.

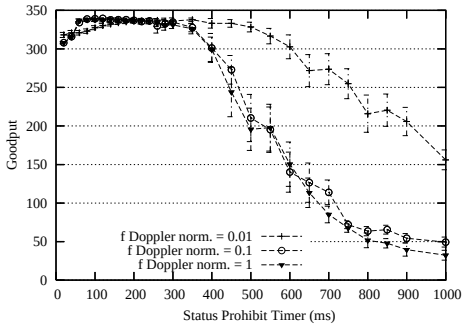


Figure 7 (a). Goodput vs. *Status Prohibit Timer*.

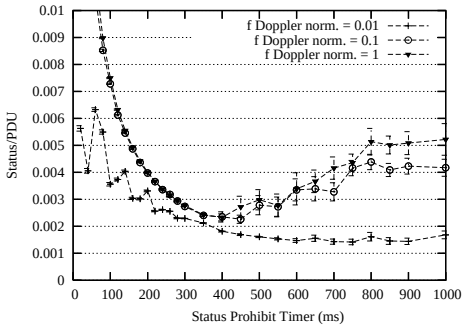


Figure 7 (b). Signalling Efficiency vs. *Status Prohibit Timer*.

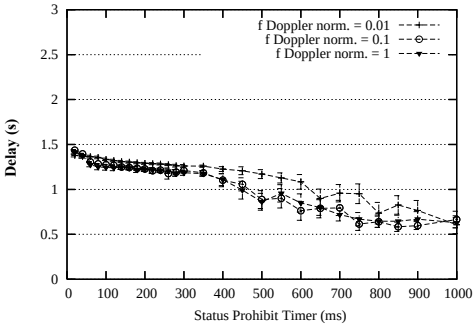


Figure 7 (c). Delay vs. *Status Prohibit Timer*.

Finally, Figure 7 (c) shows the relation between the delay and the timer duration. In the range of timer values were redundant transmission take place, the delay decays as the timer increases because, when redundant retransmissions are reduced, fewer packets are buffered. When the goodput is at its higher values, the delay is almost constant. As soon as the goodput decays, also does the delay, because less TCP packets arrive at the RLC module, reducing the buffering time at RLC, which is the most influential factor in end-to-end delay.

6.2 Interactions

Similarly to the results shown in previous section, each parameter was evaluated under different characteristics of the scenario (nominal rate, FER, f_D, etc). As a result, we have disclosed, for each RLC parameter, what environmental parameters has impact on its adjustment, in the same way that, in Section 6.1, f_D showed to have influence on *Status Prohibit Timer* configuration. Table 3 summarizes the interactions detected.

Flow Control: The RTT of the RAN and the DCH data rate determine the number of PDUs "in flight" between two RLC entities, and therefore the number of unacknowledged PDUs in the sender window. The window should be larger than this number in order to achieve the maximum PDU delivery rate.

Persistence: Obviously, more errors in the channel require more retransmissions. In addition, DCHs with higher data rates also benefit from a higher persistence.

Buffer Size: The buffer size should be able to accommodate sudden increments in the buffer occupancy. These increments are higher for longer error bursts in the channel. A longer RTT causes a larger TCP window, which also demands larger RLC buffers.

In-Sequence delivery: For this function it was checked that every environmental parameter that increases the number of lost PDUs in the window and makes its distribution more random (shorter error bursts), also increases the number of SDUs reassembled out of sequence, and therefore increases the degradation of TCP performance in case of not activating "in-sequence delivery".

Table 3. Interactions between RLC procedures and parameters of the simulated environment

	DCH nominal rate	f_D	FER	RTT radio section	RTT fixed section
Polling function		X		X	
Flow control	X			X	
Persistence	X		X		
Buffer size		X			X
In-sequence delivery	X	X	X	X	

7. CONCLUSIONS

We have presented an OMNeT++ simulator for TCP connections over RLC. The design of the simulator and the experiments performed in it are based on an comprehensive study of previous works. Regarding programming issues, we provide extensive details on the implementation of three RLC functionalities: segmentation, concatenation and reassembly of SDUs.

The main objective is RLC parameter configuration, which has been addressed with a different approach respect previous works. We evaluate the performance for all the allowed values of each RLC parameter, and repeat the simulations changing the conditions of the environment (FER, Doppler frequency, data rate, etc). This let us configure RLC functions to operate properly under many possible situations of a real network. We illustrate it showing how the Doppler frequency of the wireless channel affects the suitable configuration of the *Status Prohibit Timer*, the timer which controls the signalling overhead in retransmission procedures. Finally, all the interactions between the parameters of the environment and RLC functions are summarized.

The simulator tool presented will be upgraded with the integration of HSDPA modules in order to detect the cross-layer effects between RLC and the MAC-hs layer introduced by HSDPA.

8. ACKNOWLEDGMENTS

This project has been supported by the Spanish Research Council with the CON-PARTE project (TEC2007-67966-C03-01/TCM).

9. REFERENCES

[1] 3GPP TS 25.322, 2005. Radio Link Control (RLC) protocol specification, v. 6.4.0 (Jun. 2005).

[2] Alcaraz, J. J., Cerdán, F., and García-Haro, J. 2006. Optimizing TCP and RLC Interaction in the UMTS Radio Access Network, IEEE Network, 20, 2 (Mar. 2006), 56-64.

