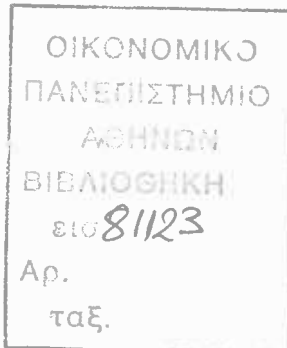




ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

POSTGRADUATE PROGRAM
IN COMPUTER SCIENCE



M.Sc. Thesis

*"Implementing and evaluating the RLC/AM protocol
of the 3GPP specification"*

Michael Makidis

Supervisor: George Xylomenos



ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ
ΚΑΤΑΛΟΓΟΣ



ATHENS, FEBRUARY 2007



Table of Contents

Table of Contents	2
Abstract	3
Acknowledgements	3
1 Introduction	4
1.1 Objectives	4
1.2 Document outline	4
2 Overview of the RLC protocol	5
2.1 Protocol Data Units (PDUs)	7
2.2 Protocol States	13
2.3 State Variables	14
2.4 Windows	16
2.5 Timers	17
3 Data transfer, polling & status reporting	20
3.1 Data transfer	20
3.2 Polling	22
3.3 Status Report Transmission	24
4 Limited Reliability of the RLC/AM	27
4.1 SDU Discard Operation Modes	27
4.2 SDU discard with explicit signaling procedure	28
5 Other procedures	31
5.1 RLC Reset procedure	31
5.2 Local Suspend function	33
5.3 Stop & Continue functions	34
5.4 Re-establishment function	34
5.5 Reconfiguration of RLC parameters by upper layers	34
6 Results	35
6.1 Simulation Setup	35
6.2 Comparison with other protocols	38
6.3 SDU discard policies	45
6.4 Conclusions	48
Appendix	50
RLC/AM features implemented	50
RLC/AM parameters	52
WAN topology diagrams	56
SDU Discard policies comparison diagrams	60
References	62
Abbreviations	64



Abstract

We implemented the RLC/AM protocol from the 3GPP specification in a simulated environment and we compared its performance over wireless links for web browsing, file transfer and continuous media distribution. We used a Uniform error model and a Two State error model to compare the protocol performance to the raw link, a simple Selective Repeat protocol and a Selective Repeat protocol with adaptive timers. We examine the behavior of the protocol and determine its applicability on each application.

Acknowledgements

I would like to thank my supervisor, Dr. George Xylomenos, for his valuable advice and never-ending patience during this project, not to mention letting me use his code infrastructure for building my extension. I would also like to thank Prof. George Polyzos for accepting the role of second supervisor.

Special thanks go to my family and friends for their support during this project.



1 Introduction

1.1 Objectives

Several of the most widely used Internet applications, like Web Browsing and File Transfer, use the TCP protocol as the transport layer protocol. With wireless networks becoming a commodity, a number of issues have been identified with TCP performance over wireless links and several solutions have been suggested [15]. One of those is the usage of a reliable link-layer protocol without TCP awareness, like the RLC.

The Radio Link Control (RLC) [1] protocol is the data link layer protocol for the Universal Mobile Telecommunications System (UMTS), the proposed successor of the popular GSM mobile networks and one of the third-generation mobile phone technologies. UMTS was designed to be a packet switching network, where data transfer and digital media (other than simple voice calls) are considered first-class citizens. As such, it supports much greater data rates compared to second-generation technologies like GSM, of up to 384 Kbps for R99 terminals or 3.6 Mbps for HSDPA terminals. This is a vast improvement compared to GSM's 9,6 Kbps.

Since the air interface is critical for a mobile network's performance, the purpose of this project was to implement and study the RLC/AM protocol used in the UMTS system in a simulated environment and compare its performance to the raw link and two Selective Repeat protocol variants, a simple and an adaptive one, which have been found to offer excellent performance for TCP applications [12]. The tests were performed using two error models and we measured the performance of web browsing, file transfer and continuous media distribution.

1.2 Document outline

In the following section we provide an overview of the acknowledged mode (AM) of the RLC protocol and we describe its PDU types, states, variables, timers and other components. In the third section we describe the data transfer procedure, the polling procedure and the status reporting procedure for positively and negatively acknowledged PDUs. In the fourth section we outline the limited reliability feature of the RLC/AM, describing the SDU Discard function and the explicit signaling procedure used to inform the Receiver of the discarded SDUs. In the fifth section we briefly discuss the remaining functions of the RLC/AM protocol, including the RLC Reset procedure. Finally, in the sixth section we discuss our results and conclude this report. An appendix at the end of the document explains the RLC features implemented and the RLC parameters used in our implementation. It also includes some performance diagrams from the sixth section for reference. At the end of this document there is a list of references and a list of abbreviations used in this report.

2 Overview of the RLC protocol

The RLC protocol is part of the data link layer of the UMTS stack. The RLC protocol has three modes: transparent mode, unacknowledged mode and acknowledged mode. The transparent mode (TM) is used when no services are required from this layer (for example, when using the UMTS channel for a voice call). The unacknowledged mode (UM) is used when the application requires a low delay channel but with better reliability than the raw link (for example, video streaming using UDP at the transport layer). This mode can perform duplicate avoidance and reordering but it does not perform retransmissions. Finally, the acknowledged mode (AM) is used when a reliable channel is required (for example, for file transfer). This mode can perform retransmissions (i.e. the RLC is an Automatic Repeat Request protocol in this mode) and it uses a sliding window like the Selective Repeat and Go-Back-N protocols. This is the mode studied in this project.

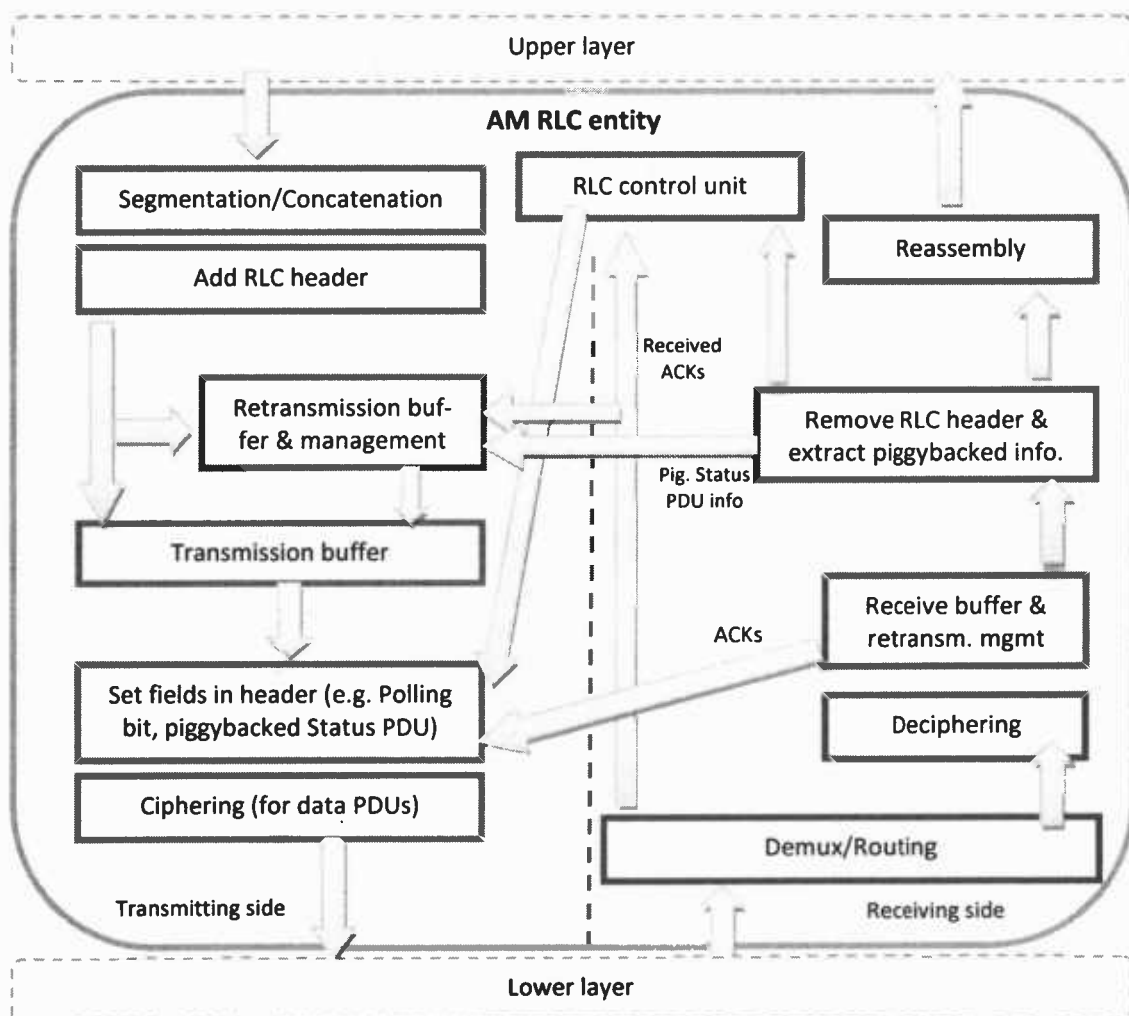


Fig. 1 Model of an acknowledged mode RLC entity

The RLC sublayer assumes that there are two AM RLC entities wishing to exchange data. Each AM RLC entity has a transmitting side (the "Sender") and a receiving side (the "Receiver") and it acts as either the Sender or the Receiver, depending on

the procedure. An AM RLC entity can be configured to use one or two logical channels to send or receive data and control PDUs. In case two logical channels are configured, data PDUs are transmitted on the first channel and control PDUs on the second channel in the uplink. Any PDU can be transmitted on any channel in the downlink. Each channel has a number of acceptable sizes for PDUs.

The RLC/AM protocol performs the following functions:

- **Transfer of user data:** The protocol supports (mostly) reliable data transfer. It can discard data that have not been successfully transmitted in a period of time or number of retransmissions. This is because it expects upper layer protocols (like TCP) to perform their own, end-to-end retransmissions as well.
- **Error correction:** The protocol detects erroneous PDUs and either asks for a retransmission or discards them altogether.
- **Sequence number check, duplicate detection and in-sequence delivery of upper layer PDUs:** The protocol detects duplicate PDUs by using sequence numbers and a PDU window. It also delivers the PDUs in-sequence. Out-of-sequence delivery is also supported.
- **Protocol error detection and recovery:** The protocol detects a number of abnormal conditions and can request retransmissions or even reset itself if all other measures fail.
- **Flow control:** The Receiver can optionally change the Sender's window size to control the flow (this function was not implemented in our simulation).
- **Segmentation and reassembly, concatenation and padding:** The SDUs received from upper layers are segmented into smaller parts (if they are larger than the available space in a PDU) and are concatenated to form PDUs of fixed size. Padding is added at the end of the PDUs if they are smaller than the indented size. These functions have not been implemented in our simulation because we needed it to be comparable to pre-existing implementations of other protocols that do not perform those functions either.
- **Ciphering:** The protocol encrypts and decrypts the data exchanged (this function was not implemented in our simulation).

The Sender uses data (AMD) PDUs to transfer the SDUs from upper layers to the Receiver (see *Data transfer*, pg. 20). Every PDU has a sequence number (in the interval $[0, 2^{12}-1]$) and the Sender and the Receiver keep buffered PDUs in a PDU window. The Receiver's PDU window is advanced upon reception of a new data PDU and the Sender's PDU window is advanced upon reception a positive acknowledgement for a data PDU. The Sender can also poll the Receiver (see *Polling*, pg 22). When the Receiver is polled, it transmits a status report to the Sender, indicating the successfully and erroneously received data PDUs (see *Status Report Transmission*, pg. 24). The Sender can

retransmit the erroneously received data PDUs or discard them (by informing the Receiver). In the latter case, their windows are advanced without retransmitting the PDUs (see *SDU discard with explicit signaling procedure*, pg. 28). Finally, the RLC protocol defines a few more functions, like RLC Reset (pg. 31), Suspend (pg. 33) and others.

The RLC/AM protocol can be configured via a number of parameters (see *RLC/AM parameters*, pg. 52). The behavior of the protocol can vary significantly depending on the parameters used. It is left up to the UMTS operator to determine the appropriate values of these parameters. A (very) general guideline is that performance is improved and radio resources are used more efficiently with larger window sizes and buffers, as the buffer usage follows the TCP congestion control algorithms [3] [4] [9] [16].

2.1 Protocol Data Units (PDUs)

The RLC/AM protocol uses Data PDUs (AMD PDUs) for transmitting the user's data and Control PDUs (Status, RESET and RESET ACK PDUs) for the protocol's control messages. The Status PDUs can be independent PDUs or piggybacked on an AMD PDU (in its padding space). The RLC/AM PDUs are bit strings with a multiple of 8 bits in length.

2.1.1 Data (AMD) PDU

The AMD PDU is used to convey sequentially numbered PDUs containing RLC SDU data from the Sender to the Receiver (see *Data transfer*, pg. 20) along with the Polling bit and optionally piggybacked status information.

The AMD PDU header consists of the first two bytes, which contain the sequence number (SN), the polling bit (P), and a Header Extension (HE) type field (used by the segmentation and reassembly function). The RLC header consists of the AMD PDU header and all the bytes that contain Length Indicators (LIs). AMD PDUs have a fixed size, defined by upper layers.

The RLC header for the AMD PDU has the following fields:

- **D/C:** This bit indicates whether this is a Data or Control PDU. It is set for Data PDUs.

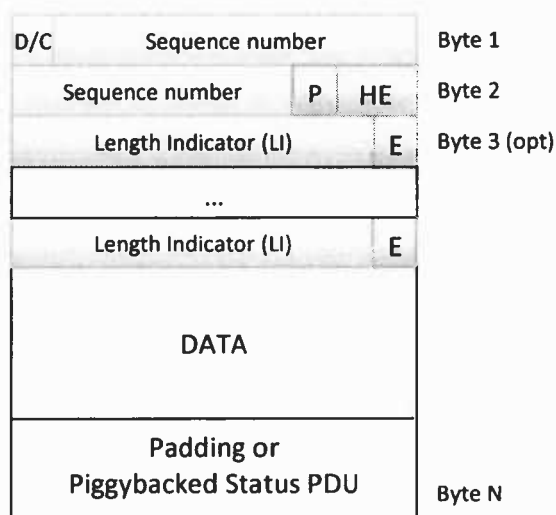


Fig. 2 Layout of an AMD PDU

- **Sequence number:** The sequence number of the AMD PDU. Its length is 12 bits.
- **Polling bit (P):** This bit is set to request a status report from the Receiver. If it is not set then a status report is not requested.
- **Extension bit (E):** This bit indicates whether the next field is data/piggybacked Status PDU/padding or a LI field and E bit.
- **Header Extension (HE) type:** This two-bit field indicates whether the next byte is data or a LI and E bit.
- **Length Indicator (LI):** This field is used for retransmission and reassembly. It indicates the last byte of each SDU ending in this PDU. The size of this field is 7 bits if the AMD PDU size is equal to or less than 125 bytes; otherwise the size of this field is 15 bits. This means that, barring any other restrictions, an AMD PDU can have a maximum payload of 32 KB.
- **Data:** SDUs or segments of SDUs are mapped to this field. The last segment of an SDU is concatenated with the first segment of the next SDU so that the data field is filled completely and padding is minimized. Segmentation and reassembly is not implemented in our simulation.
- **Padding:** Unused space at the end of the PDU which is ignored by the Sender and the Receiver. Padding has a length such that the PDU as a whole has one of the predefined length totals. Padding is not implemented in our simulation.

Piggybacked Status PDU


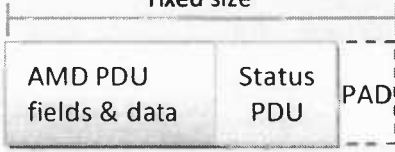
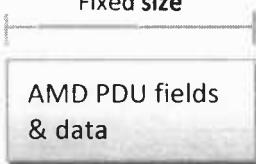
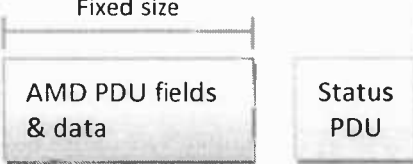
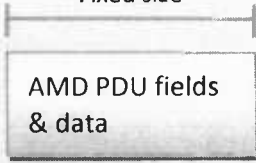
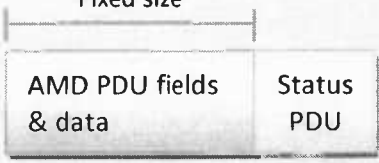
A Status PDU can be optionally piggybacked on an AMD PDU, if it fits in the padding space of the AMD PDU. The piggybacked Status PDU's format is the same as an ordinary Status PDU's format. Our implementation includes this feature for the sake of completeness but it deviates from the specification in an important way. Since our PDUs do *not* have padding space (since no SaR or concatenation is performed and no padding is added), the Status PDU is piggybacked at the very end of the AMD PDU. This means that AMD PDUs with piggybacked Status PDUs do *not* have a fixed size, which is against the specification. The piggybacked Status PDU usually adds approximately 3 to 8 bytes in each AMD PDU, depending on the SUFIs and SUFI sizes of the Status PDU.

The RLC protocol does not specify when a Status PDU should be piggybacked (in order to promote competition among implementations). Our piggybacking feature is based on a delayed Status PDU function. When a Status PDU is triggered, a timer is started. If an AMD PDU is sent while the timer is active, the Status PDU is piggybacked on it. If the timer expires and no AMD PDU is sent during that time then an ordinary Status PDU is sent instead.

Predictably, the RLC protocol with this kind of piggybacking has a lower performance compared to the RLC without any piggybacking. This is because AMD PDUs



with piggybacked Status PDUs are larger than ordinary AMD PDUs (in our implementation) and there is a higher probability of an error to occur to such an AMD PDU during transmission over the air. As a result, the piggybacking feature was not used in our simulations. Piggybacking in our implementation can be enabled or disabled by setting a parameter (see *RLC/AM parameters*, pg. 52).

	AMD PDU without piggybacked Status PDU	AMD PDU with piggybacked Status PDU
RLC Specification	<p>Fixed size</p> 	<p>Fixed size</p>  <p>The AMD PDU with a piggybacked Status PDU has the same size as an ordinary AMD PDU.</p>
Our implementation with piggybacking disabled	<p>Fixed size</p>  <p>The AMD PDU has no padding space. Its size is fixed (because SDUs from the upper layer have a fixed size).</p>	<p>Fixed size</p>  <p>The AMD PDU has no padding space. The Status PDU is not piggybacked. Two separate PDUs are sent.</p>
Our implementation with piggybacking enabled	<p>Fixed size</p>  <p>Same as above.</p>	<p>Fixed size</p>  <p>The AMD PDU with piggybacked Status PDU is larger than ordinary AMD PDUs. Its size is also variable (as there is no padding space).</p>

2.1.2 Status PDU

A Status PDU is used

- By the Receiver to inform the Sender about missing and received AMD PDUs (see *Status Report Transmission*, pg. 24).
- By the Sender to request the Receiver to move the reception window (see *SDU discard with explicit signaling procedure*, pg. 28).
- By the Receiver to acknowledge the Sender about the reception of the request to move the window (see *SDU discard with explicit signaling procedure*, pg. 28).
- By the Receiver to inform the Sender of the size of the transmission window whenever the Receiver needs to update the Sender's window.

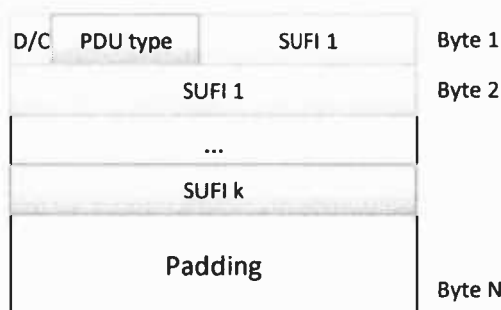


Fig. 3 Layout of a Status PDU

The Status PDU has the following fields:

- **D/C:** This bit indicates whether this is a Data or Control PDU. It is set for Data PDUs.
- **PDU type:** This field indicates the type of Control PDU (Status, RESET or RESET ACK).
- **SUFI(s):** The Status PDU is essentially a sequence of Super Fields (SUFIs). SUFIs are fields that have variable size and it is implementation dependent which SUFIs to use. A SUFI has three sub-fields, Type, Length and Value, of which only the first is mandatory. The Type and Length fields have fixed size.

Status PDU SUFIs

Acknowledgment & Negative acknowledgement SUFIs

SUFI	Description
ACK	Acknowledgement
	This super-field consists of a Type field and a sequence number (value) field. This SUFI is also indicating the end of the data part of a Status PDU and it must always be the last SUFI. This means that a NO_MORE

	SUFI is not required when the ACK SUFI is present. This SUFI acknowledges the reception of all AMD PDUs with sequence number less than the value of the SUFI.
LIST	<p>List</p> <p>This SUFI consists of a Type, a Length and a Value sub-field. In its Value sub-field there is a list of sequence number/length pairs. Each pair indicates the sequence number of an AMD PDU that has not been correctly received (in the sequence number part) and the number of consecutive AMD PDUs not correctly received following that AMD PDU (in the length part).</p>
BITMAP	<p>Bitmap</p> <p>This SUFI consists of a Type, a Length and a Value sub-field. In its Value sub-field there is a first sequence number (FSN) and a bitmap. The FSN indicates the sequence number of the first bit in the bitmap. Each bit in the bitmap indicates the status of the AMD PDUs with the corresponding sequence number. If the value of a bit is 1, then the AMD PDU with the corresponding sequence number has been correctly received by the Receiver; otherwise it has not been correctly received.</p>
RLIST	<p>Relative List</p> <p>This SUFI consists of a Type, a Length and a Value sub-field. In its Value sub-field there is a first sequence number (FSN) and a list of codewords. The FSN indicates the first erroneous AMD PDU received by the Receiver. The codewords (when decoded) indicate a list with the distance between the previous indicated erroneous AMD PDU up to the next one.</p>

Other SUFIs

SUFI	Description
NO_MORE	<p>No More Data</p> <p>This super-field indicates the end of the data part of a Status PDU and it does not have Length or Value sub-fields. It must always be placed as the last SUFI in a Status PDU. All data after this SUFI is regarded as padding.</p>
MRW	<p>Move Receiving Window</p> <p>This SUFI is used by the Sender to request the Receiver to move its reception window and optionally to indicate the set of discarded RLC SDUs as a result of an RLC SDU discard in the Sender. Among others, it</p>

	includes a sequence number indicating that all erroneous AMD PDUs with sequence number less than the indicated SN must be discarded.
MRW_ACK	Move Receiving Window Acknowledgement
	This SUFI acknowledges the reception of a MRW SUFI by the Receiver. Its value (in an SN_ACK subfield) indicates the new value of the VR(R) state variable (see <i>Receiver variables</i> , pg. 16).
WINDOW	Window Size
	This SUFI consists of a Type field and a window size number (value). The Receiver is always allowed to change the transmission window size of the Sender during a connection. The reception window of the Receiver is not changed. The value of this SUFI indicates the value of the VT(W) parameter to be used by the Sender (see <i>Sender variables</i> , pg. 14).

