Emulation Based Performance Investigation of FTP File Downloads over UMTS Dedicated Channels

Oumer M. Teyeb[1], Malek Boussif[1], Troels B. Sørensen[1], Jeroen Wigard[2], and Preben E. Mogensen[2]

[1] Department of Communication Technology, Aalborg University, Fredrik Bajers Vej 7A, 9220 Aalborg East, Denmark
{oumer, mb, tbs}@kom.aau.dk
[2] Nokia Networks, Aalborg R&D, Niels Jernes Vej 10, 9220 Aalborg East, Denmark
{jeroen.wigard, preben.mogensen}@nokia.com

Abstract. The Radio Link Control (RLC) protocol of Universal Mobile Telecommunication System (UMTS) provides link layer reliability that could mitigate the effects of the hostile radio propagation channel on packet data transmission. In this paper, the impact of some of the RLC reliability mechanisms on the performance of File Transport Protocol (FTP) is investigated. The investigations are carried out using a real time emulation platform, which makes the results from this study more realistic than simulation or simplified analytical studies as the overall End-2-End performance is analyzed involving real world protocol implementations.

1 Introduction

Provision of data services is the main driving force behind the current standardization and deployment of 3G (Third Generation) networks. The fact that most of the packet services on the wired Internet today use Transmission Control Protocol (TCP) (TCP averages about 95% of the bytes, 90% of the packets, and 80% of the flows on the Internet [1]) is a clear indication that TCP is also going to be the transport protocol of choice for services running over 3G and beyond networks.

TCP is intended for use as a highly reliable host-to-host protocol in a packet switched computer communication networks[2]. The main features of TCP are its well-designed flow and congestion control mechanisms, which operate on top of its provision of reliability [3]. However, these TCP mechanisms were designed under the assumption that packet losses are only due to network congestion. Though this assumption holds for wired networks, it does not in wireless networks such as UMTS. These networks differ inherently from their wired counterparts, as they have higher error rates, higher latency, and lower yet highly variable bandwidth. As such, TCP performance over wireless networks is quite different from that in

P. Lorenz and P. Dini (Eds.): ICN 2005, LNCS 3420, pp. 388–396, 2005.

wired networks. In UMTS, link layer retransmission mechanisms already exist that can mitigate the influence of higher error rates on TCP performance.

In this paper, we discuss the effects of the settings of some of the UMTS RLC protocol reliability mechanisms on file downloads using FTP through emulation-based studies. Section 2 describes the RLC protocol. A description of the tool used to perform the investigations along with the different mechanisms that are under focus and the performance evaluation metrics is given in section 3. Section 4 discusses the performance evaluation results, and finally section 5 gives conclusions and some pointers to future work.

2 The RLC Protocol

In UMTS, reliable data transmission over the radio channel is provided through the RLC protocol[4]. RLC achieves this link layer reliability through Selective Repeat (SR) Automatic Repeat reQuest (ARQ) mechanism[1]. When the RLC receives a Service Data Unit (SDU) from upper layers, it segments it into RLC Protocol Data Units (PDUs), schedules the PDUs for transmission and then stores them into its (re)transmission buffer. Each PDU is given a unique Sequence Number (SN). Each Transmission Time Interval (TTI), the sender transmits a given number of PDUs depending on the instantaneous allocated bit rate of the air interface.

The receiver keeps the received PDUs in its reception buffer. When all the PDUs that comprise an SDU are received, the receiver assembles the SDU and sends it to upper layers. Depending on the setting of the parameter *in-sequence delivery*, SDUs can be sent to upper layers in- or out-of sequence. PDUs that are received properly are positively acknowledged (ACKed) by the receiver, and those that are not received properly are negatively acknowledged (NACKed). The sender removes the ACKed PDUs from its retransmission buffer, and retransmits the NACKed ones. The receiver sends the ACKs and NACKs using *status* PDUs that contain cumulative ACKs and bitmap fields. A value of n in the cumulative ACK field signifies the correct reception of all PDUs with $SN \leq n$. NACKs and non-cumulative ACKs are sent using bitmaps. For example, if the receiver has received PDUs $\#1, 2, 4, 6$ but not PDU $\#3$ and 5, it will put 2 in the cumulative ACK field and the values $[0, 1, 0, 1]$ in the bitmap.

Status reporting is triggered by the sender, the receiver or both. The sender triggers status reporting by setting the polling bit of some of the PDUs that it sends. The different mechanisms that control poll triggering are:

- **Poll last PDU and Poll last Retransmitted PDU:** If *Poll last (Retransmitted) PDU* is set, the last PDU in the transmission (retransmission) buffer that is sent will be polled. In fig. 1(a), PDU #4 will be polled at TTI

[1] The RLC protocol can operate in three modes, namely Transparent, Unacknowledged and Acknowledged. Only the Acknowledged mode supports retransmission mechanisms, and throughout this paper by RLC it is meant RLC Acknowledged mode.

$\#i$ if Poll last PDU is set and the retransmitted PDU #2 will be polled at TTI $\#j$ if Poll last Retransmitted PDU is set.