[3] Gurtov, A., and Floyd, S. 2004. Modeling Wireless Links for Transport Protocols, ACM SIGCOMM Computer Communication Review. 34, 2 (Apr. 2004), 85-96.

[4] OMNeT++ user manual. ver. 3.2. *OMNeT++ Community Site*, http://www.OMNeTpp.org/

[5] Gilbert, E. N. 1960. Capacity of a Burst-Noise Channel. Bell Systems Tech. Journal. 39 (Sept. 1960), 1253-1266.

[6] Alcaraz, J. J., Pedreño, G., and Cerdán, F. A Control-Based Approach to Transport Channel Synchronization in UTRAN. IEEE Comm. Letters. 11, 7 (July 2007), 595-597.

[7] 3GPP TS 25.323 2007. Packet Data Convergence Protocol (PDCP) Specification, v6.9.0. 3GPP TS 25.323 (Oct. 2007).

[8] Zorzi, M., Rao, R. R., and Milstein, L. B. 1995. On the Accuracy of a first-order Markov Model for Data Block Transmissions on Fading Channels. In Proc. IEEE ICUPC'95 (Nov. 1995).

[9] Bestak, R., Godlewski, P., and Martins, P. 2002. RLC Buffer Occupancy When Using A TCP Connection Over UMTS. In Proc. IEEE PIMRC'02, vol. 3, (Sep. 2002).

[10] Inamura, H., et al. 2003, TCP over Second (2.5G) and Third (3G) Generation Wireless Networks. IETF RFC 3481 (Feb. 2003).

[11] Meyer, M., Sachs, J., and Holzke, M. 2003, Performance Evaluation of a TCP Proxy in WCDMA Networks, IEEE Wireless Communications. (Oct. 2003), 70-79.

[12] Pentikousis, K. 2005. Active Goodput Measurements from a Public 3G/UMTS Network, IEEE Comm. Letters, 9, 9 (Sept. 2005), 802-804.

[13] Mutter, A., Necker, M. C., and Lück, S. 2004. IP Packet Service Time Distributions in UMTS Radio Access Networks. In Proc. EUNICE 2004.

[14] Cano-García, J. M., Gonzalez-Parada, E., and Casilari, E. 2006. Experimental Analysis and Characterization of Packet Delay in UMTS Networks. In Proc. NEW2AN 2006, (May. 2006), 396-407.

[15] Chen, Y., et al. 2003. Simulation Analysis of RLC for Packet Data Services in UMTS Systems. In Proc. IEEE PIMRC 2003. vol. 1, (Sep. 2003), 926-30.

[16] Fukuda, Y., et al. 2004. Performance Evaluation of TCP under Dynamic Allocation scheme for Down-Link Transmision Rate in W-CDMA Systems. Wireless Comm. and Mobile Computing. 4, 2 (March 2004), 223-232.

[17] Rossi, M., Scaranari, L., and Zorzi, M. 2003. On the UMTS RLC Parameters Setting and their Impact on Higher Layers Performance. In Proc. EEE VTC 2003-Fall, vol. 3 (Oct. 2003), 1827- 32.

[18] Chakravorty, R., et al. 2005. Using TCP flow-Aggregation to Enhance Data Experience of Cellular Wireless Users. IEEE JSAC. 23, 6, (Jun. 2005), 1190-1204.

[19] Xu. H. et al. 2002. Performance Analysis on the Radio Link Control Protocol of UMTS System. In Proc. VTC 2002-Fall, vol. 4 (Sep. 2002), 2026 -30.

[20] Chockalingam, A., and Bao, G. 2000. Performance of TCP/RLC Protocol Stack on Correlated Fading DS-CDMA Wireless Links. IEEE Trans. on Vehicular Tech. 47, 1 (Jan. 2000), 28-33.

[21] Chiasserini, C. F., and Meo, M. 2001. Energy Efficiency of Radio Link Protocols in 3GPP Systems. In *Proc. IEEE VTC 2001*, vol. 4, (May. 2001) 2615-19.

[22] Mukhtar, R., et al. 2003. Efficient Internet Traffic Delivery over Wireless Networks. IEEE Comm. Mag. 41, 12 (Dec. 2003), 46-53.

[23] Chakravorty, R. *et al*. 2005. Optimizing Web Delivery over Wireless Links: Design, Implementation and Experiencies. *IEEE JSAC*. 23, 2, (Feb. 2005), 402-416.

[24] Hu, J., and Feng, G. 2003. Hierarchical Cache Design for Enhancing TCP over Heterogeneous Networks with Wired and Wireless Links. *IEEE Trans. on Wireless Comm*. 2, 2, (Mar. 2003), 205-217.

[25] Chan, M. C., and Ramjee, R. 2005. TCP/IP Performance over 3G Wireless Links with Rate and Delay Variation. *Wireless Networks*. 11, 1-2, (Nov. 2005), 81-97.

[26] 3GPP TS 25.331 2006, Radio Resource Control (RRC); Protocol specification, v 6.10.0. 3GPP TS 25.331 (Jun. 2006).

[27] Zhang, Q., and Su, H. 2002. Performance of UMTS Radio Link Control. In *Proc. IEEE ICC '02*, vol. 5, (May. 2002), 3346-50.