Our implementation uses only the NO_MORE, ACK, BITMAP, MRW and MRW_ACK SUFIs.

2.1.3 RESET & RESET ACK PDU

A RESET PDU is used to transmit a reset command and it is sent by the Sender to the Receiver, while a RESET ACK PDU is used to transmit an acknowledgement to a RESET PDU and it is sent by the Receiver to the Sender (see *RLC Reset procedure*, pg. 31).

These PDUs have the following fields:

- **D/C:** This bit indicates whether this is a Data or Control PDU. It is set for Data PDUs.
- **PDU type:** This field indicates the type of Control PDU (Status, RESET or RESET ACK).
- **Reset Sequence Number:** This bit indicates the sequence number of the transmitted RESET PDU. If this RESET PDU is a retransmission of the original RESET PDU then the retransmitted PDU will have the same sequence number as the original; otherwise it will have the next RSN value. The value of this field is not reinitialized when the RLC entity is reset.
- **Reserved 1 (R₁):** This field is not used.
- **Hyper Frame Number Indicator (HFNI):** This field is used to syn-

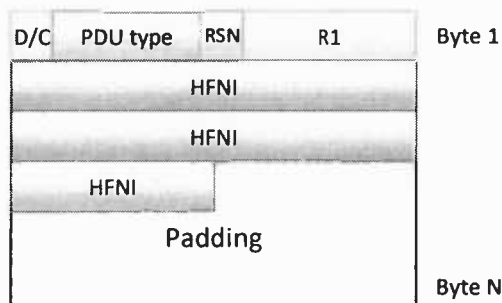


Fig. 4 Layout of the RESET and RESET ACK PDUs

chronize the hyper frame numbers (used for ciphering) in the peer entities.

2.2 Protocol States

The RLC protocol has five distinct states in AM. These are the NULL, DATA_TRANSFER_READY, RESET_PENDING, LOCAL_SUSPEND and RESET_AND_SUSPEND states, as shown in Fig. 5.

The NULL state is the initial state of the RLC entity. In this state, the RLC entity does not exist (and cannot send or receive data). Upon request from upper layers, the entity is created and enters the DATA_TRANSFER_READY state.

In the DATA_TRANSFER_READY state, the entity can exchange data with its peer entity. Upon request from upper layers, it can be terminated (and enter the NULL state) or be suspended (and enter the LOCAL_SUSPEND state). It can initiate the RLC Reset procedure by sending a RESET PDU (and enter the RESET_PENDING state) if a reset condition is triggered. If the entity receives a RESET ACK PDU, it does nothing but if it receives a RESET PDU, it resets itself, responds with a RESET ACK PDU and continues normal data transmission (see *RLC Reset procedure*, pg. 31).

In the RESET_PENDING state, the entity waits for a response from its peer entity. No data can be exchanged between the entities. Upon request from upper layers, the entity can be terminated (and enter the NULL state) or be suspended (and enter the RESET_AND_SUSPEND state). If the entity receives a proper RESET ACK PDU, it resets itself and enters the DATA_TRANSFER_READY state, and if receives a RESET PDU, it responds with a RESET ACK PDU and stays in this state, according to the RLC

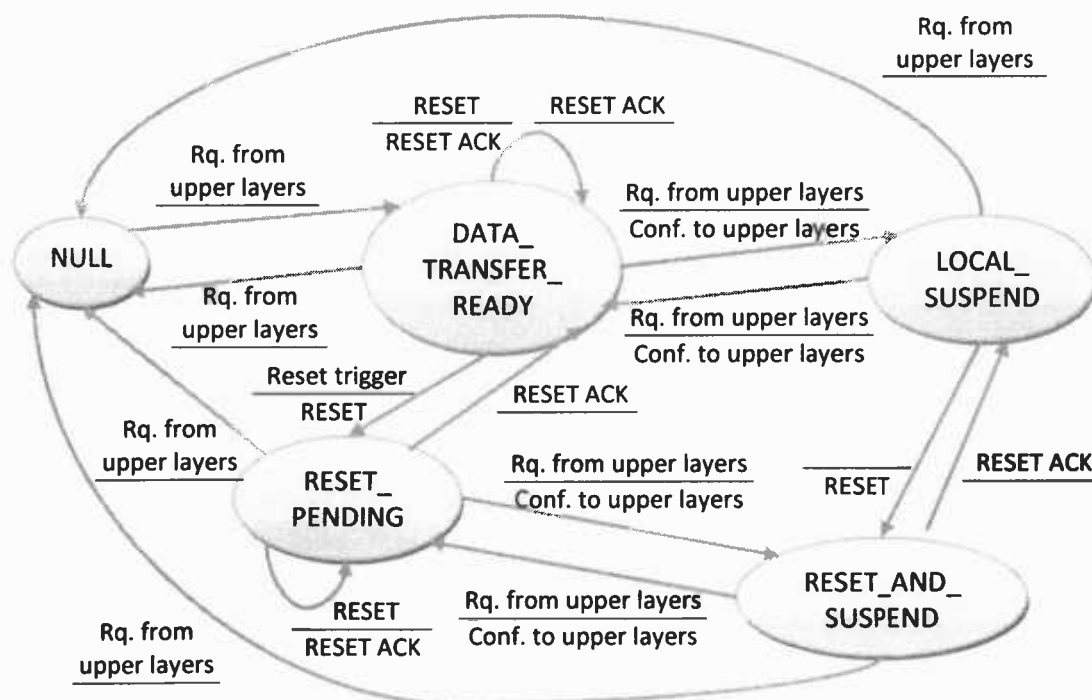


Fig. 5 The state model for AM RLC entities

Reset procedure.

In the LOCAL_SUSPEND state, the entity is suspended, i.e. it does not send AMD PDUs with sequence number greater than a specified value. Upon request from upper layers, it can be terminated (and enter the NULL state) or resume normal data transmission (and enter the DATA_TRANSFER_READY state). It can initiate the RLC Reset procedure by sending a RESET PDU (and enter the RESET_AND_SUSPEND state) if a reset condition is triggered.

Finally, in the RESET_AND_SUSPEND state, the entity waits for a response from the peer entity or a request from upper layers and no data can be exchanged between the entities. Upon request from upper layers, the entity can be terminated (and enter the NULL state) or resumed (and enter the RESET_PENDING state). If the entity receives a proper RESET ACK PDU, it resets itself (according to the RLC Reset procedure) and enters the LOCAL_SUSPEND state.

In our simulation the LOCAL_SUSPEND and RESET_AND_SUSPEND states are not implemented because we do not implement the Local Suspend function (see pg. 33).

2.3 State Variables

All state variables for the RLC protocol are nonnegative integers. AMD PDUs are numbered by modulo sequence numbers (SN) cycling through the field 0 to $2^{12}-1$ (=4095), therefore all arithmetic operations in variables containing sequence numbers¹ are affected by this modulus. When performing arithmetic operations, the modulus base is subtracted (within the appropriate field) from all the values involved and then an absolute comparison is performed. At the Sender the modulus base is VT(A) and at the Receiver it is VR(R).

2.3.1 Sender variables

Variable	Description
VT(S)	Send state variable This variable contains the sequence number of the next AMD PDU to be transmitted for the first time (i.e. excluding retransmissions) by the Sender. It is update when an AMD PDU is transmitted for the first time. Its initial value is 0.
VT(A)	Acknowledge state variable This variable contains the sequence number following the sequence number of the last in-sequence acknowledged AMD PDU. This forms the

¹ i.e. for the variables VT(S), VT(A), VT(MS), VR(R), VR(H) and VR(MR).



	lower edge of the transmission window of acceptable acknowledgements. This variable is updated when a Status PDU with an ACK and/or MRW_ACK SUFI is received. Its initial value is 0.
VT(DAT)	This variable counts the number of times an AMD PDU has been scheduled to be transmitted (or retransmitted). There is one VT(DAT) variable for each PDU and each is incremented every time the corresponding PDU is transmitted (or retransmitted). The initial value of this variable is 0.
VT(MS)	<p>Maximum Send state variable</p> <p>This variable contains the sequence number of the first AMD PDU that can be rejected by the Receiver. This variable represents the upper edge of the transmission window. This means that $VT(MS) = VT(A) + VT(WS)$. The Sender cannot transmit PDUs with sequence number equal to or greater than this variable. It is updated when VT(A) or VT(WS) is updated. Its initial value is the value of the Configured_Tx_Window_Size parameter.</p>
VT(PDU)	This state variable is used when "Every x PDU" polling is configured (see <i>Polling</i> , pg. 22). It is incremented every time an AMD PDU is transmitted (including both new and retransmitted PDUs). When it becomes equal to the value of the Poll_PDU parameter, a poll is triggered and the variable is reset to 0. The initial value of this variable is 0.
VT(SDU)	<p>This state variable is used when "Every x SDU" polling is configured (see <i>Polling</i>, pg. 22). It is incremented every time a SDU is transmitted for the first time, or more specifically, when the AMD PDU containing the first segment of the SDU is transmitted for the first time. When it becomes equal to the value of the Poll_SDU parameter, a poll is triggered and the variable is reset to 0. The initial value of this variable is 0.</p> <p>In our simulation, since segmentation and reassembly are not implemented, this variable simply counts the PDUs received from the upper layers. The only difference with VT(PDU) is that VT(SDU) does not count retransmitted PDUs.</p>
VT(RST)	<p>Reset state variable</p> <p>This variable counts the number of times a RESET PDU is transmitted before the reset procedure is completed. It is only reset when a RESET ACK PDU is received, i.e. it is not reset when an RLC reset initiated by the peer entity occurs. The initial value of the variable is 0. See <i>RLC Reset procedure</i>, pg. 31.</p>
VT(MRW)	MRW command send state variable

	This variable counts the number of times a MRW command is transmitted. It is reset when the SDU discard with explicit signaling procedure is terminated. Its initial value is 0. See <i>SDU discard with explicit signaling procedure</i> , pg. 28.
VT(WS)	Transmission window size state variable
	This variable contains the size of the transmission window. It is updated by a WINDOW SUFI in a Status PDU by the Receiver. Its initial value is the value of the Configured_Tx_Window_Size parameter.

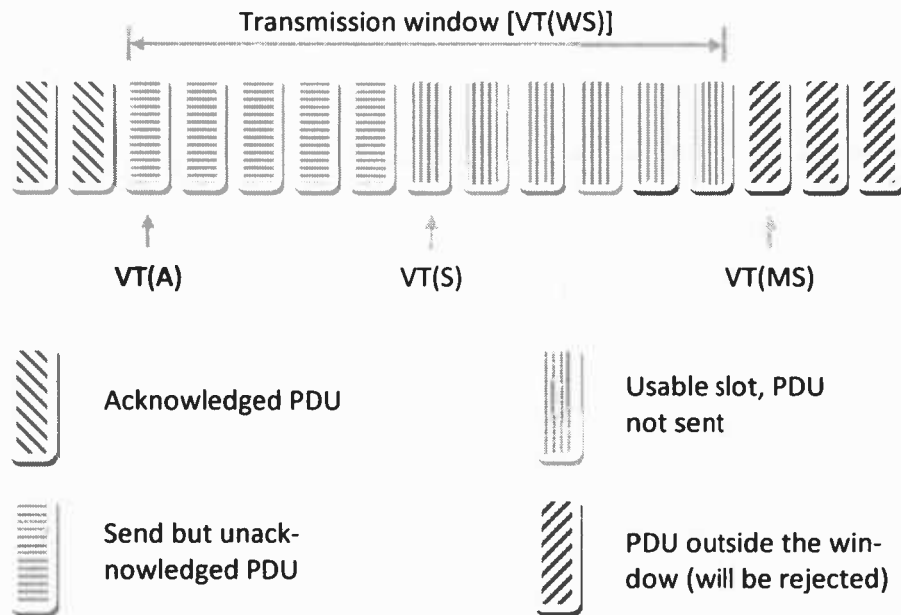
2.3.2 Receiver variables

Variable	Description
VR(R)	Receive state variable
	This variable contains the sequence number following that of the last in-sequence AMD PDU received. It is updated when an AMD PDU with sequence number equal to its value is received. Its initial value is 0.
VR(H)	Highest expected state variable
	This variable contains the sequence number following the highest sequence number of any received AMD PDU. When an AMD PDU is received with sequence number x such that $VR(H) \leq x < VR(MR)$, this variable is set to $x + 1$. Its initial value is 0.
VR(MR)	Maximum acceptable Receive state variable
	This state variable contains the sequence number of the first AMD PDU that will be rejected by the Receiver. This means that $VR(MR) = VR(R) + \text{Configured_Rx_Window_Size}$

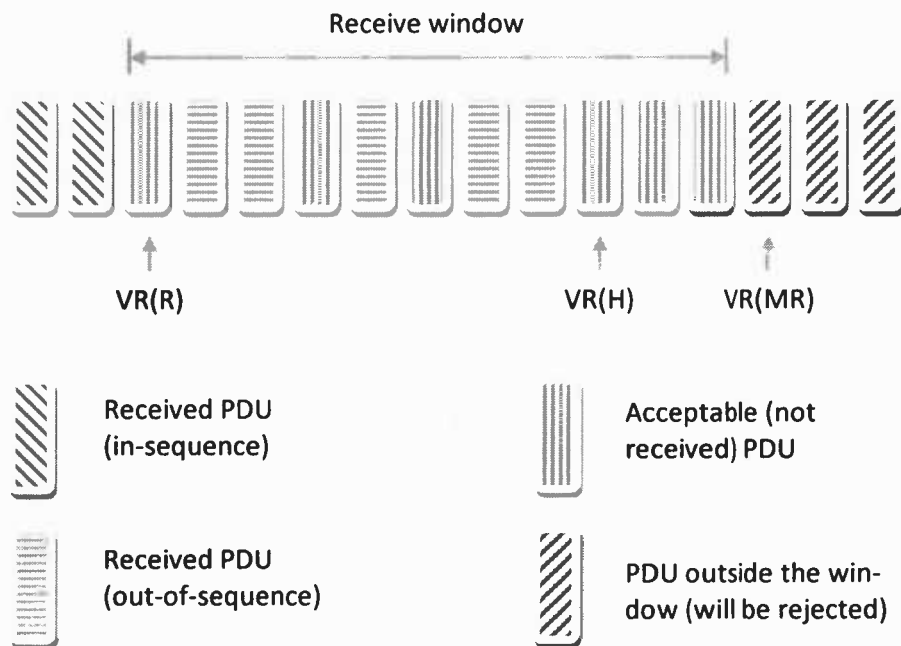
2.4 Windows

The following figures show the transmission window (at the Sender) and the receive window (at the Receiver). Note that the transmission and receive windows do not necessarily have the same size; the Receiver can enlarge (up to a maximum size) or shrink the Sender's window as part of the flow control it performs. The flow control function of the Receiver uses the WINDOW SUFI of a Status PDU to inform the Sender of the new size of the transmission window. The RLC protocol does not specify the flow control algorithm (leaving that up to the implementation). It specifies only the means (signaling) to control the flow. Apparently, this is done in order to promote competition among potential implementations.

2.4.1 Transmission window



2.4.2 Receive window



2.5 Timers

The RLC/AM protocol uses a number of timers for its operation. The timers are considered active from the moment they are started until they expire or are stopped.

2.5.1 Polling timers

Timer	Description
Timer_Poll	<p>This timer is started (or restarted) upon a transmission of an AMD PDU containing a poll. If x is the value of the VT(S) variable after the poll was submitted to the lower layer then the Poll Timer is stopped upon receiving positive acknowledgements for all the AMD PDUs with sequence number up to and including $x - 1$ or a negative acknowledgement for ADM PDU with sequence number $x - 1$. If the timer expires and no Status PDU fulfilling the above criteria is received then a poll is triggered, the new value of VT(S) is saved and the timer restarts.</p> <p>This timer is used only if configured so by upper layers. See <i>Polling</i>, pg. 22.</p>
Timer_Poll_Prohibit	<p>This timer is used to prohibit transmission of polls within a certain time period. It is started (or restarted) when an AMD PDU with a poll is transmitted. Polling is prohibited while this timer is active. Any polls triggered while they are prohibited must be delayed and sent when the timer expires. Only one poll is sent in such a case (despite the fact that many polls may have been triggered while polling was prohibited).</p> <p>This timer is used only if configured so by upper layers. See <i>Polling</i>, pg. 22.</p>
Timer_Poll_Periodic	<p>This timer is used only if "timer-based" polling is configured by upper layers. This timer is started when the RLC entity is created. When it expires, a poll is triggered and the timer is restarted. See <i>Polling</i>, pg. 22.</p>

2.5.2 Status reporting timers

Timer	Description
Timer_Status_Prohibit	<p>This timer is used to prohibit the Receiver from sending consecutive status reports. The timer is started upon transmission of a status report in a Status PDU (piggy-backed or ordinary). While this timer is active, the Receiver cannot send any status reports (i.e. Status PDUs with a LIST, BITMAP, RLIST or an ACK SUFI). Other SUFIs are not prohibited. Any status reports triggered during the</p>

time the timer is active (other than the ones triggered by lower layers) must be delayed and sent when the timer expires. The status reports can be updated while they are delayed.

This timer is used only if configured so by upper layers. See *Status Report Transmission*, pg. 24.

Timer_Status_Periodic This timer triggers a status report transmission in the Receiver periodically. It is started when the RLC entity is created and when it expires, it triggers a status report and it is restarted.

This timer is used only if configured so by upper layers. See *Status Report Transmission*, pg. 24.

2.5.3 Other timers

Timer	Description
Timer_RST	<p>This timer is used to handle the loss of a RESET or RESET ACK PDU. It is started (or restarted) upon transmission of a RESET PDU and it is stopped when the Reset procedure terminates. If it expires, the RESET PDU is retransmitted and the reset counter (VT(RST) variable) is incremented.</p> <p>See <i>RLC Reset procedure</i>, pg. 31.</p>
Timer_MRW	<p>This timer is used to retransmit a Status PDU containing a MRW SUFI. It is started when a Status PDU with a MRW SUFI is sent for the first time. It is stopped if the MRW procedure is acknowledged. If it expires, the MRW SUFI is retransmitted and a counter for the MRW command (VT(MRW) variable) is incremented.</p> <p>See <i>SDU discard with explicit signaling procedure</i>, pg. 28.</p>
Timer_Discard	<p>This timer is used only when the Timer-based discard variant for SDU discard is configured (see <i>Limited Reliability of the RLC/AM</i>, pg. 27). An instance of this timer is started for each SDU sent for the first time. If the timer expires and the SDU has not been acknowledged, it is discarded with the SDU discard function.</p>

3 Data transfer, polling & status reporting

3.1 Data transfer

The (acknowledged mode) data transfer procedure is obviously the most important procedure of the RLC protocol, as it is the one used for transferring the user's data between two peer RLC entities. The data is transferred from the Sender to the Receiver. This procedure is used only when the RLC entity is in the DATA_TRANSFER_READY state of LOCAL_SUSPEND state. The procedure is initiated upon a request for data transfer from the upper layers or upon retransmission of AMD PDUs.

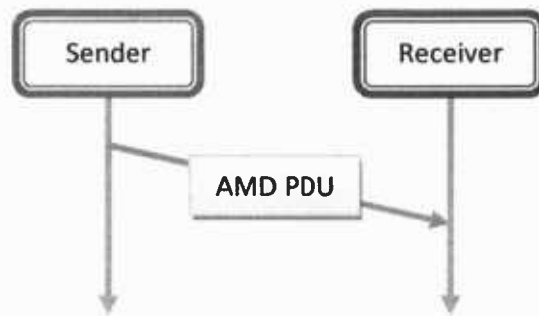


Fig. 6 Data transfer procedure

If the procedure is initiated from the upper layers then the Sender must:

- Perform segmentation of the RLC SDU(s) received from the upper layer and, if possible, concatenation of the SDU(s) into AMD PDUs. The PDU size is fixed and configured by upper layers and a Length Indicator (LI) field is set for each SDU that ends in an AMD PDU. In our simulation, segmentation and reassembly is not implemented; we expect the upper layers to have dealt with that, delivering fixed-size packets to the RLC. This is so that the RLC implementation can be comparable with other pre-existing protocols, such as several Selective Repeat implementations.
- For each PDU, set the PDU's sequence number equal to the VT(S) variable, increment the variable and set the PDU's polling bit if a poll has been triggered and is not prohibited (see *Polling*, pg. 22).
- Start a timer *Timer_Discard* for each SDU received from upper layer, if the "Timer based discard with explicit signaling" variant of the SDU Discard function is used (see *Timer based discard with explicit signaling*, pg. 27).
- Schedule the PDU(s) for transmission.

If the PDU is a retransmission, the Sender must:

- Set the PDU's sequence number as in the original transmission of the AMD PDU.
- Update the PDU's LI field(s) (for example, if a piggyback Status PDU was transmitted originally, it is not retransmitted).

- Set the PDU's polling bit if a poll has been triggered and is not prohibited.
- Schedule the PDU for transmission.

Each time a PDU is scheduled for transmission or retransmission, the Sender

- Increments the value of the corresponding VT(DAT) variable (counting the number of transmissions for this PDU) and initiates the SDU Discard function or the Reset procedure if the "SDU discard after x number of transmissions" or "No_discard after x number of transmissions" variant is used (see *Limited Reliability of the RLC/AM*, pg. 27) and the value of the VT(DAT) variable is equal to or greater than the value of the MaxDAT parameter.
- Optionally attaches a piggybacked Status PDU to the padding space of the AMD PDU by updating the AMD PDU's LI fields, if a Status PDU has been scheduled.
- Makes sure that it is not prohibited to schedule the PDU for transmission (for example, in case the Local Suspend function is initiated).
- Makes sure that the sequence number of the transmitted PDU is less than the value of the VT(MS) variable or equal to VT(S) - 1.
- Treats retransmissions with higher priority than AMD PDUs transmitted for the first time.
- Starts the timer Timer_Poll if the polling bit of the PDU has been set (see *Polling*, pg. 22).
- Buffers the PDUs according to the SDU Discard configuration.

Upon reception of an AMD PDU, the Receiver:

- Updates its VR(R), VR(H) and VR(MR) state variables, i.e. it moves its window bounds.
- If the received AMD PDU has its polling bit set or the Receiver detects a missing PDU and is configured to initiate a status report, then it assembles a status report and initiates the Status PDU transfer procedure.
- Reassembles the received AMD PDUs into RLC SDUs (not implemented in our simulation).
- Delivers the SDUs in-sequence, if configured to do so, otherwise it delivers them arbitrarily. Our implementation supports in-sequence delivery only.

Abnormal cases

The protocol specifies a number of abnormal cases.

- **Receiving an AMD PDU outside the reception window:** Upon reception of an AMD PDU with sequence number SN outside the interval $VR(R) \leq SN \leq VR(MR)$, the Receiver discards the AMD PDU. Still, if the AMD PDU had its polling bit set, then the receiver prepares a status report and initiated the Status PDU transfer procedure.
- **Receiving an AMD PDU within the reception window more than once (Handling of Duplicates):** Upon reception of an AMD PDU with sequence number SN within the interval $VR(R) \leq SN \leq VR(MR)$, for which an AMD PDU has already been received, the Receiver discards the AMD PDU and considers the AMD PDU with this sequence number as having been correctly received in the next status report. However, before discarding the duplicate PDU, the Receiver processes its polling bit (and sends a status report if the polling bit is set) and processes its piggy-backed Status PDU, if there is any.
- **Timer_Discard timeout:** Upon expiry of a timer `Timer_Discard`, the Sender initiated the SDU discard with explicit signaling procedure (see pg. 28).
- **Invalid LI field value:** If the value of the LI field of a received AMD PDU is larger than the PDU size or if its value is one of the reserved values then the Receiver ignores that AMD PDU.
- **Invalid PDU size:** If the PDU size of a received PDU is different than the configured AMD PDU size then the Receiver ignores that AMD PDU.
- **Full Buffer Behavior (for the UE):** The UE may have memory limitations. When the buffer memory is full, the UE is not required to segment the SDUs into PDUs. It should be able to process incoming AMD PDUs (especially the AMD PDU with sequence number equal to $VR(R)$, i.e. the next expected AMD PDU) and operate according to the normal protocol (e.g. process status reports and perform retransmissions). However, it may discard received AMD PDUs within the receive window and consider the discarded AMD PDUs as not having been received.

3.2 Polling

The polling function is used by the Sender to request a status report from the Receiver. This is done by setting the polling bit in an AMD PDU. Upon reception of an AMD PDU with the polling bit set, the receiver must transmit a Status PDU (or piggy-backed Status PDU) with positive and/or negative acknowledgements of AMD PDUs, according to the Status PDU transfer procedure.

There are several triggers for the polling function in the receiver; any combination of which may be active, as configured by upper layers. These triggers are:

- **Every last PDU in buffer:** When an AMD PDU is to be transmitted for the first time and it is the last PDU scheduled or allowed to be transmitted then a poll is triggered for that AMD PDU. This trigger was not implemented in our simulation because the Suspend function of the RLC protocol was not implemented and a scheduling buffer was not used in this layer.

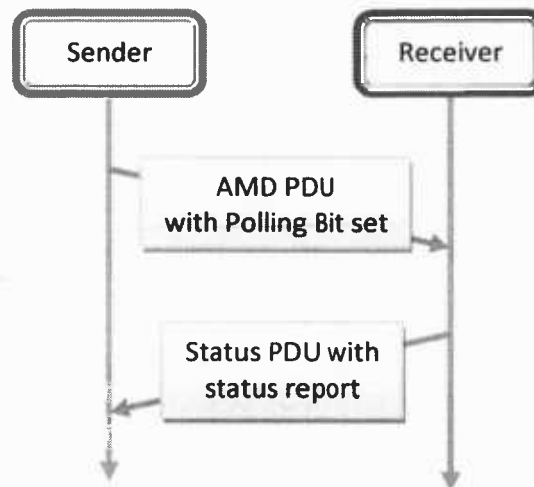


Fig. 7 Polling & status report transmission procedure

- **Every last PDU in Retransmission buffer:** When a retransmitted AMD PDU is submitted to the lower layer and it is the last PDU scheduled for retransmission or the last PDU allowed to be retransmitted, then a poll is triggered for that AMD PDU.
- **Poll Timer:** A poll is triggered when the Poll Timer (Timer_Poll) expires. The Poll Timer is started (or restarted) when an AMD PDU containing a poll is submitted to the lower layer. If x is the value of the VT(S) variable after the poll was submitted to the lower layer then the Poll Timer is stopped upon receiving positive acknowledgements for all the AMD PDUs with sequence number up to and including $x - 1$ or a negative acknowledgement for ADM PDU with sequence number $x - 1$. If the timer expires and no Status PDU fulfilling the above criteria is received then a poll is triggered, the new value of VT(S) is saved and the timer is restarted.
- **Every x PDU:** A poll is triggered by the Sender for every x PDUs, where x equals the Poll_PDU parameter. Both new and retransmitted AMD PDUs are counted.
- **Every x SDU:** A poll is triggered by the Sender for every x SDUs, where x equals the Poll_SDU parameter. The poll is triggered for the last PDU that contains the x^{th} SDU. In our implementation segmentation, reassembly and padding are not implemented. Therefore, the only difference compared to the "Every x PDU" trigger is that no retransmissions are counted this time.
- **Window Based:** A poll is triggered by the Sender when a percentage of the Sender's window is used. The portion is defined by the Poll_Window parameter. If the percentage J used is at least as much as the Poll_Window parameter, i.e. if $J \geq \text{Poll_Window}$, then a poll is triggered.

$$J = \frac{(4096 + VT(S) + 1 - VT(A)) \bmod 4096}{VT(WS)} \cdot 100$$

- **Timer Based:** A poll is triggered periodically by the Sender by using the `Timer_Poll_Periodic`.

It shall be noted that misconfiguration can lead to deadlock situations [7]. It has been shown that good performance results can be achieved when using the “every last PDU in buffer” and “every last PDU in retransmission buffer” triggers [17].

The specification suggests that when the polling function is triggered and there are no PDUs scheduled to be transmitted and there are unacknowledged PDUs, then the last unacknowledged PDU shall be retransmitted with its polling bit set. This feature was not implemented in our simulation because we do not use a scheduling buffer in this layer.

The Poll Prohibit function can be used by the Sender to delay the initiation of the Polling function, if so configured by upper layers. When the Poll Prohibit function is used, polling is prohibited (delayed) when the `Timer_Poll_Prohibit` is active. The polling procedure is changed so that:

- If a poll is triggered and polling is not prohibited then the polling bit is set (as before) and the `Timer_Poll_Prohibit` is started.
- If a poll is triggered and polling is prohibited then the polling function is not initiated.
- When the `Timer_Poll_Prohibit` expires, if the polling function was triggered at least once (and was delayed) then the polling function is initiated.

It has been observed that there is a tradeoff between protocol throughput and delay when setting the `Timer_Poll` and `Timer_Poll_Prohibit`. Throughput is increased for low values of these timers, as it is suggested that small polling periods should improve performance [4], but in practice a UMTS operator might not be able to identify an optimal configuration [10].

3.3 Status Report Transmission

The Receiver transmits status reports to the Sender in order to inform the Sender about which AMD PDUs have been successfully received (positive acknowledgement) and which have not been received (negative acknowledgement). The Status Transmission procedure triggers are:

- **Polling:** If an AMD PDU with the polling bit set is received by the Receiver then the Status Transmission procedure is triggered.
- **Detection of Missing PDU(s):** The Receiver can trigger a Status report

transmission if it detects one or more missing PDUs and is so configured by upper layers.

- **Timer based status report transfer:** The Receiver triggers a Status report transmission to the Sender periodically, if so configured by upper layers. This is accomplished by using the `Timer_Status_Periodic`. It has been found that best performance is achieved for timeout values slightly higher than the round trip time [2].
- **Request from lower layers:** The lower layers can request the generation of a status report following a MAC-hs reset. This trigger was not implemented in our simulation since there is no support for such a function from the lower layers in our simulated environment.

A status report transmission consists of one or more Status PDUs. The Status PDUs must have complete SUFIs. The Receiver can use the LIST, BITMAP, RLIST and ACK SUFIs of a Status PDU in order to inform the Sender about the received and not received PDUs. It is not specified which SUFIs exactly to use (other than ACK), apparently in order to promote competition among potential implementations. In our implementation, we use the BITMAP and ACK SUFIs.

There is a Status Prohibit Function that prohibits the transmission of Status PDUs with the above SUFIs. Status PDUs with other SUFIs are not prohibited by this function. This function can be used if so configured by upper layers. If a Status PDU containing a status report (and using one or more of the above SUFIs) is triggered and is prohibited then it must be delayed. The Status Prohibit Function uses the `Timer_Status_Prohibit` so that if this timer is active then status reporting is prohibited and it does not prohibit status reports that were triggered by lower layers.

If the Status Prohibit Function is used then the Status Transmission procedure is changed so that:

- The `Timer_Status_Prohibit` is started upon submission of a Status PDU with one or more of the status reporting SUFIs (LIST, BITMAP, RLIST and ACK) to the lower layers.
- If another such status report is triggered and this timer is active then the status reporting is prohibited and delayed until the timer expires (unless it was triggered by lower layers). The status report can be updated during this time. The transmission of SUFIs MRW, MRW_ACK, WINDOW or NO_MORE is not restricted. If a Status PDU containing one or more of those SUFIs and one or more of the status reporting SUFIs is triggered, then it is transmitted excluding the status reporting SUFIs.
- When the timer expires, if a status transmission with the status reporting SUFIs has been triggered and is no longer prohibited, then it is transmitted.

Upon reception of a Status PDU with a status report, the Sender must inform the

upper layers of any positively acknowledged PDUs, if requested to do so, and update its state variables VT(A) and VT(MS) (i.e. move its window bounds) according to the information in the received Status PDU. The Sender also stops any `Timer_Discard` timers, if the corresponding AMD PDUs have been acknowledged (see *Timer based discard with explicit signaling*, pg. 27). If the Status PDU includes negatively acknowledged AMD PDUs then the Sender must retransmit them. Retransmitted AMD PDUs have higher priority than AMD PDUs to be transmitted for the first time. If an AMD PDU is negatively acknowledged more than once in a Status PDU (for example, in multiple SUFIs) then it shall be retransmitted once only. If a Status PDU has an inconsistent status indication then it is considered erroneous and it must be discarded.

4 Limited Reliability of the RLC/AM

The RLC/AM offers limited reliability on the delivery of SDUs from upper layers. SDUs that have not been correctly received within a certain time period or number of retransmissions are discarded by the Sender and the Sender and Receiver windows are advanced (moved). This is done in order to minimize retransmission conflicts with the retransmissions at the TCP layer and it also helps prevent buffer overflows. The **SDU discard function** is used for this purpose.

The SDU Discard function is used by the Sender in order to discard one or more PDUs from its buffers and (in the AM) inform the Receiver of the discarded PDUs. There are three alternative operation modes of the Discard function for the AM.

It shall be noted that in our implementation the discard function discards PDUs, not SDUs, as we do not perform segmentation and reassembly.

4.1 SDU Discard Operation Modes

4.1.1 Timer based discard with explicit signaling

This alternative uses a timer (`Timer_Discard`) for triggering the discard function. This makes the SDU discard function insensitive to variations in the channel rate and provides means for exact definition of maximum delay.

For every SDU received from upper layers the Sender starts a `Timer_Discard`. When such a timer expires, the SDU is discarded. Explicit signaling is used by the Sender to inform the Receiver of the discarded SDU, so that the Receiver can move its window.

A variant of this mode but without explicit signaling is used in the UM or TM modes.

4.1.2 SDU discard after x number of transmissions

This alternative uses the number of transmissions as a trigger for the discard function. This makes the discard function dependent on the channel rate. This variant also strives to keep the SDU loss rate constant for the connection, on the cost of variable delay.

In this alternative, if the number of times an AMD PDU is scheduled for transmission¹ (stored in a `VT(DAT)` variable) reaches or exceeds the value of the `MaxDAT` parameter then all the SDUs it contains are discarded. Explicit signaling is used by the Sender to inform the Receiver of the discarded SDU, so that the Receiver can move its window.

¹ This includes the first transmission and all the subsequent retransmissions.

4.1.3 No_discard after x number of transmissions

This alternative is similar to the previous one. The difference is that the SDUs are not discarded if the maximum number of transmissions is reached but the RLC Reset procedure is initiated instead (so all buffers at the Sender and Receiver are emptied and all variables reset).

4.2 SDU discard with explicit signaling procedure

The SDU discard with explicit signaling procedure is used for discarding SDUs and transferring the discard information between two peer RLC entities. This is used if the SDU Discard function uses the "Timer based discard with explicit signaling" or "SDU discard after x number of transmissions" variants. The Sender discards an SDU that has not been successfully transmitted for a period of time or a number of transmissions and sends a Move Receiving Window (MRW) SUFI to the Receiver.

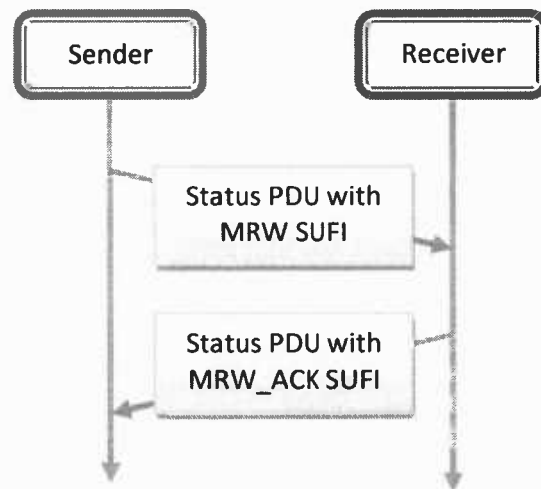


Fig. 8 SDU discard with explicit signaling

According to the MRW SUFI, the Receiver must discard AMD PDUs carrying that SDU and update (move) the reception window accordingly. The Receiver acknowledges the update by responding with a MRW_ACK SUFI.

The procedure is initiated if one or more SDUs must be discarded when using the "Timer based discard with explicit signaling" or "SDU discard after x number of transmissions" variants. Upon initiation, the Sender discards the appropriate SDUs and PDUs. In the "Timer based discard with explicit signaling" variant, the Sender discards all SDUs up to and including the SDU for which the discard function was initiated. Then, the AMD PDUs that contain segments of these SDUs are discarded (as long as they do not contain segments of other SDUs). In the "SDU discard after x number of transmissions" variant, the Sender discards all AMD PDUs with sequence number SN in the interval $VT(A) \leq SN \leq X$, where X is the sequence number of the AMD PDU with $VT(DAT) \geq MaxDAT$, and all SDUs with segments in these PDUs. If requested, the Sender can inform the upper layers of the discarded SDUs.

Once the appropriate SDUs and PDUs have been discarded, the Sender assembles one or more MRW SUFIs with the discard information, schedules them to be sent by the lower layers in a Status PDU (ordinary or piggybacked) and starts the Timer_MRW.

Upon reception of a Status PDU with an MRW SUFI, the Receiver delivers to the upper layers all AMD PDUs (as SDUs) that have successfully received up to the se-

quence number mentioned in the MRW SUFI and discards all the ones that have been unsuccessfully received. Then, it updates its $VR(R)$, $VR(H)$ and $VR(MR)$ variables according to the received MRW SUFI (i.e. it moves its window), it assembles a MRW_ACK SUFI with the new value of $VR(R)$ and it schedules the MRW_ACK SUFI to be sent to the Receiver.

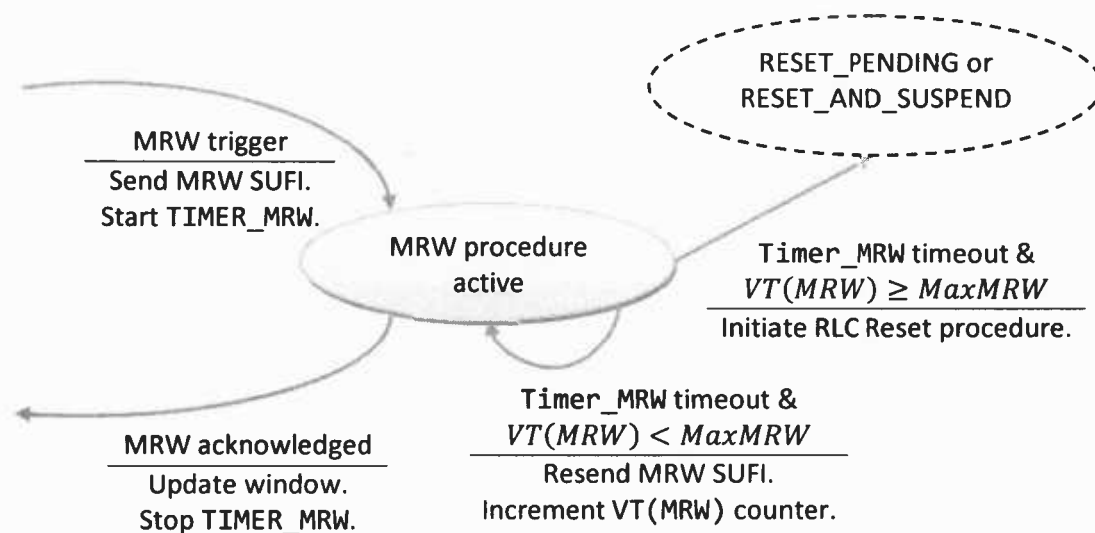


Fig. 9 The MRW procedure for the Sender

The procedure is terminated when the Sender receives a Status PDU (piggy-backed or ordinary) that contains one of the following:

- An MRW_ACK SUFI and its SN_ACK field is greater than the sequence number in the transmitted MRW SUFI, i.e. it indicates that all PDUs up to that point have been discarded.
- An ACK SUFI and the Status PDU indicates that all AMD PDUs up to and including the AMD PDU with sequence number equal to that in the transmitted MRW SUFI have been received or discarded by the peer entity.

Upon termination, the Sender stops the timer $Timer_MRW$ and updates its $VT(A)$ and $VT(MS)$ variables according to the received Status PDU (i.e. it moves its window).

The protocol addresses a number of abnormal cases. If the $Timer_MRW$ expires, then the Sender must increment its $VT(MRW)$ variable by one. If its value is now equal to or greater than the value of the $MaxMRW$ parameter, then the MRW procedure is terminated and the RLC Reset procedure must be initiated. Otherwise, the Sender retransmits the MRW SUFI as previously transmitted (even if additional SDUs were discarded in the mean time) and it restarts the $Timer_MRW$. Another abnormal case is when the Receiver receives an obsolete or corrupted MRW SUFI, in which case it discards the MRW SUFI and schedules an MRW_ACK SUFI for transmission with the current value of the $VR(R)$ parameter. Finally, if the Sender receives an obsolete or corrupted MRW_ACK SUFI (for example, if there is no ongoing SDU discard proce-

ture) then it must discard the SUFI.

Only one MRW procedure can be active at any time. If another MRW procedure is triggered while the first one is still active, then the second one must be delayed and re-initiated when the first one finishes.

5 Other procedures

5.1 RLC Reset procedure

The Reset procedure is used to reset the two RLC entities. It uses the RESET and RESET ACK PDUs, as shown in the figure. These PDUs have a higher priority than AMD PDUs. During the Reset procedure the hyper frame numbers (HFN) used for ciphering in UTRAN and UE are synchronized¹.

The Reset procedure can be triggered by a number of events:

- The Discard function is using the "No_Discard after x number of transmissions" variant and the VT(DAT) variable for an AMD PDU equals the value of the Max-DAT parameter.
- The variable VT(MRW), which counts the number of attempts for successfully sending a Status PDU with a MRW SUFI, equals the value of the MaxMRW parameter.
- A Status PDU (or piggybacked Status PDU) with erroneous sequence numbers in the status report (LIST, BITMAP, RLIST or ACK SUFIs) is received.

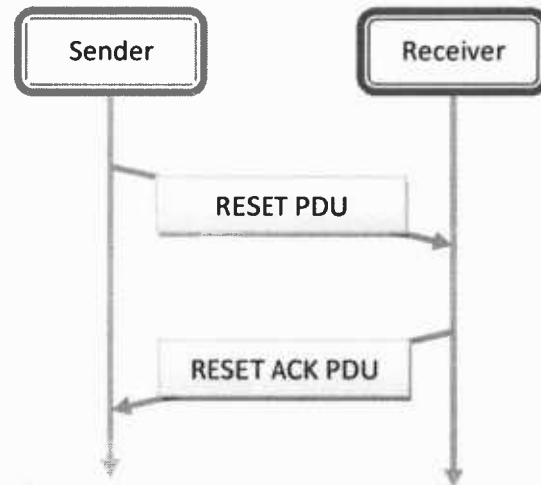


Fig. 10 The RLC Reset procedure

When the Reset procedure is triggered, the Sender enters the RESET_PENDING state (or the RESET_AND_SUSPEND state, if its previous state was LOCAL_SUSPEND) and it stops transmitting any AMD PDUs or Status PDUs and ignores any incoming AMD PDUs and Status PDUs (piggybacked or ordinary). The Sender increments the value of the VT(RST) variable (this variable counts the number of times a Reset PDU is sent). If the value of this variable is lower than the value of the MaxRST parameter then the Sender submits a RESET PDU to the lower layer and starts the Timer_RST. Otherwise, it terminates the Reset procedure and signals an unrecoverable error to the upper layers.

The RESET PDU has a RSN (Reset Sequence Number) field and its size is one bit only. The value of this field is 0 for the first RESET PDU sent (since the RLC entity is established or re-established) and it is incremented (flipped) every time a new RESET PDU is transmitted, but not when a RESET PDU is retransmitted. This is used in order

¹ This was not implemented in our simulation, as we did not implement ciphering.

to detect new reset attempts from older ones that may still have PDUs pending.

When a RESET PDU is received by the Receiver, the Receiver checks its RSN field to determine whether a new Reset procedure has been triggered again or the PDU was sent during the previous (and now complete) Reset procedure. In the latter case, the RSN field of this PDU will have the same value as the one of the last received RESET PDU. If this is the case, the Receiver simply submits a RESET ACK PDU to the lower layer with its contents set exactly as the last transmitted RESET ACK PDU and the entity is not reset.

If the RESET PDU received by the Receiver has an RSN value that is different than the last RESET PDU, and consequently is part of a new Reset procedure, (or if it is the first one the Receiver receives since its establishment or re-establishment) then the

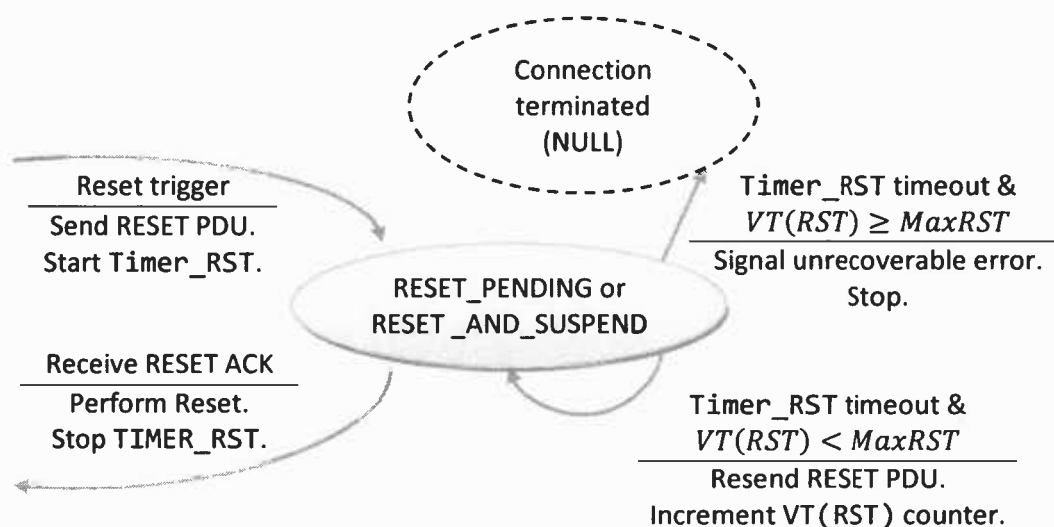


Fig. 11 The Reset procedure for the Sender

Receiver must reset itself. This includes the following:

- Resetting all its state variables (except $VT(RST)$) to their initial values.
- Stopping all timers except $Timer_RST$, $Timer_Discard$, $Timer_Poll_Periodic$ and $Timer_Status_Periodic$.
- Resetting all configurable parameters to their configured values.
- Discarding all RLC PDUs in the receiving side of the AM RLC entity.
- Discarding all RLC PDUs that were transmitted before the reset in the transmitting side of the AM RLC entity.
- Synchronizing its HFN numbers with the ones in the RESET PDU.
- Informing the upper layers of the reset, if configured to do so.
- Submitting a RESET ACK PDU to the lower layer with its RSN field set to the same value as the corresponding received RESET PDU.

When the Sender receives the RESET ACK PDU, it checks its RSN to determine if it corresponds to the last RESET PDU it sent. If it does not, then the PDU is discarded. Otherwise, the Sender resets itself similarly to the Receiver. This includes the follow-



ing:

- Resetting all its state variables to their initial values.
- Stopping all timers except `Timer_Discard`, `Timer_Poll_Periodic` and `Timer_Status_Periodic`.
- Resetting all configurable parameters to their configured values.
- Discarding all RLC PDUs in the receiving side of the AM RLC entity.
- Discarding all RLC PDUs that were transmitted before the reset in the transmitting side of the AM RLC entity.
- Synchronizing its HFN numbers with the ones in the RESET ACK PDU.
- Informing the upper layers of the reset, if configured to do so.

By this time, the Reset function is completed and the Sender returns to its previous state (`DATA_TRANSFER_READY` or `LOCAL_SUSPEND`). However, there are some abnormal cases that are addressed. If the `Timer_RST` expires, indicating that a RESET ACK PDU has not been received since the initiation of the Reset procedure, then the Sender increments the value of the `VT(RST)` variable. If it is lower than the `MaxRST` parameter, then the RESET PDU is retransmitted and the timer is restarted. Otherwise, the Reset procedure is terminated and an unrecoverable error is indicated to the upper layers. Another abnormal case is when a RESET PDU is received by the Sender. In this case, the Sender behaves as if it was the Receiver (by resetting itself as a Receiver and submitting a RESET ACK PDU).

In our simulation we have implemented an option to disable the Reset procedure. When this option is enabled, the reset procedure never initializes. As a result,

- If the Discard function is using the “No_Discard after x number of transmissions” variant, the `VT(DAT)` variable for an AMD PDU can have a value much greater than the `MaxDAT` parameter.
- The variable `VT(MRW)` can have a value much greater than the `MaxMRW` parameter. In this case, the MRW function terminates only upon successful completion (i.e. the acknowledgement of the MRW by the Receiver), as the MRW SUFI is sent indefinitely until it is acknowledged.
- Status PDUs with erroneous sequence numbers are simply dropped.

5.2 Local Suspend function

When this function with parameter N is initiated by the upper layers, the RLC entity does not send AMD PDUs with sequence number greater than or equal to $VT(S) + N$.

The RLC entity can be resumed by upper layers. If the RLC Reset procedure was not ongoing then the entity resumes the data transfer procedure. Otherwise, it removes the suspend constraint and resumes the RLC Reset procedure.



This function was not implemented or used in our simulation since there is no support for such a function from the upper layers in our simulated environment.

5.3 Stop & Continue functions

The RLC entity can be stopped by upper layers. This does not affect its timers. When an RLC entity is stopped, it does not submit any PDUs to lower layers or receive any PDUs and it delays triggered polling functions and status transmissions until it is continued.

This function was not implemented or used in our simulation since there is no support for such a function from the upper layers in our simulated environment.

5.4 Re-establishment function

The RLC entity can be re-established by upper layers. In this function, it resets state variables and configurable parameters and discards the AMD PDUs in the receiving side, among other things.

This function was not implemented or used in our simulation since there is no support for such a function from the upper layers in our simulated environment.

5.5 Reconfiguration of RLC parameters by upper layers

The RLC parameters for an RLC entity can be modified by upper layers. The parameters that can be reconfigured include the Configured_Rx_Window_Size parameter (which affects the VR(MR) variable) and Configured_Tx_Window_Size parameter (which affects the VT(WS) and VT(MS) variables). This function can cause a number of PDUs to be discarded at the UE due to limited memory.

This function was not implemented or used in our simulation since there is no support for such a function from the upper layers in our simulated environment.

6 Results

6.1 Simulation Setup

We used the ns-2 [8] simulator augmented with additional wireless links, link layer protocols and applications [11] for our experiments. We tested HTTP and FTP traffic over wireless links with and without contention. Each test was performed 30 times to compensate for statistical fluctuations, so all results shown are average values and their 99% confidence intervals. We compared the RLC/AM protocol with two variants of SR. We also evaluated the SDU discard policy impact.

We used two error models for the wireless link, a *Uniform* error model ("Cellular" link) and a *Two State* error model ("PCS" link), representing the error models on the wireless link of a cellular mobile network. For each error model, we simulated the client-server topology shown in Fig. 12.

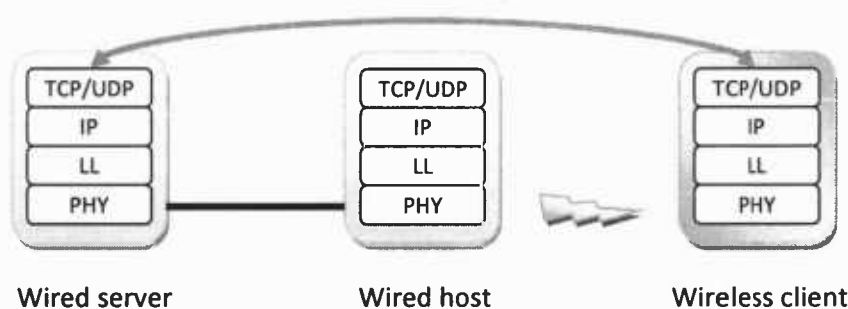


Fig. 12 The simulation topology

The wireless client connects to a wired server via a wired host. The wired channel can be either a LAN connection (with 10 Mb data rate and 1 ms delay) or a WAN connection (with a 2 Mb data rate and 50 ms delay). The wireless link supports a 64 Kbps data rate with a 50 ms delay and a 250 byte frame plus a header. In the Uniform error model for the wireless link there is independent frame loss at rates 1.5%, 2.5%, 5.4% and 9.8%. In the Two State error model, the channel can be in a good state, with a bit error rate of 10^{-6} , or a bad state, with a bit error rate of 10^{-2} . Both states have exponential durations, with the average duration of the good state being 10 s and the average duration of the bad state being 100 ms, 200 ms, 500 ms or 1000 ms. It has been found experimentally that with these parameters the average frame loss rate of the two-state model is 1.5%, 2.5%, 5.4% and 9.8%, matching the Uniform model. The TCP flavor used has the Reno variant, the most widely used variant in the Internet.

The RLC/AM parameters used in our experiments are summarized in the following table. These parameters were found with direct experimentation to provide good results in most applications and error models. The main difference between the two error models was the addition of the "Every last PDU in retransmission buffer" polling

trigger and the Status Prohibit function in the Two State model. It was found experimentally that that delaying the Status reporting and making sure a status request was sent at the end of every retransmission phase during bad channel states apparently increases the throughput of the TCP applications tested. For an explanation of these parameters, see *RLC/AM parameters*, pg. 52.

Parameter	Uniform model ("Cellular" link)	Two State model ("PCS" link)
Window size (Sender & Receiver)	128 frames	128 frames
Poll triggers	Timer_Poll, Window Based	Timer_Poll, Window Based, Every last PDU in retransmission buffer
Status report triggers (other than polling)	Detection of missing PDU, Periodic status reporting	Detection of missing PDU, Periodic status reporting
Poll Prohibit enabled	Yes	Yes
Status Prohibit enabled	No	Yes
RESET enabled	No	No
Poll Window	70%	80%
SDU Discard Mode	SDU discard after x number of transmissions	SDU discard after x number of transmissions
MaxDAT	3	2
Timer_Status_Prohibit timeout	-	90 ms
Timer_Status_Periodic timeout	400 ms	500 ms
Timer_Poll timeout	200 ms	200 ms
Timer_Poll_Prohibit timeout	100 ms	100 ms
Timer_MRW timeout	500 ms	110 ms
Status report piggybacking enabled	No	No

To evaluate the RLC/AM protocol, we used Web Browsing (HTTP) and FTP file transfer, two of the most popular applications on the Internet, both with and without contention. In Web Browsing, the client accesses pages containing text, embedded objects and links to other pages, stored on the server. The client-server interaction consists of transactions (i.e. the request from the client and the response, with the web page and embedded objects, from the server). Only one transaction is in progress at any time, with no pauses between transactions. The performance metric used was the throughput for the server-to-client data, defined as the amount of all application data transferred from the server to the client divided by the time required for the transfer.

All results shown reflect the state at the end of the last completed transaction during the simulated period, which was 2000 s.

The file transfer (FTP) application simulated a file transfer from the server to the client. This application is unidirectional, with only TCP ACKs in the reverse direction, and it sends data as fast as possible. A 10 MB file was transferred and we measured application throughput, defined as the amount of application data transferred divided by time required for the transfer.

Contention was simulated with a UDP real-time Media Distribution application, in which a speaker sends audio or video and alternates between *talking* and *silent* states with exponential durations, averaging 1 s and 1.35 s respectively. Media is transmitted only in the talking state, at a constant bit rate of 56 Kbps. On average, this application consumes 37.5% of the available bandwidth. Since it is delay sensitive, it does not use the reliable link layer protocol used by the other two applications.

The RLC/AM protocol is evaluated by comparison to two variants of the well-known Selective Repeat protocol, which has been found to offer excellent performance for TCP applications [12], without requiring TCP awareness. In Selective Repeat, the Sender transmits frames in sequence within a transmission window of N frames and buffers them in case they need to be retransmitted. The peer entity receives the frames sent by the Sender and forwards them to the upper layers if they are received in-sequence. The Receiver uses ACKs to inform the Sender that all frames up to (and including) the one being acknowledged have been successfully received and the Sender deletes the acknowledged frames from its buffers and advances its window (this is similar to how RLC/AM operates as well). If the Receiver receives a frame out of sequence (and a gap in the sequence is detected), this frame is buffered and, in our implementation, a negative acknowledgement is transmitted to the sender for each missing frame. The Sender retransmits each NACKed frame. ACKs are delayed for a short interval in order to be piggybacked into a data frame, to reduce protocol overhead. The Sender uses a timer for each data frame sent and if the timer expires and the frame has not been acknowledged, it is retransmitted. This is so that the Sender shall not exhaust its window waiting for ACKs or NACKS that may have been lost. Our variant also supports *multireject*, allowing each missing frame to be NACKed multiple times.

The first variant of SR we tested uses a fixed retransmission timer with timeout value of 1.1 s. However, variants with adaptive timers have been found to perform better [13]. Adaptive timers are based on the same principle used in TCP's congestion control algorithm. An adaptive (self-clocking) SR protocol monitors the average round-trip time and its variation and dynamically changes the timeout value of its retransmission timer. A number of policies exist for predicting the round trip time [14]. Based on previous research, the second variant of SR that we tested is an adaptive SR and it uses the $3 \times srtt + 2 \times srttvar$ policy for the Uniform error model and $4 \times srtt + 0 \times srttvar$ for the Two State error model.

6.2 Comparison with other protocols

The RLC/AM protocol clearly improves throughput in all cases when compared to the raw link. We shall examine more closely its behavior for every application and link. Note that the diagrams for the WAN topology are in the appendix (pg. 56).

6.2.1 Applications without contention

In this section, we examine the performance of Web Browsing and FTP applications without contention. For Web browsing, it is clear that the RLC/AM protocol is not as efficient as the SR variants. When using the Uniform error model ("Cellular" link), we notice a significant drop in throughput compared to the SR variants, even when the frame loss rate is

only 1.5%. As the frame loss rate is increased, the difference between fixed and adaptive SR is increased as well and by the time the error rate reaches 9.8% the RLC/AM tends to have similar performance with the fixed SR. Of course, the adaptive SR protocol has higher throughput compared to the other two for all error rates.

The situation is more dramatic when using the Two State error model ("PCS" link). The RLC/AM

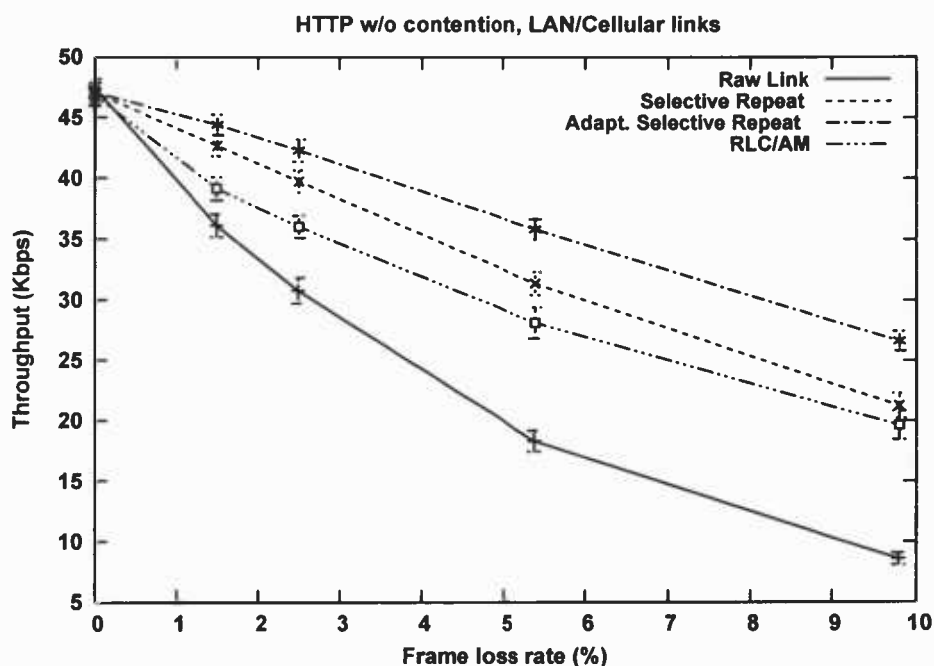


Fig. 13 Web browsing throughput without cont., Uniform error model, LAN topology

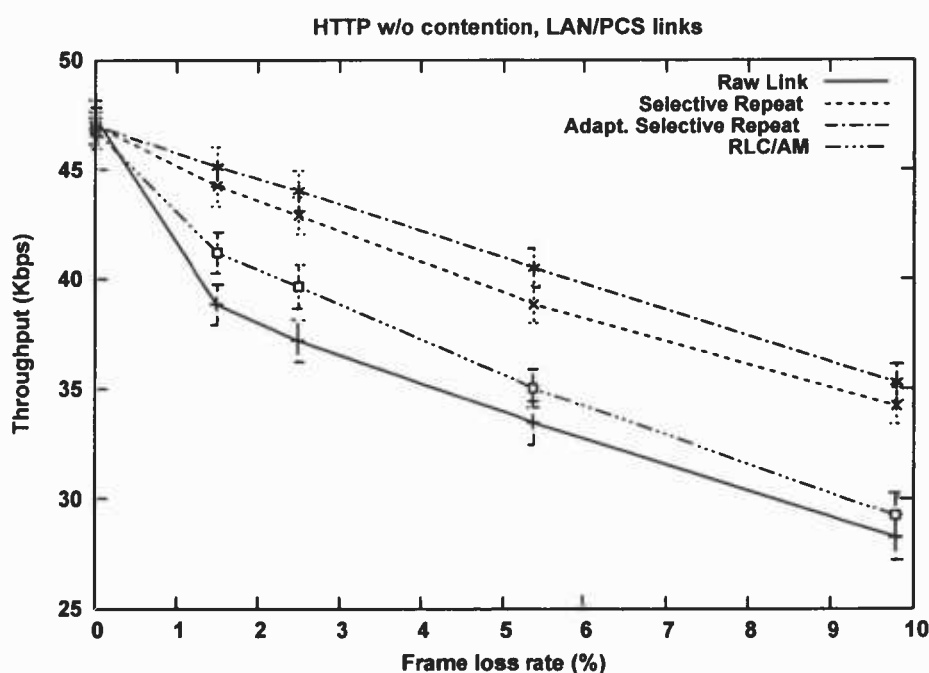


Fig. 14 Web browsing throughput without cont., Two State error model, LAN topology

has a slightly lower throughput even with no frame loss because of its significantly higher overhead. When the frame loss rate reaches the 1.5% point, the RLC/AM throughput takes a steep dive and it loosely follows the performance of the raw link, while the SR throughput (both variants) has a simple, almost linear drop in performance as the frame loss rate increases for both variants. In fact, the RLC/AM

throughput when using a LAN wired link tends to drop to as low as the raw link, as the frame error rate increases, and it is almost half the throughput of the SR protocols. It is obvious that, although the RLC/AM slightly improves the situation when compared to the raw link (especially when using a WAN connection for the wired link), it is no match for the SR variants, at least in HTTP traffic.

This impression is visibly improved when evaluating the RLC/AM in the file transfer (FTP) application. In the Uniform error model, while the RLC/AM performs about the same as the SR variants for 1.5% and 2.5% frame error rates, it

sports better throughput than both SRs as the error rate reaches 5.4% and 9.8%. In this case, the SRs seem to have a higher drop rate as the frame loss rate increases. This behavior applies to both LAN and WAN wired links.

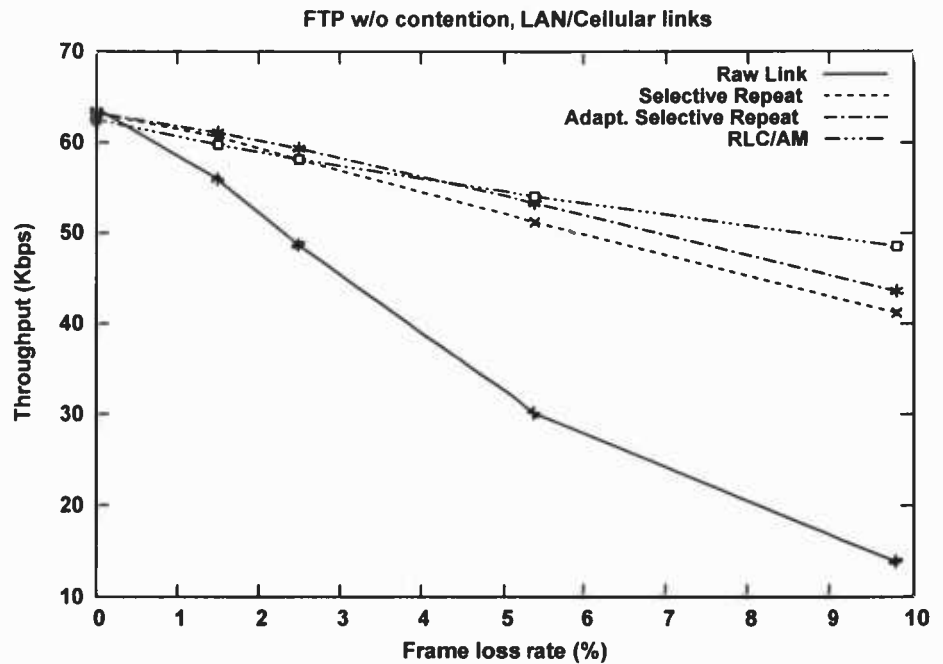


Fig. 15 File transfer throughput without cont., Uniform error model, LAN topology

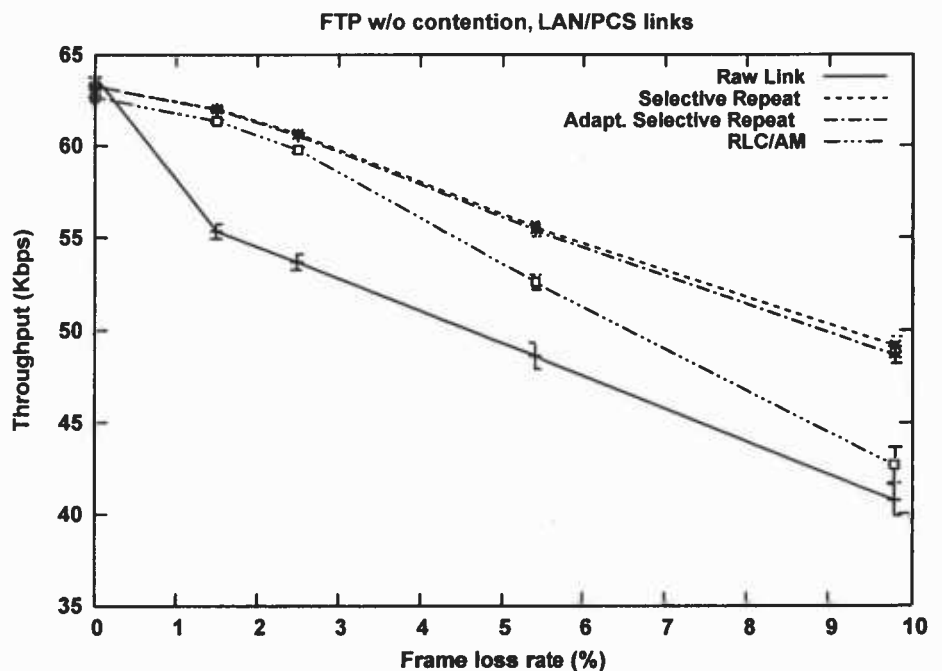


Fig. 16 File transfer throughput without cont., Two State error model, LAN topology

With the Two State error model, the RLC/AM throughput was noticeably worse. Both SR protocols had an almost identical throughput as the error rate increased, with a seemingly linear drop. The RLC/AM had a slightly lower throughput in 1.5% and 2.5% frame error rates. The difference with the other two protocols increased considerably in higher error rates, with the RLC/AM throughput tending to fall to the raw channel throughput. Again, this behavior applies to both LAN and WAN wired links.

A final observation for all TCP applications tested was that throughput is better in the Uniform error model than in the Two State error model, as expected [6].

6.2.2 Applications with contention

We also tested the two SR variants and the RLC/AM protocol with the same applications contenting with a CBR source.

The protocols behaved as before when running the Web Browsing simulation with the Uniform error model. Apparently, the CBR source effect was minimal on throughput. One thing we noted was that the RLC/AM throughput was closer to that of

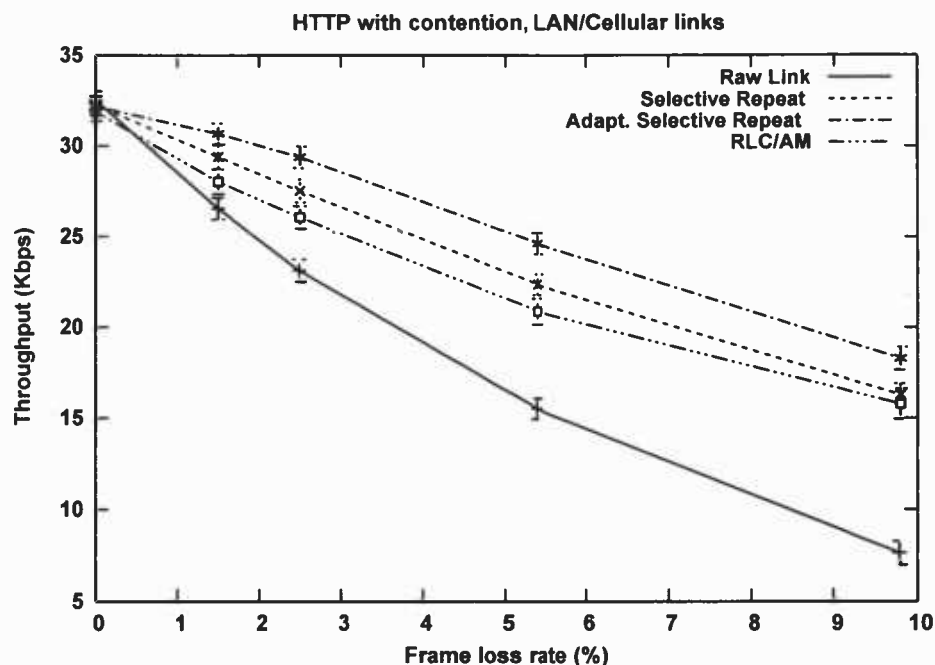


Fig. 17 Web browsing throughput with contention, Uniform error model, LAN topology

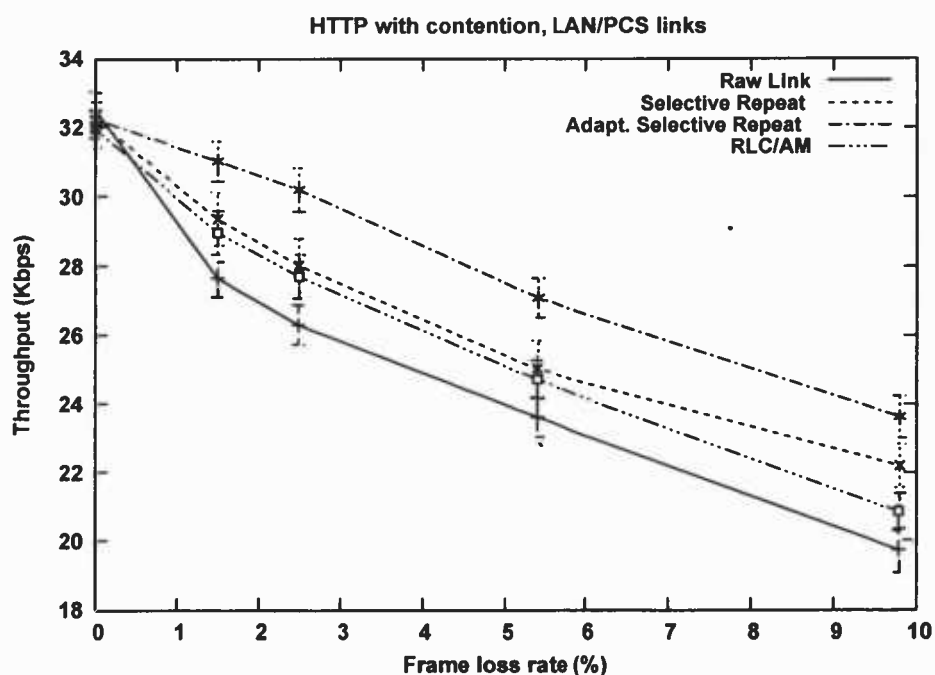


Fig. 18 Web browsing throughput with cont., Two State error model, LAN topology

the fixed SR protocol this time. Although the RLC/AM is obviously an improvement compared to the raw link, its throughput is still much lower than the SR variants throughput. Once again, the RLC/AM protocol throughput takes a steeper dive (than the SR protocols) while the frame error rate increases up to 5.4%, after which point it tends to recover and perform almost as good as the fixed SR variant.

When using the Two State error model for the wireless channel and a LAN connection, both the RLC/AM and the fixed Selective Repeat take a sharp dive while the frame error rate increases, but the fixed SR seems to recover after the FER reaches 5.4%. When using a WAN connection, the fixed SR performed much better than the RLC/AM.

File transfer throughput behavior with the Uniform error model remains relatively unchanged for the RLC/AM when compared to the behavior without contention. Again, the RLC/AM protocol starts slightly lower than the SR variants because of its high overhead, but it appears to have better performance than both of them as the frame error rate increases. This is for both WAN and LAN wired connections. We also note the lower throughput of the SR protocol this time.

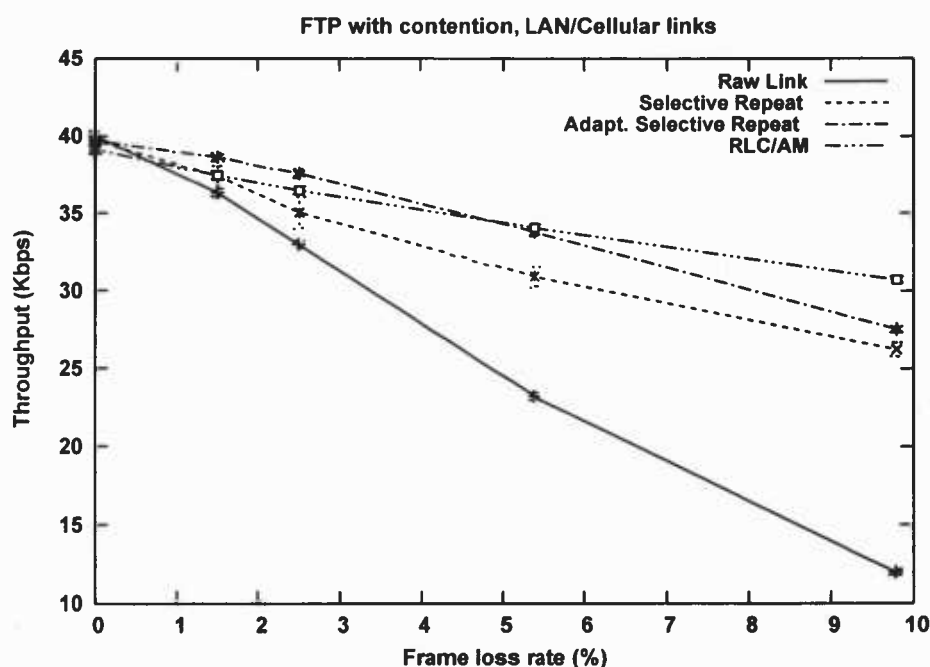


Fig. 19 File transfer throughput with contention, Uniform error model, LAN topology

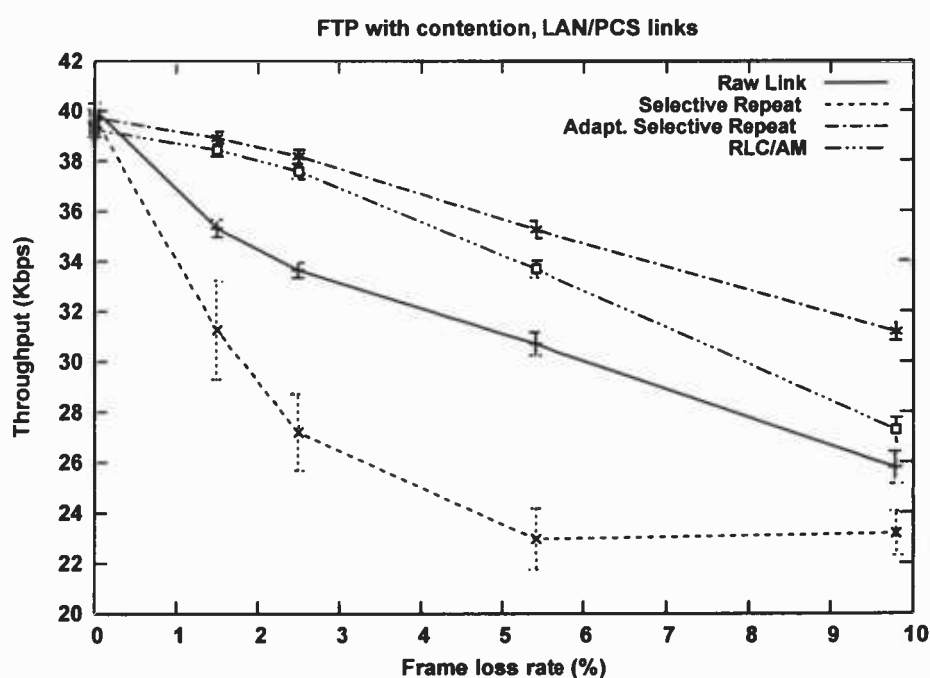


Fig. 20 File transfer throughput with contention, Two State error model, LAN topology

When we used the Two State error model for the wireless connection, one thing that stood out was the very poor throughput performance of the fixed SR protocol, which was even lower than the raw channel in the LAN topology. Again, the adaptive SR has the highest throughput, in both WAN and LAN links. The RLC/AM has an acceptable throughput as long as the frame error rate is up to 5.4%, but its throughput falls close to that of the raw channel in higher frame error rates.

The application for Continuous Media Distribution we simulated uses UDP for delivering the media. As a result, it bypasses the reliable link layer protocol (RLC/AM or a SR variant) and uses the raw channel directly. In this case, we measure the average delay of each packet to see how well the media application would perform while the user is surfing the web or downloading a file from the server using FTP at the same time.

In all cases, none of the protocols achieve as little delay as the raw link. However, there seems to be a relative scale, in which the RLC/AM has the lowest delay, followed closely by the adaptive SR protocol while the fixed SR protocol has the highest delay of them all. When using the Uniform error model on the wireless link with the FTP application, the fixed SR protocol clearly has

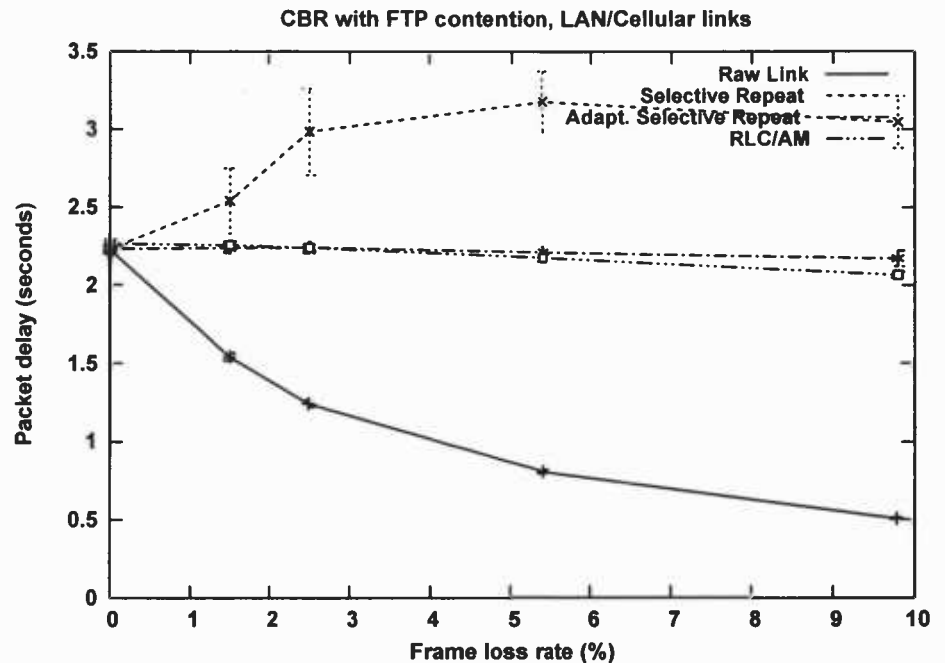


Fig. 21 CBR delay with FTP contention, Uniform error model, LAN topology

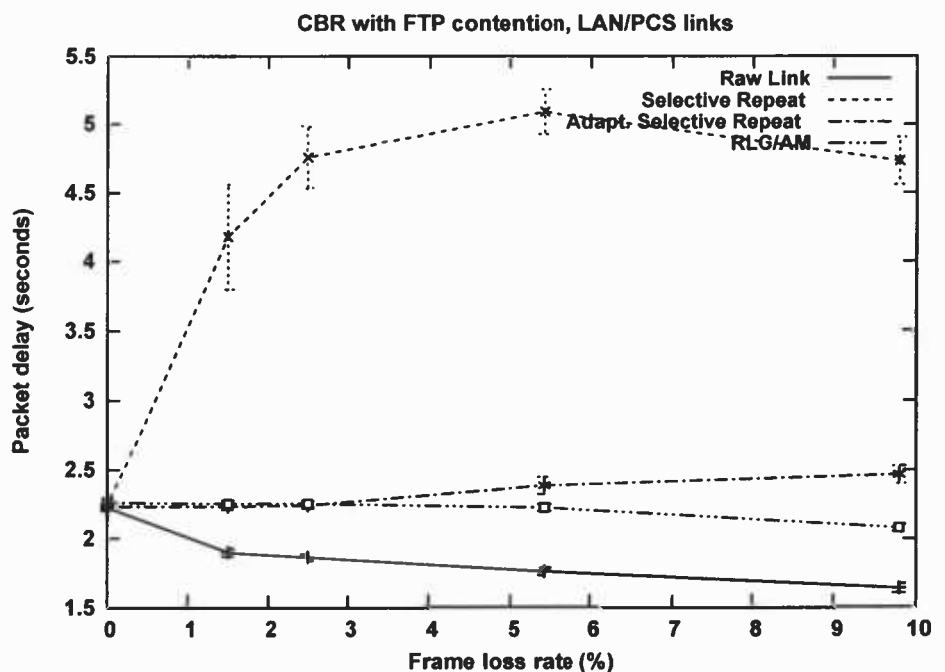


Fig. 22 CBR delay with FTP contention, Two State error model, LAN topology

the highest delay, which increases as the frame loss rate increases. Meanwhile, the RLC/AM and adaptive SR have an almost constant delay, which is significantly lower than the fixed SR and the delay of the RLC/AM appears to be slowly decreasing as the frame error rate increases.

When using the Two State error model with the FTP application, the previous conclusions are far more pronounced. The increase of the fixed SR delay is steeper and the maximum delay is almost double as before. This time, the adaptive SR delay is also increasing as the frame error rate increases when the frame error rate reaches 2.5%, albeit much more slowly compared to the fixed variant. Its delay appears to be the same as the delay of the RLC/AM protocol for frame error rate up to the 2.5%. Interestingly, the RLC/AM delay seems to be decreasing again as the frame error rate increases

(especially from the 5.4% onwards). The results are similar for the LAN and WAN topologies (apart from the fact that the fixed SR protocol delay increases a bit more slowly in the WAN topology).

When testing the HTTP application with the Uniform model, the RLC/AM still causes the lowest delay to the CBR stream. The delay is slowly decreasing as the frame

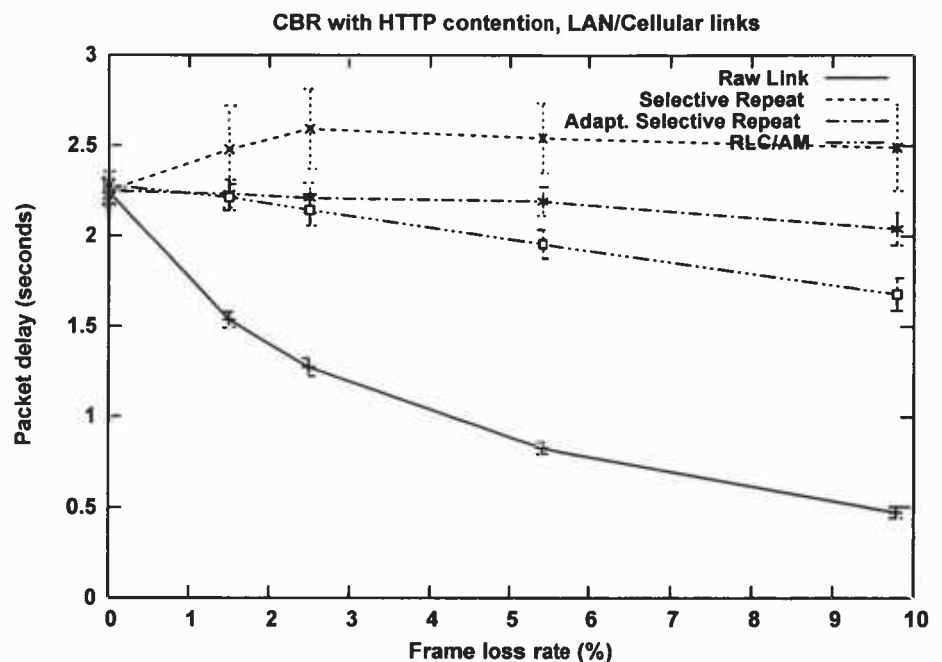


Fig. 23 CBR delay with HTTP contention, Uniform error model, LAN topology

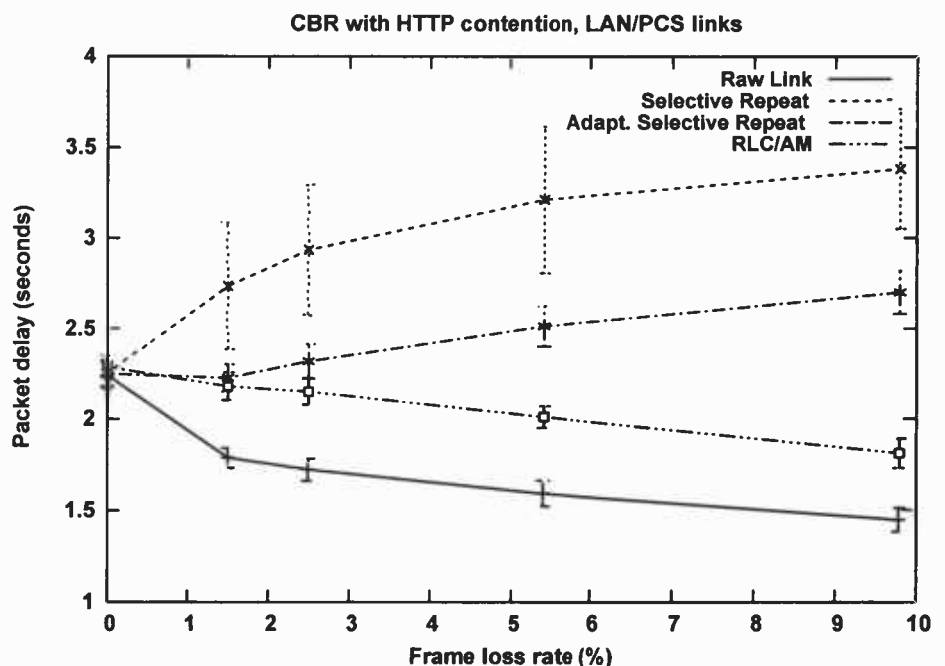


Fig. 24 CBR delay with HTTP contention, Two State error model, LAN topology

error rate increases. The adaptive SR protocol appears to have very slowly decreasing delay, while the fixed SR is the only one that has a higher delay than the others, which appears to be relatively constant after the 2,5% FER point. This is for both WAN and LAN topologies.

The delay for the HTTP application with the Two State model on the wireless link is (relatively) similar to the Uniform model. The main difference is that the delay of the fixed and adaptive SR variants has a steeper increase as the frame loss rate increases, with the adaptive SR delay tending to be as high as the fixed SR delay, when the FER increases. The RLC/AM delay is decreasing again (more sharply this time) as the error rate increases.

One final observation for RLC/AM concerning both TCP applications and all

error models and topologies is that the media stream delay is lower when the throughput for the TCP application is low too. When the TCP application has a high throughput, the media stream packet delay is (slightly) increased as well.

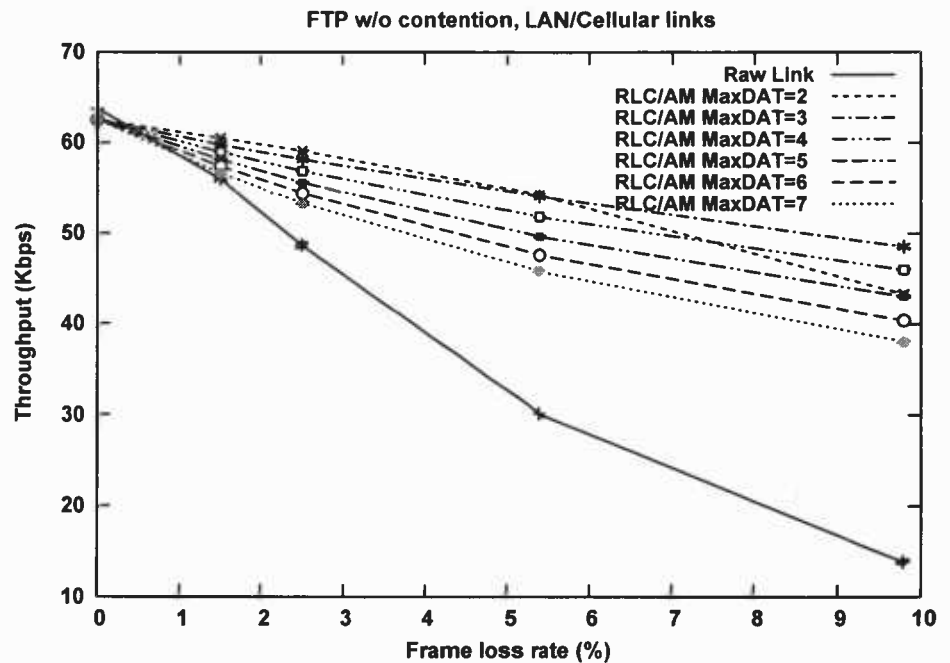


Fig. 25 File transfer throughput without cont., Uniform error model, LAN topology

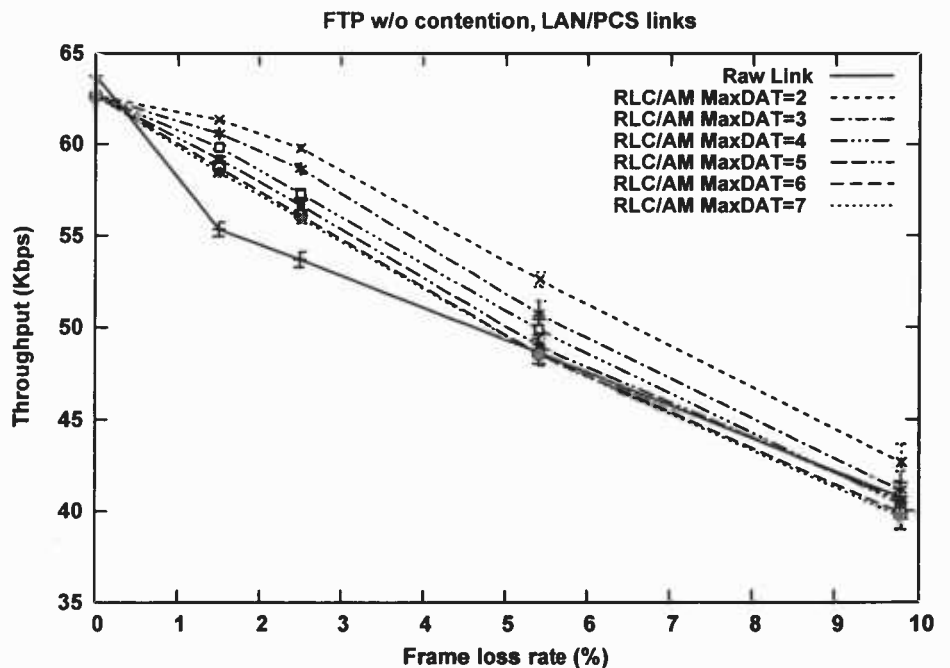


Fig. 26 File transfer throughput without cont., Two State error model, LAN topology

6.3 SDU discard policies

We tested another aspect of the RLC/AM protocol, namely the behavior of two of the SDU discard variants: the timer-based discard and the SDU discard based on number of transmissions. We used only the LAN topology without contention to the TCP applications. We tried several values of the MaxDAT parameter and several values of the Timer_Discard timeout value (see

RLC/AM parameters, pg. 52). We kept all other parameters the same. When using the "SDU discard after x transmissions" variant, the highest throughput was achieved for MaxDAT=3 in the case of the Uniform error model and MaxDAT=2 for the Two State error model. This means that only one retransmission is allowed in the Uniform model, and no retransmissions in the Two State model. In general, as the number of retransmissions

increased, throughput decreased as it conflicts with TCP retransmissions. In the Uniform model, it is interesting to note that for MaxDAT=2 we get marginally higher throughput (than the value MaxDAT=3) for up to 5,8% frame error rate, and the throughput drops significantly for higher error rates.

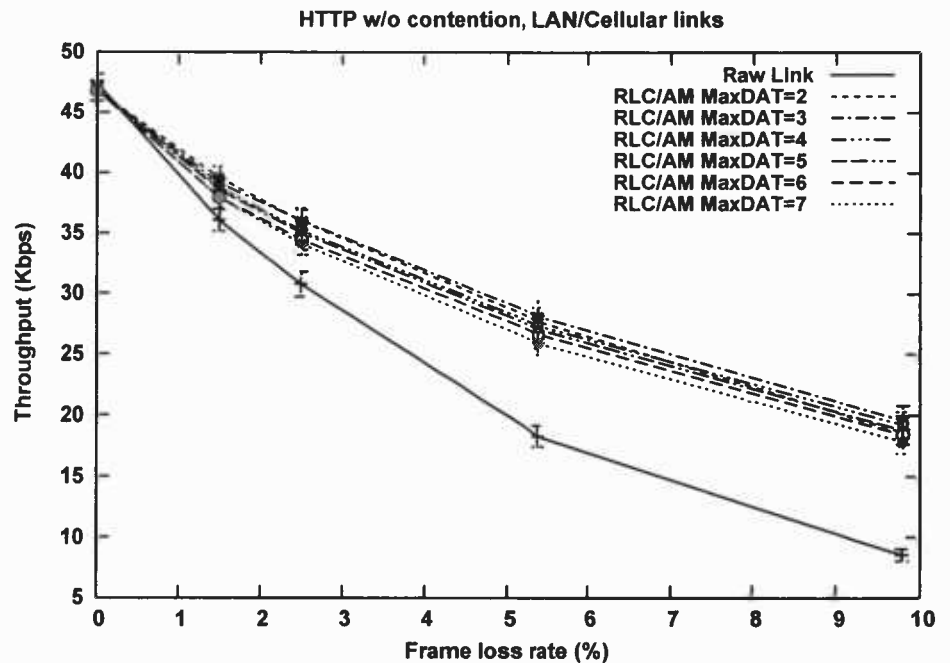


Fig. 27 Web browsing throughput without contention, Uniform model, LAN topology

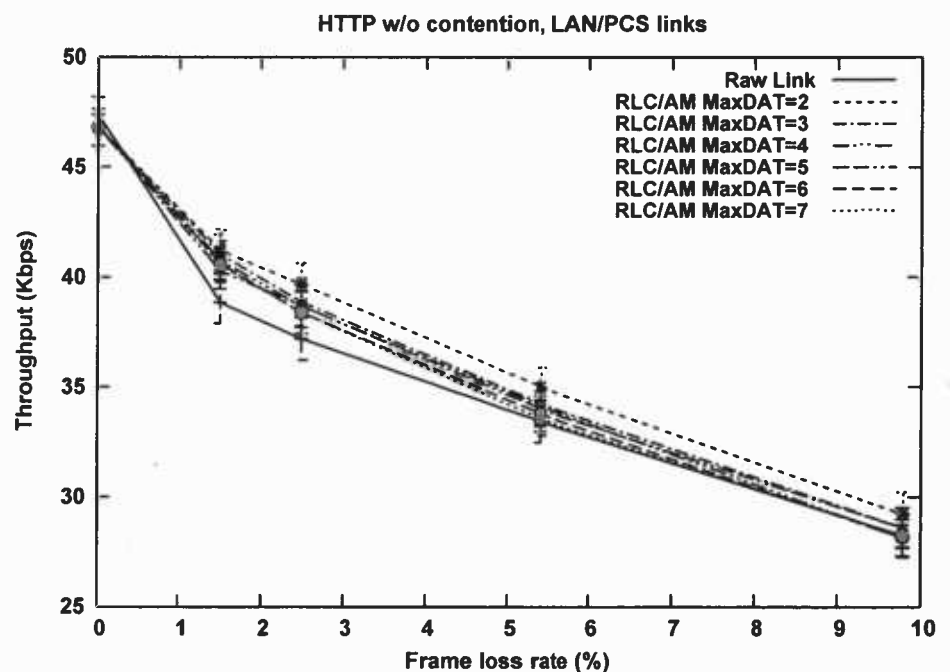


Fig. 28 Web browsing throughput without contention, Two State model, LAN topology

The results are similar for the Web Browsing application as well. For the Uniform error model we get the maximum throughput for MaxDAT=3 once more. The value MaxDAT=2 has a lower throughput and for values greater than 3, throughput drops as the number of re-transmissions increases. For the Two State model we get the maximum throughput for MaxDAT=2 and the throughput drops as the number of re-transmissions increases.

We also note that, at least for the Uniform model, the throughput increases as the number of re-transmissions increases in the link layer, then it reaches a maximum (MaxDAT=3) and then decreases, as it conflicts with the TCP re-transmissions.

When using the timer-based discard variant, the first thing we notice is that throughput is significantly worse than the previous variant for both TCP applications. When testing the FTP application with the Uniform error model, we notice that for all values of the timeout parameter tested, throughput is worse than even the raw link, at least for up to 5.4% frame error rate. After the 5.4% point, RLC/AM throughput is higher than the throughput of the raw link for almost all values of the timeout parameter (except the lowest value tested, 0.1s). In general, performance is improved when using larger values for the Ti-

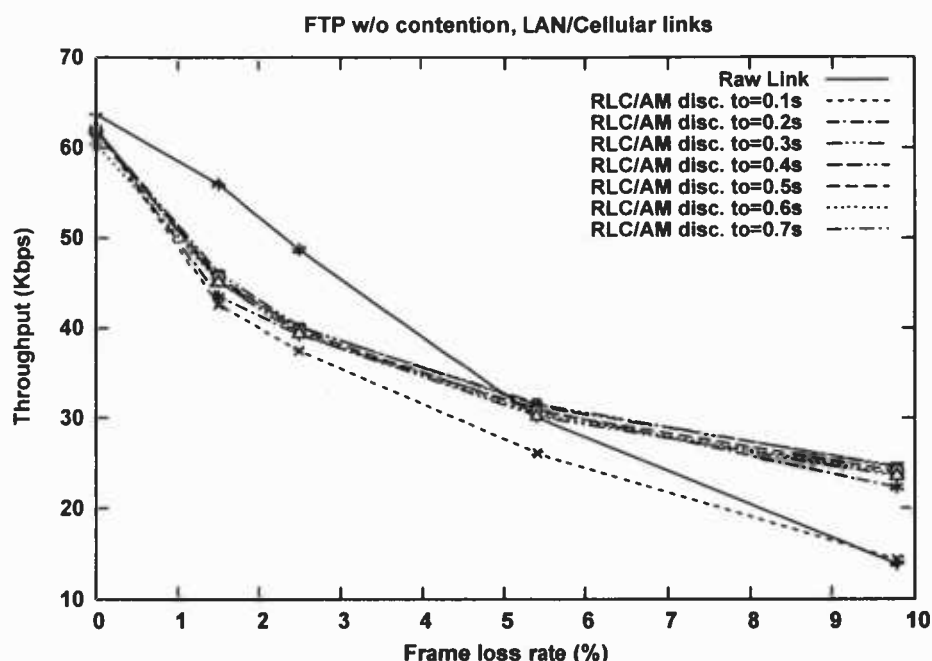


Fig. 29 File transfer throughput without cont., Uniform error model, LAN topology

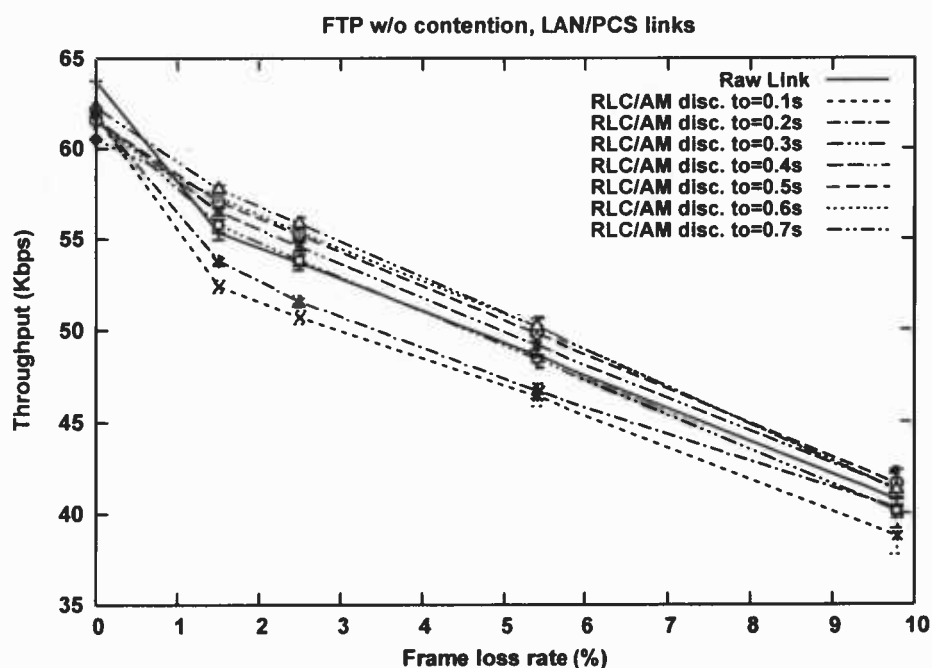


Fig. 30 File transfer throughput without cont., Two State error model, LAN topology

mer_Discard timeout, although the improvement is quite small and in fact almost any timeout value between 300 ms and 700 ms has similar throughput.

For the FTP application with the Two State error model, throughput is almost the same as the raw link. Lower values of the timeout parameter

(up to 300 ms) have slightly worse throughput than the raw link while higher values (up to 600 ms) seem to improve the throughput. The value of 700 ms appears to have the highest throughput up to the 5,8% point of the FER, and its throughput is lower than the 600 ms value for higher frame loss rates.

HTTP throughput on the Uniform model is quite similar to FTP throughput. For all values tested, perfor-

mance is worse than the raw link for up to 5,4% frame loss rate. From that point onwards, timeout values greater than 200 ms seem to have better throughput than the raw link, with throughput increasing as the timeout value increases.

Web browsing throughput with the Two State error model is again similar to the FTP throughput. Once more, RLC/AM throughput follows very closely the throughput of the raw channel, with timeout values up to 300 ms having lower throughput than the raw link and higher values having throughput higher than the raw link. In general, performance is improving for higher timeout values; yet high timeout values have

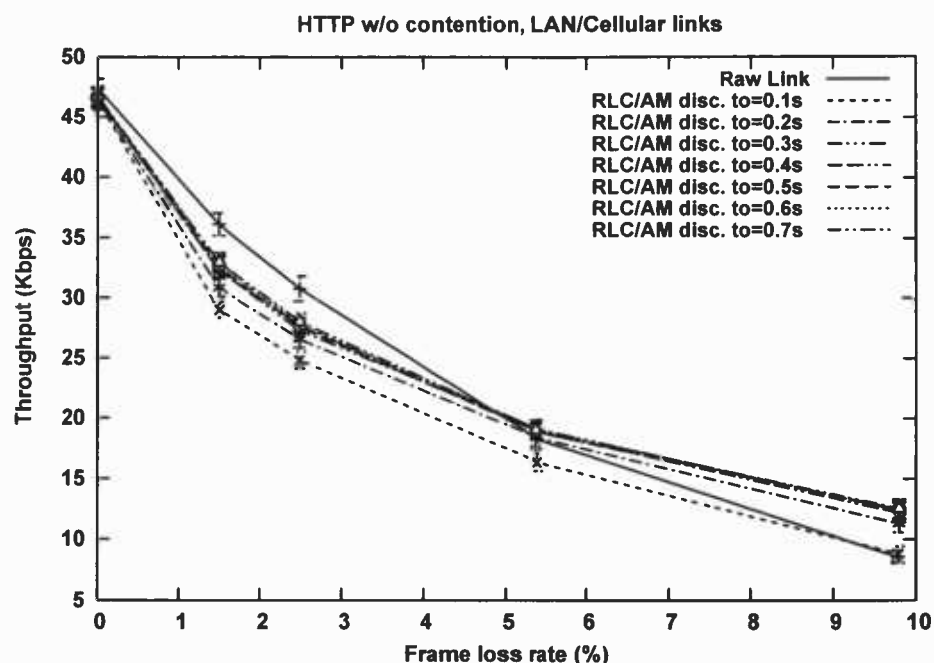


Fig. 31 Web browsing throughput without cont., Uniform model, LAN topology

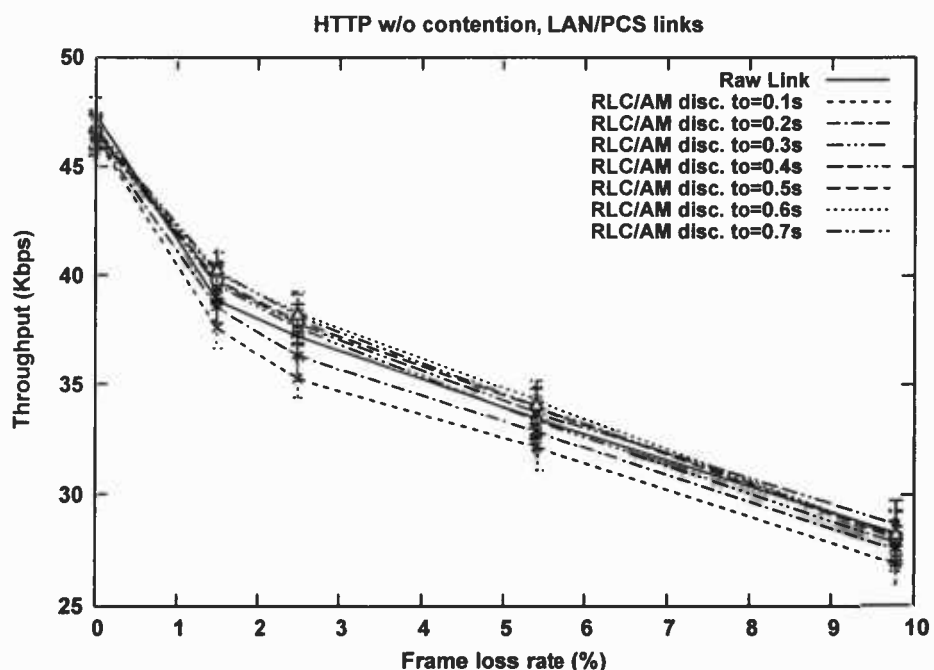


Fig. 32 Web browsing throughput without cont., Two State model, LAN topology

worse performance in high frame loss rates (9.8%).

To conclude, as expected [2] [5] the timer-based discard variant performs significantly worse in all error models and TCP applications we tested when compared to the "discard after x transmissions" variant. The difference is higher in the Uniform model. In the appendix (at pg. 60) you can find the diagrams comparing the two discard policies. These diagrams are the same ones presented in this section but each diagram shows the throughput from both policies.

6.4 Conclusions

From the previous results, we can conclude that the RLC/AM protocol is well suited for applications with contention. Although the TCP application will have a lower throughput, as the channel is not overloaded, the media application using UDP will experience the lowest packet delay when compared to the other protocols as the frame error rate increases. Still, the adaptive Selective Repeat protocol is preferred if we factor in the TCP application throughput. We also noticed that the RLC/AM throughput for the Web Browsing application is significantly lower than the two SR variants. The RLC/AM performs better than SR when transferring large files over a link with Uniform error model and its performance is acceptable when transferring files over a link with the Two State error model, as long as the frame error rate does not get too high.

As a result, the RLC/AM protocol is dependent on the application used. It sports much better performance when using FTP than Web Browsing. Contention from the CBR stream seemed to have minimal impact on the RLC/AM throughput in all applications and it causes the lowest delay on the CBR stream when compared with the SR variants.

There seem to be no significant differences between the LAN and WAN topologies for all protocols, when using TCP applications without contention. Web browsing throughput is lower in the WAN topology due to the higher delay of the interactive application but the relative performance of all protocols remains the same. In applications with contention, the only substantial difference between LAN and WAN topologies is that the fixed SR protocol is performing worse in the LAN topology compared to the other two protocols (especially when using the Two State error model).

Compared to the raw link, the RLC/AM protocol improves the situation in every case tested when evaluating TCP applications. Naturally, the RLC/AM delay on the CBR stream is not as low as the raw link, but it is the lowest compared to the other protocols. The fixed SR protocol has a higher throughput than RLC/AM in HTTP (all cases) and FTP with the Two State model and lower throughput in FTP with the Uniform model or with contention. The fixed SR also causes the highest delay on the CBR link compared to all the other protocols. The adaptive SR protocol has the highest throughput in all applications, with or without contention, compared to the other two protocols. The only exception is the FTP application with the Uniform error model. The adaptive SR delay on the CBR stream is generally low, but not as low as the RLC/AM.

The "SDU discard after x transmissions" variant of the RLC/AM protocol clearly offers much better throughput when compared to the timer-based discard mode. In general, throughput is increased as the number of retransmissions is increased (when using the "SDU discard after x transmissions" variant), until it reaches a maximum (which, incidentally, in the case of the Two State model is the value $\text{MaxDAT}=2$ – the very first value of the MaxDAT variable). Then throughput drops as retransmissions are increased, as there is conflict with the retransmissions at the transport layer. As far as the timer-based discard variant is concerned, throughput was lower or (at best) equal to the raw link with the parameters we used. Still, a different setup for the timer-based discard policy is not expected to provide a significant increase in performance for TCP applications.



Appendix

RLC/AM features implemented

The following features of the RLC/AM protocol have been implemented in our simulation:

Functions

- ☐ Segmentation and reassembly
- ☐ Concatenation
- ☐ Padding
- ☒ Transfer of user data
- ☒ Error correction
- ☒ In-sequence delivery of upper layer PDUs
- ☐ Out-of-sequence delivery of upper layer PDUs
- ☒ Duplicate detection
- ☐ Flow control (using the WINDOW SUFI)
- ☒ Sequence number check
- ☒ Protocol error detection and recovery
- ☐ Ciphering

Procedures

- ☒ Data Transfer
- ☒ Poll & Poll Prohibit
- ☒ Status Report (ACKs & NACKs) & Status Prohibit
- ☒ SDU Discard
- ☒ Reset
- ☐ Local Suspend
- ☐ Stop & Continue
- ☐ Re-establishment
- ☐ Reconfiguration of RLC parameters by upper layers

Poll triggers

- ☐ Every last PDU in buffer
- ☒ Every last PDU in retransmission buffer
- ☒ Poll Timer
- ☒ Every x PDU
- ☒ Every x SDU
- ☒ Window Based
- ☒ Timer Based



Status Report Triggers

- ☒ Polling
- ☒ Detection of Missing PDU(s)
- ☒ Timer based status report transfer
- ☐ Request from lower layers

SDU Discard Operation Modes

- ☒ Timer based discard with explicit signaling
- ☒ STU discard after x number of transmissions
- ☒ No_discard after x number of transmissions

Status PDU SUFIs

- ☒ NO_MORE
- ☒ ACK
- ☐ LIST
- ☒ BITMAP
- ☐ RLIST
- ☒ MRW
- ☒ MRW_ACK
- ☐ WINDOW

Protocol States

- ☒ NULL
- ☒ DATA_TRANSFER_READY
- ☒ RESET_PENDING
- ☐ LOCAL_SUSPEND
- ☐ RESET_AND_SUSPEND

Other features

- ☒ Piggybacked Status PDUs

RLC/AM parameters

The behavior of the RLC protocol can be controlled via a set of parameters. The following table summarizes the most important parameters of the RLC/AM protocol and gives the name of the parameter in our implementation.

Name in spec.	Impl. Name	Description
MaxDAT	max_dat_	The maximum number of transmissions of an AMD PDU is equal to MaxDAT-1. This parameter represents the upper limit of a VT(DAT) state variable. When a VT(DAT) variable reaches the value of MaxDAT, the Discard function or Reset procedure is initiated. See <i>Limited Reliability of the RLC/AM</i> , pg. 27.
Poll_PDU	poll_pdu_	This protocol parameter indicates how often a poll is triggered by the Sender, when "Polling every x PDUs" is configured (see <i>Polling</i> , pg. 22). It represents the upper limit for the state variable VT(PDU). When VT(PDU) equals Poll_PDU a poll is triggered.
Poll_SDU	poll_sdu_	This protocol parameter indicates how often a poll is triggered by the Sender, when "Polling every x SDUs" is configured (see <i>Polling</i> , pg. 22). It represents the upper limit for the state variable VT(SDU). When VT(SDU) equals Poll_SDU a poll is triggered.
Poll_Window	poll_window_	This parameter indicates when a poll is triggered by the Sender when "window-based polling" is configured (see <i>Polling</i> , pg. 22). A poll is triggered if the transmission window percentage is greater than the Poll_Window parameter.
MaxRST	max_rst_	This parameter indicates the maximum number of transmissions for a RESET PDU. More specifically, the maximum number of transmissions for a reset PDU is equal to MaxRST-1 and the parameter represents the upper limit of the VT(RST) state variable. When the value of the VT(RST) variable is equal to or greater than the MaxRST value, an unrecoverable error is indicated to the upper layers.
Configured-	rlcamsw (in skele-	This parameter indicates the maximum allowed

_Tx_Window- _Size	ton.tcl), c_txrw	transmission window size and the value for the state variable VT(WS).										
Configured - Rx_Window- _Size	rlcamrw (in skele- ton.tcl), c_rxrw	This parameter indicates the reception window size.										
-	sduDiscardMode	<p>This parameter indicates the variant used for the SDU Discard function. The possible values are:</p> <table><tr><th>Value</th><th>Mode</th></tr><tr><td>0</td><td>Timer based discard with explicit signaling.</td></tr><tr><td>1</td><td>Timer based discard without explicit signaling. This variant is not applicable for AM and thus not implemented. If this value is set, SDU discard is disabled. This value should <i>not</i> be used.</td></tr><tr><td>2</td><td>SDU discard after x number of transmissions.</td></tr><tr><td>3</td><td>No_discard after x number of transmissions.</td></tr></table>	Value	Mode	0	Timer based discard with explicit signaling.	1	Timer based discard without explicit signaling. This variant is not applicable for AM and thus not implemented. If this value is set, SDU discard is disabled. This value should <i>not</i> be used.	2	SDU discard after x number of transmissions.	3	No_discard after x number of transmissions.
Value	Mode											
0	Timer based discard with explicit signaling.											
1	Timer based discard without explicit signaling. This variant is not applicable for AM and thus not implemented. If this value is set, SDU discard is disabled. This value should <i>not</i> be used.											
2	SDU discard after x number of transmissions.											
3	No_discard after x number of transmissions.											
		See <i>Limited Reliability of the RLC/AM</i> , pg. 27.										
MaxMRW	max_mrw_	The maximum number of transmissions of an MRW SUFI is equal to MaxMRW. This parameter prepresents the upper limit for the state variable VT(MRW). When VT(MRW) equals the value of MaxMRW the RLC Reset procedure is initiated.										
-	pTrg_lastPduInRtx- Buf_	This Boolean parameter indicates whether “Every last PDU in Retransmission buffer” polling is configured. See <i>Polling</i> , pg. 22.										
-	pTrg_pollTimer_	This Boolean parameter indicates whether “Poll Timer” polling is configured. See <i>Polling</i> , pg. 22.										
-	pTrg_everyPPPdu_	This Boolean parameter indicates whether “Every x PDU” polling is configured. See <i>Polling</i> , pg. 22.										
-	pTrg_everyPSSdu_	This Boolean parameter indicates whether “Every x SDU” polling is configured. See <i>Polling</i> , pg. 22.										
-	pTrg_windowBased_	This Boolean parameter indicates whether										

		"window-based" polling is configured. See <i>Polling</i> , pg. 22.
-	pTrg_timerBased_	This Boolean parameter indicates whether "timer-based" polling is configured. See <i>Polling</i> , pg. 22.
-	sTrg_missPdu_	This Boolean parameter indicates whether a status report transmission is triggered by detection of missing PDU(s). See <i>Status Report Transmission</i> , pg. 24.
-	sTrg_tmr_	This Boolean parameter indicates whether a status report transmission is triggered periodically. See <i>Status Report Transmission</i> , pg. 24.
-	use_poll_proh_	This Boolean parameter indicates whether poll prohibiting is configured. See <i>Polling</i> , pg. 22.
-	use_st_proh_	This Boolean parameter indicates whether status report prohibiting is configured. See <i>Status Report Transmission</i> , pg. 24.
-	enable_reset_	This Boolean parameter indicates whether the RLC Reset procedure is enabled. This is not specified in the protocol specification. See <i>RLC Reset procedure</i> , pg. 31.
-	enable_pgbck_	This Boolean parameter indicates whether Status PDU piggybacking is enabled. This is not specified in the protocol specification. See <i>Protocol Data Units (PDUs)</i> , pg. 7.
-	tm_poll_to_	Timeout value for the timer <i>Timer_Poll</i> . See <i>Polling</i> , pg. 22.
-	tm_pl_proh_to_	Timeout value for the timer <i>Timer_Poll_Prohibit</i> . See <i>Polling</i> , pg. 22.
-	tm_pl_per_to_	Timeout value for the timer <i>Timer_Poll_Periodic</i> . See <i>Polling</i> , pg. 22.
-	tm_st_proh_to_	Timeout value for the timer <i>Timer_Status_Prohibit</i> . See <i>Status Report Transmission</i> , pg. 24.
-	tm_st_per_to_	Timeout value for the timer <i>Timer_Status_Periodic</i> . See <i>Status Report Transmission</i> , pg. 24.
-	tm_rst_to_	Timeout value for the timer <i>Timer_RST</i> . See <i>RLC Reset procedure</i> , pg. 31.
-	tm_mrw_to_	Timeout value for the timer <i>Timer_MRW</i> . See <i>SDU discard with explicit signaling procedure</i> , pg. 28.

-	delack_	Timeout value for the timer used when sending delayed Status PDUs. This is not specified in the protocol specification. See <i>Protocol Data Units (PDUs)</i> , pg. 7.
-	rtxto_	Timeout value for the timer Timer_Discard . See <i>Limited Reliability of the RLC/AM</i> , pg. 27.
-	rlcdebug_	This Boolean parameter indicates whether the debug mode is enabled for this simulation. This is not specified in the protocol specification. In this mode, a file in the /tmp directory is created for each node in the simulation each time the simulation is run. The file contains debugging information (triggering of events, values of variables etc.).

WAN topology diagrams

This section contains all the diagrams of section 6.2 using the WAN connection for the wired link.

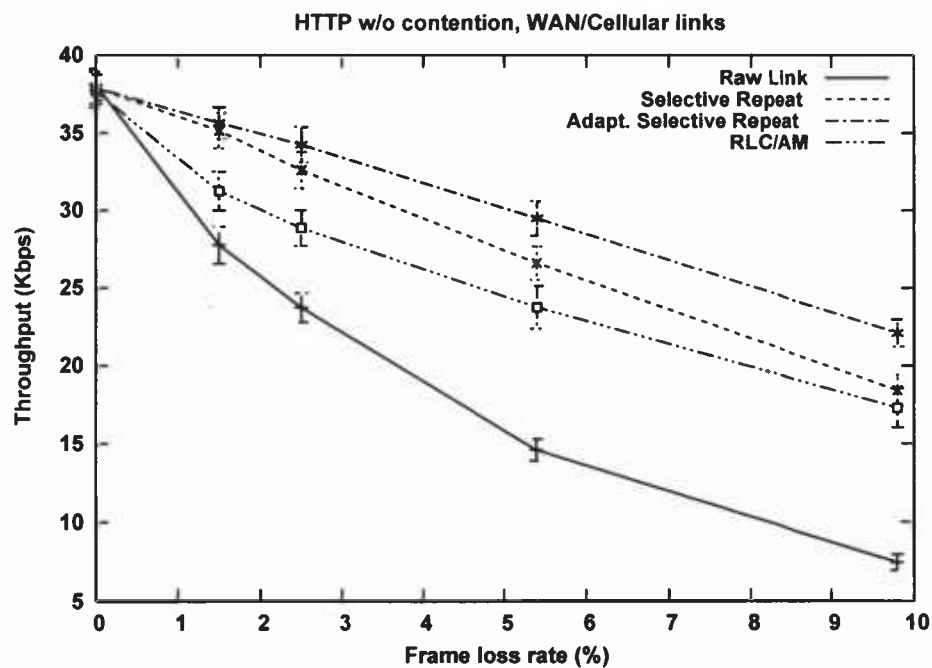


Fig. 33 Web browsing throughput without contention, Uniform model, WAN topology

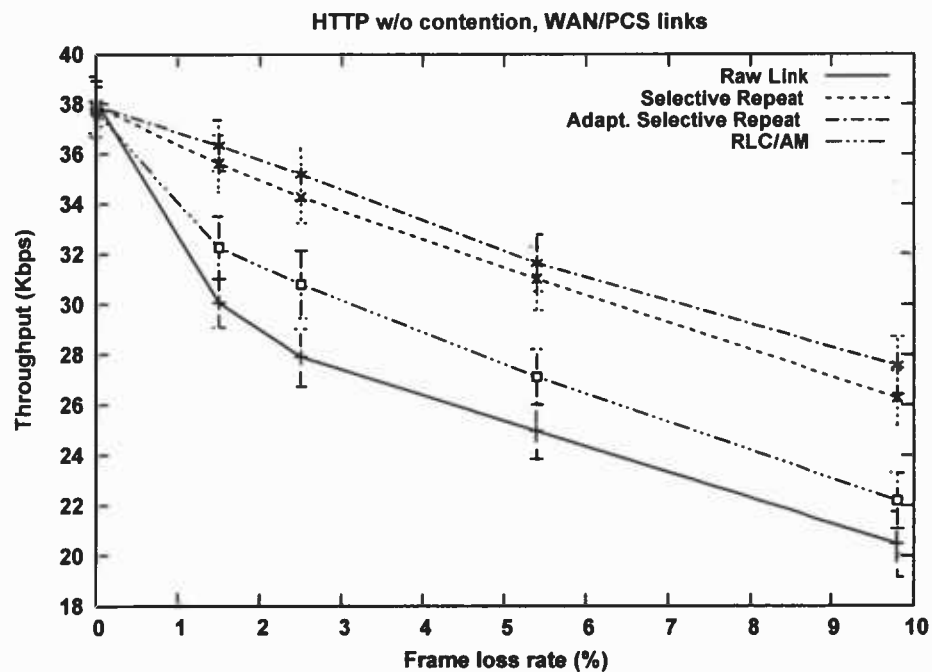


Fig. 34 Web browsing throughput without contention, Two State error model, WAN top.

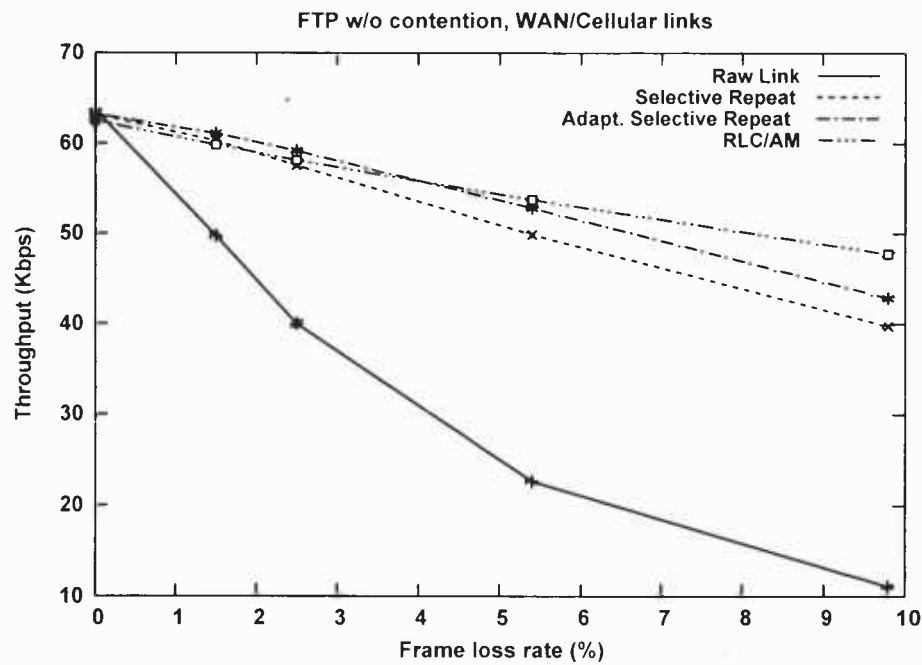


Fig. 35 File transfer throughput without contention, Uniform error model, WAN topology

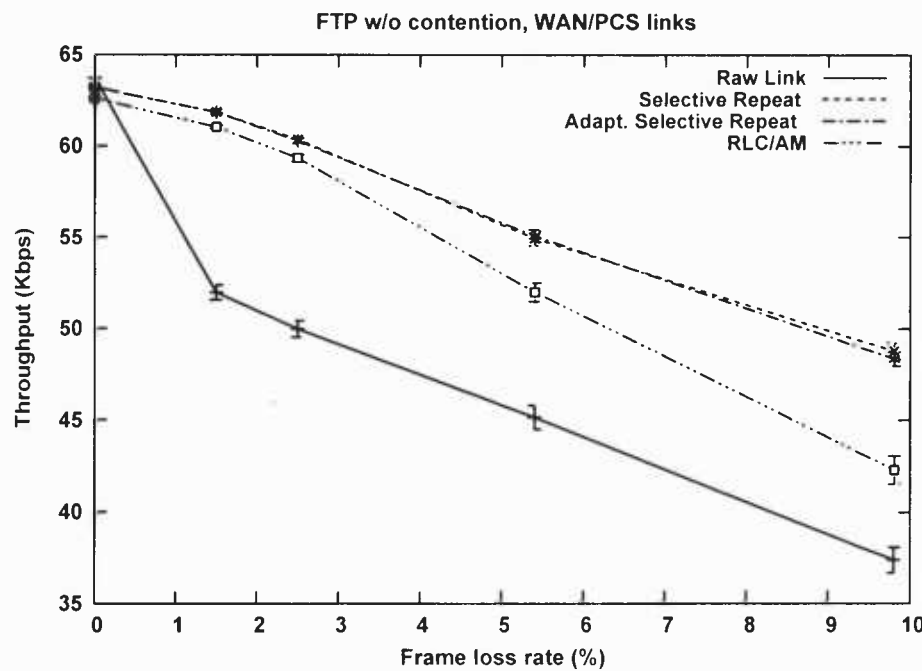


Fig. 36 File transfer throughput without contention, Two State error model, WAN topology

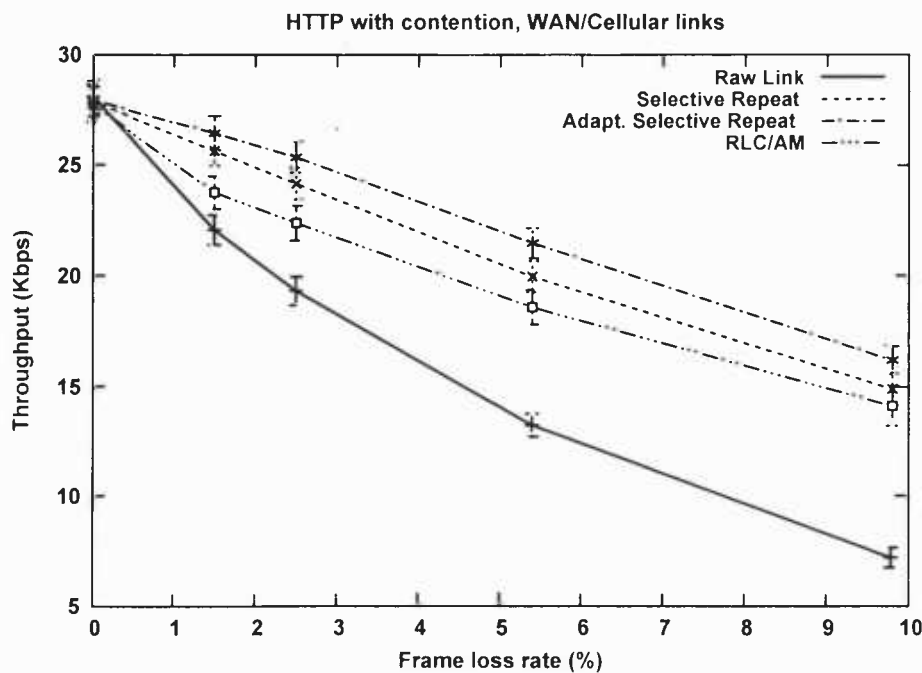


Fig. 37 Web browsing throughput with contention, Uniform error model, WAN topology

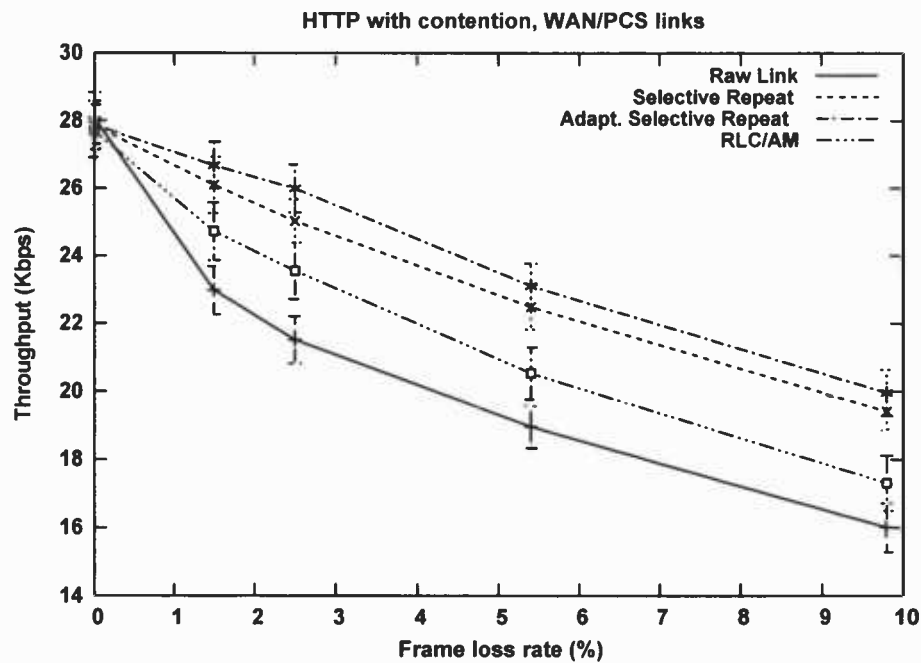


Fig. 38 Web browsing throughput with contention, Two State error model, WAN topology

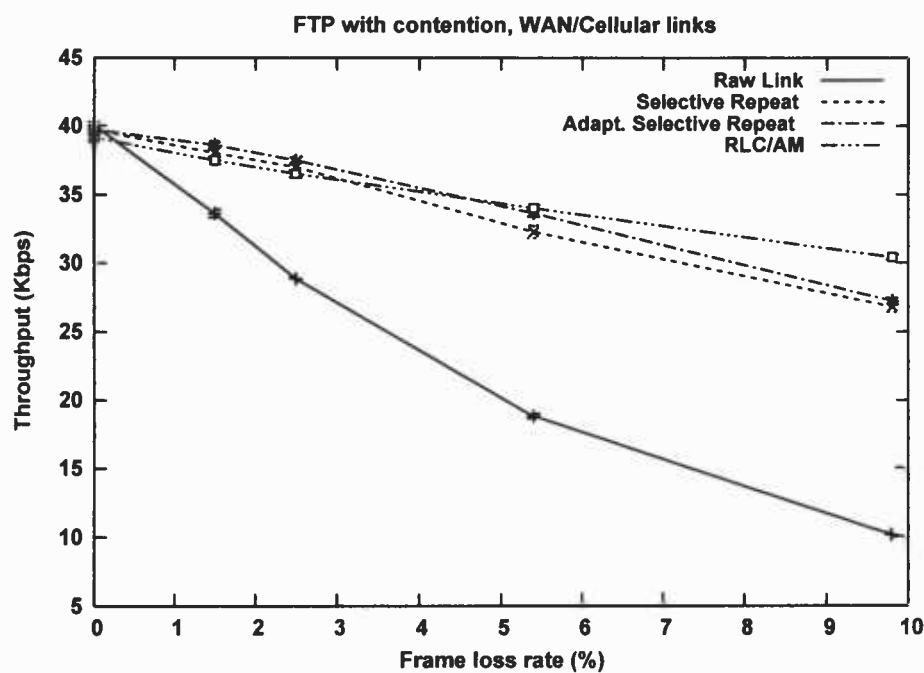


Fig. 39 File transfer throughput with contention, Uniform error model, WAN topology

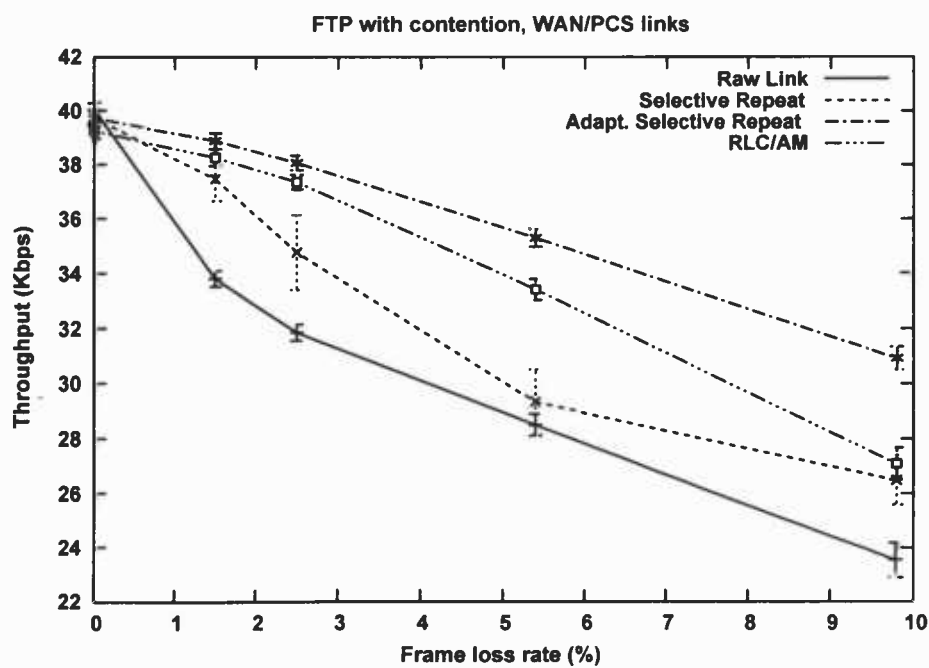


Fig. 40 File transfer throughput with contention, Two State error model, WAN topology

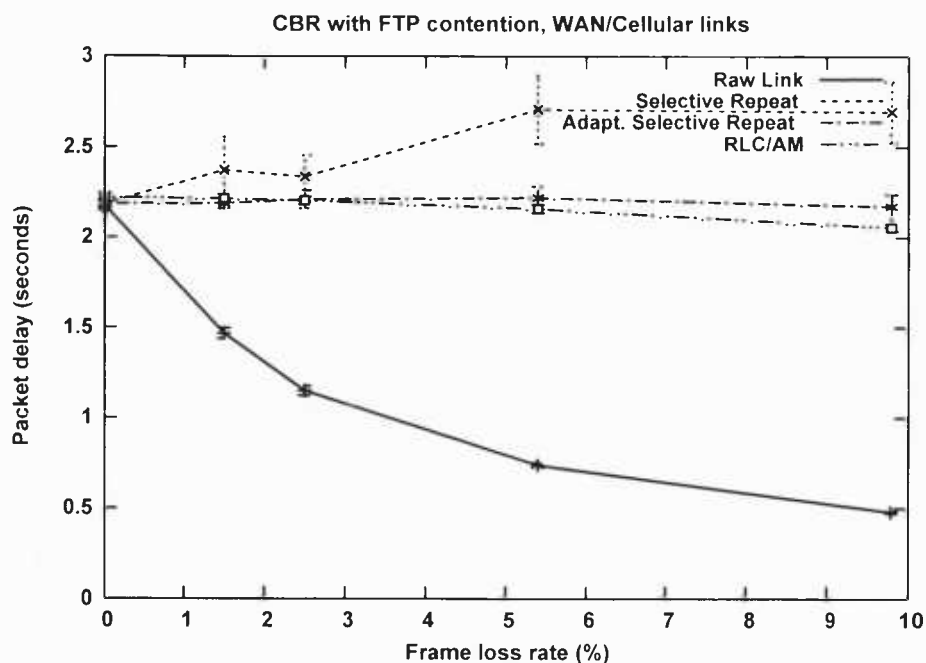


Fig. 41 CBR delay with FTP contention, Uniform error model, WAN topology

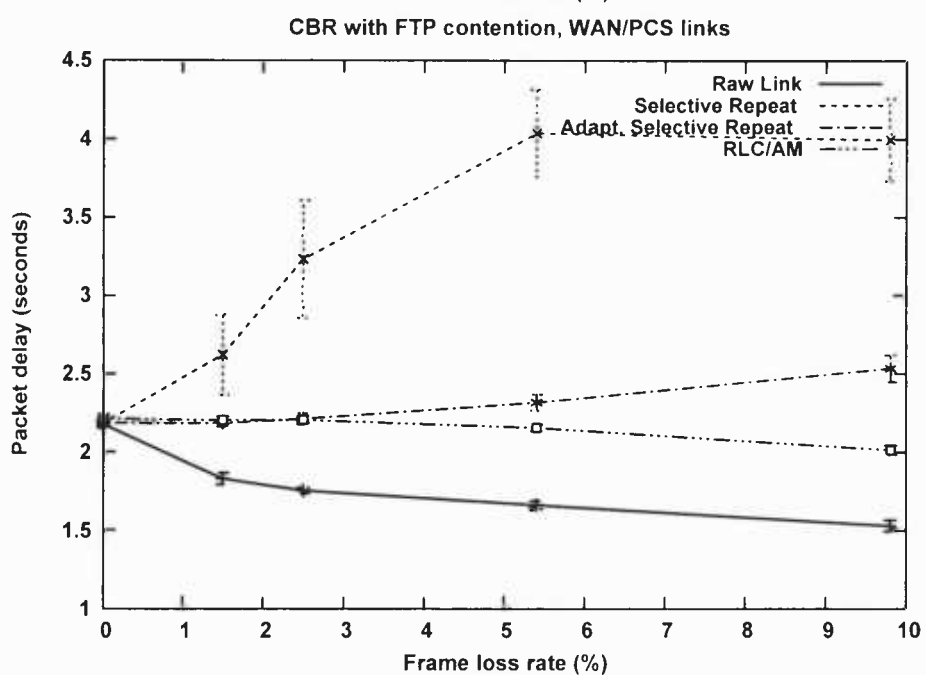


Fig. 42 CBR delay with FTP contention, Two State error model, WAN topology

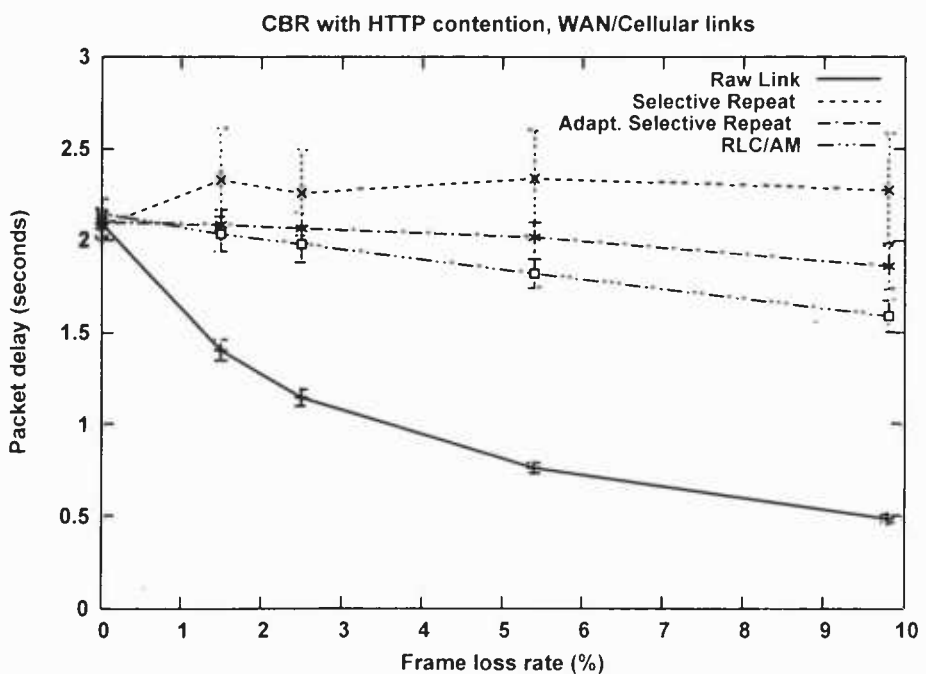


Fig. 43 CBR delay with HTTP contention, Uniform error model, WAN topology

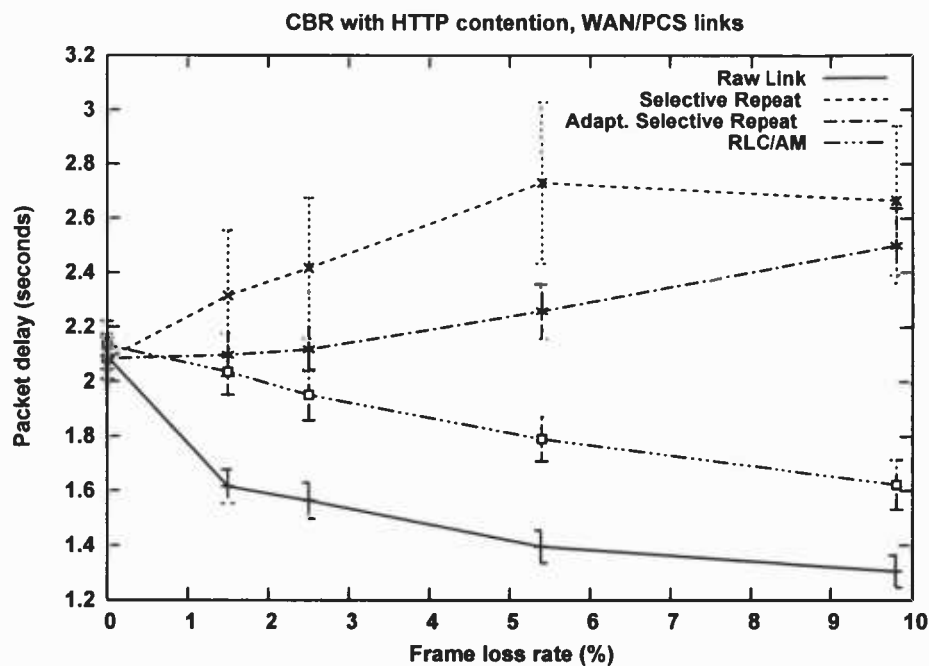


Fig. 44 CBR delay with HTTP contention, Two State error model, WAN topology

SDU Discard policies comparison diagrams

This section includes diagrams from section 6.3 that show the performance of both SDU discard policies tested.

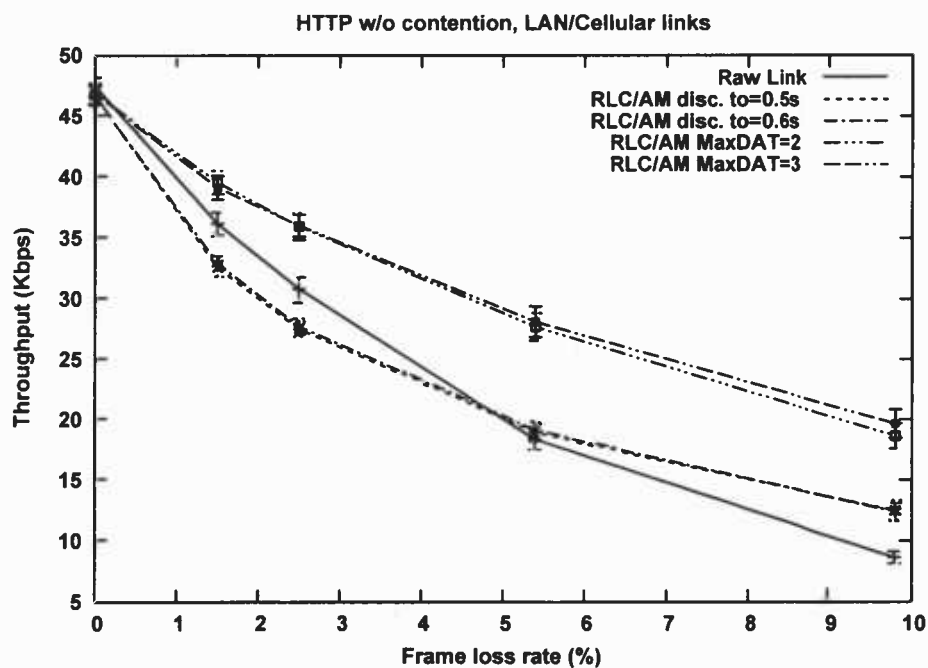


Fig. 45 Web browsing throughput without cont., Uniform error model, LAN topology

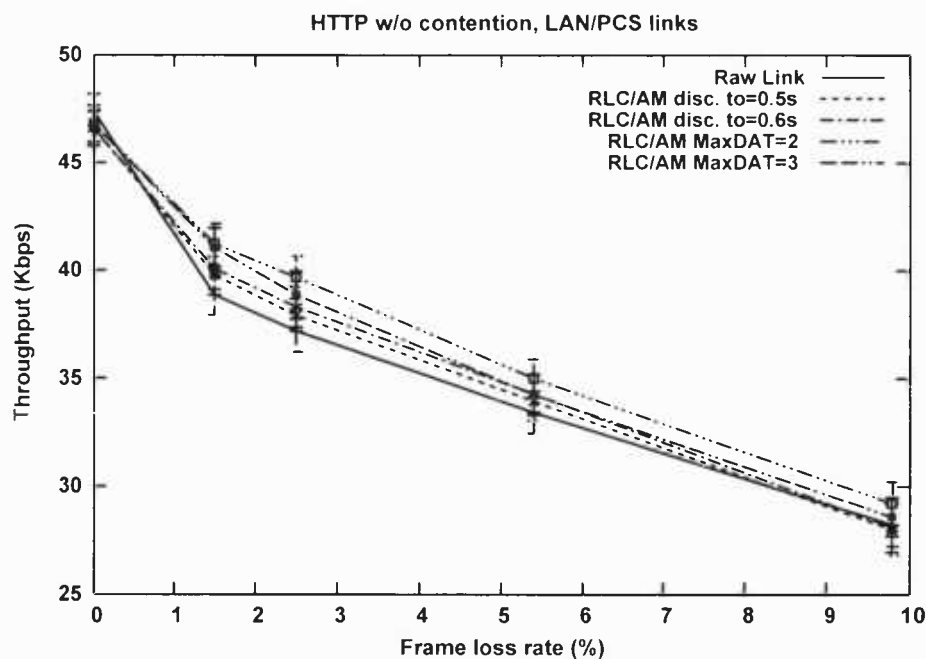


Fig. 46 Web browsing throughput without contention, Two State error model, LAN topology

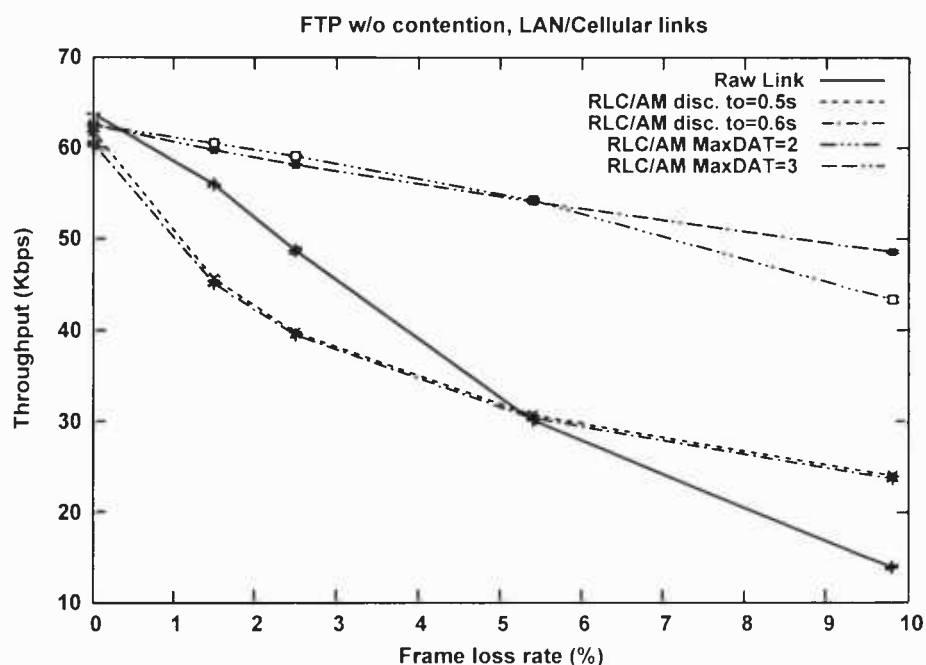


Fig. 47 File transfer throughput without contention, Uniform error model, LAN topology

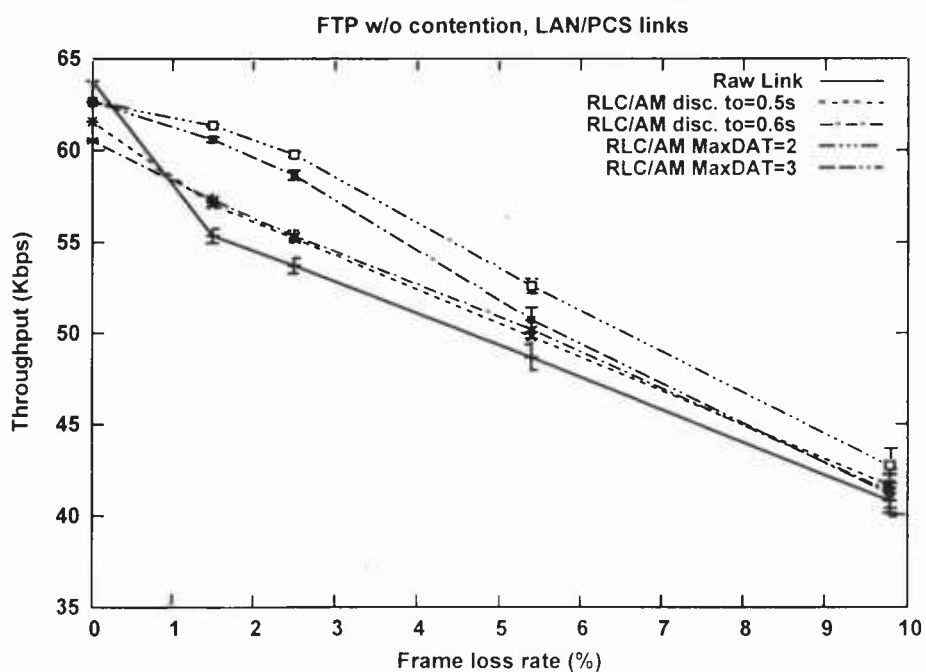


Fig. 48 File transfer throughput without contention, Two State error model, LAN topology

References

- [1] 3rd Generation Partnership Project (3GPP). Radio Link Control (RLC) protocol specification (Release 7). Technical Specification 25.322, V7.0.0. Available at <http://www.3gpp.org> [accessed January 4th, 2007]
- [2] J. J. Alcaraz, F. Cerdan and J. García-Haro. Optimizing TCP and RLC Interaction in the UMTS Radio Access Network. *IEEE Network*, vol 20, no. 2, Mar. 2006, pp. 56-64.
- [3] R. Bestak, P. Godlewski and P. Martins. RLC Buffer Occupancy when Using a TCP Connection over UMTS. *Proc. IEEE PIMRC*, Sep. 2002, vol. 3, pp 1161-1165.
- [4] Yi-Chiun Chen, Xiao Xu, Hua Xu, Eren Gonen, Peijuan Liu. Simulation analysis of RLC for packet data services in UMTS systems. *Proc. IEEE PIMRC 2003*, vol. 1, pp. 926- 930.
- [5] P. Karn. The Qualcomm CDMA digital cellular system. *Proc. Of the USENIX Mobile and Location-Independent Computing Symposium*, 1993, pp. 35-39.
- [6] F. Lefevre and G. Vivier. Optimizing UMTS link layer parameters for a TCP connection. *Proc. IEEE Vehicular Technology Conference*, 2001, vol. 4, pp. 2318-2321.
- [7] M. Rossi, L. Scaranari and M. Zorzi. On the UMTS RLC Parameters Setting and Their Impact on Higher Layers Performance. *Proc. IEEE VTC 2003*, vol. 3, pp. 1827-32.
- [8] UCB/LBNL/VINT. Network Simulator - ns (version 2). Available at <http://www.isi.edu/nsnam/> [accessed January 4th, 2007]
- [9] Hua Xu, Yi-Chiun Chen, Xiao Xu, Eren Gonen, and Peijuan Liu. Performance analysis on the radio link control protocol of UMTS system. In *Proc. IEEE 56th Vehicular Technology Conference*, September 2002, volume 4, pages 2026-2030.
- [10] Xiao Xu, Hua Xu, Yi-Chiun Chen, Eren Gonen, and Peijuan Liu. Simulation analysis of RLC timers in UMTS systems. In *Proc. of the 2002 Winter Simulation Conference*, December 2002, vol. 1, pp. 506-512.
- [11] G. Xylomenos. Multi service link layers for ns-2. Available at <http://mm.aueb.gr/~xgeorge/codes/codephen.htm> [accessed January 4th, 2007]
- [12] G. Xylomenos and G. C. Polyzos. A multi-service link layer architecture for the wireless internet. *International Journal of Communication Systems*, 17(6):553-574, 2004.
- [13] G. Xylomenos. Limitations of Fixed Timers for Wireless Links. *Proc. ISPA 2006*, pp. 159-170.
- [14] G. Xylomenos. Adaptive Timeout Policies for Wireless Links. *Proc. IEEE AINA 2006*, vol 1, pp. 497-502.
- [15] G. Xylomenos, G.C. Polyzos, P. Mahonen, M. Saaranen. TCP performance issues over wireless links. *IEEE Communications Magazine*, vol. 39, no. 4, 2001, pp. 52-58.

- [16] U. Yoon, S. Park and P. Min. Performance analysis of multiple rejects ARQ at RLC (Radio Link Control) for packet data service in W-CDMA system. Proc. IEEE Global Communications Conference, 2000, pp. 48-52.
- [17] Q. Zhang and H.-J. Su. Performance of UMTS Radio Link Control. Proc. IEEE International Conference on Communications, 2002, pp. 3346-3350.



Abbreviations

The following abbreviations are used in this document:

Term	Stands for
ACK	Acknowledgement
AM	Acknowledged Mode
AMD	Acknowledged Mode Data
GSM	Global System for Mobile communications
HSDPA	High Speed Downlink Packet Access
LI	Length Indicator
MAC	Medium Access Control
MRW	Move Receiving Window
NACK	Negative Acknowledgement
PDU	Protocol Data Unit
RLC	Radio Link Control
SDU	Service Data Unit
SN	Sequence Number
SUFI	Super Field
TM	Transparent Mode
UE	User Equipment
UM	Unacknowledged Mode
UMTS	Universal Mobile Telecommunications System
UTRAN	UMTS Terrestrial Radio Access Network