- **Poll every poll_PDU:** When this is configured, the polling bit is set on every poll_PDUth (re)transmitted PDU. For example, in fig. 1(b), the poll_PDU value was set to 4. Thus, at TTI $\#i$, when the 4^{th} PDU is sent, and when PDU #7 is sent at TTI $\#j$ the poll bits are set.
- **Poll every poll_SDU:** When this is configured, the polling bit is set on the last PDU of every poll_SDUth SDU. For example, in fig. 1(c), the poll_SDU is set to 1. At TTI $\#i$, when #2 is sent, and at TTI$\#j$, when #9 is sent, the polling bit is set, as they are the last PDUs of the corresponding SDUs.
- **Window based polling:** With *window based polling*, a poll is sent when the % of the occupied transmission window (i.e. the spacing between the first and last non-ACKed PDUs) reaches a certain threshold. In fig. 1(d), this threshold is set to 60%, and the window size is set to 10 PDUs. At TTI $\#j$, % reaches 60, triggering the sending of a poll along with PDU #6.
- **Periodic polling:** If this is configured, a poll is sent regularly at a specified period. In fig. 1(e), the period is set to ten TTIs. When the ten TTIs have elapsed (when we reach TTI $\#j$), a poll is triggered. As there are no PDUs scheduled to be sent, one of the PDUs that have not been acknowledged yet will be resent with the poll.
- **Timer based polling:**When this is configured, a timer is started whenever a poll is sent. When the timer expires, if the sender has not received a NACK or a cumulative ACK for the PDU with the highest SN that was waiting for an ACK when the poll was sent, polling will be triggered. In fig. 1(f), a poll was sent at TTI $\#i$ and when this poll was sent, the last PDU that was expecting an ACK was #4. When the timer expired at TTI $\#j$, neither a NACK nor a cumulative ACK for #4 has been received so a poll will be sent along with the retransmission of #4.

The sender sends a poll in order to receive the status of the PDUs that it has transmitted. The receiver responds to this request by sending a status PDU. In fig. 2(a), for example, the receiver gets a status request along with PDU #4 at TTI $\#i$. During the next TTI, the receiver sends a status report. In this case, as everything up to PDU #4 is received, the status contains a cumulative ACK for #4. There are two other mechanisms in RLC that enable the receiver to send a status report without being explicitly polled by the sender:

- **Periodic status reporting:** With this configured, a status is sent regularly at a specified period. In fig. 2(b), the status periodic is set to two TTIs. At TTI$\#i$ a poll is received so the receiver sends a status report (containing cumulative ACK for #2). At TTI $\#i + 2$, the periodic status fires for the first time. At TTI $\#i + 4$, the periodic status timer fires again and a new status report, containing cumulative ACK #4, is sent.
- **Missing PDU indication:** When this option is set, a status is sent when the receiver gets PDUs out of order, which is a likely indication that some PDUs may have been lost. In fig. 2(c), PDUs #3 and #4 are lost in the air interface. The receiver gets PDU #5 while it was expecting #3 at TTI

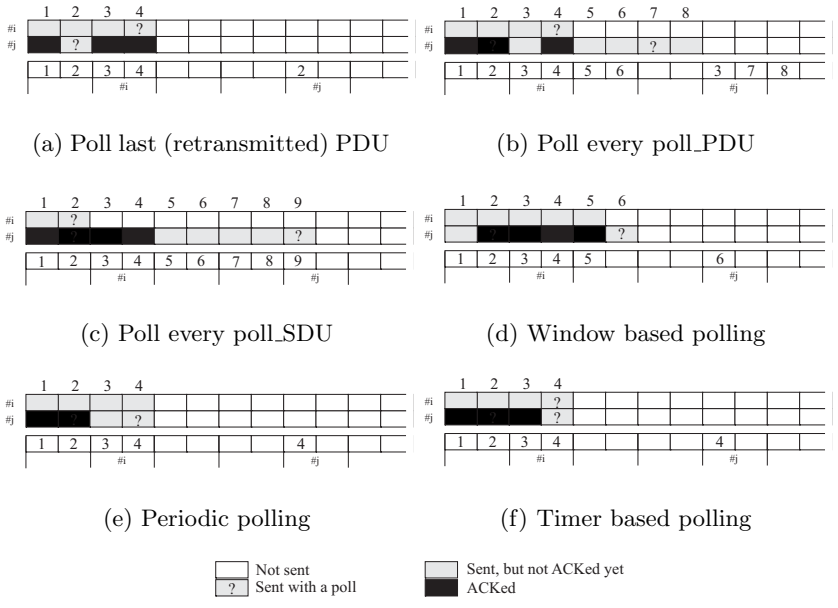


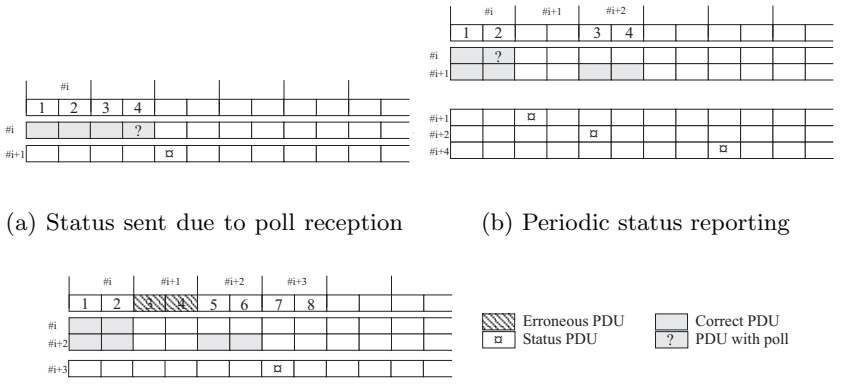
Fig. 1. RLC polling mechanisms. In the sub-figures, the 1^{st} and 2^{nd} rows represent the state of the transmission buffer at different TTIs, and the 3^{rd} row represents the SNs of the PDUs sent at different TTIs

$\#i + 2$. The receiver responds, if missing PDU indication is set, by sending a status PDU, containing cumulative ACK for #2, and a $[0, 0, 1, 1]$ bitmap.

The minimum temporal spacing between consecutive polls and status reports can be controlled by using *Poll Prohibit* and *Status Prohibit* timers, respectively. When these are set, consecutive polls (status reports) will not be sent unless they are separated by a time greater than the configured timers. The number of times that a given PDU could be retransmitted can be specified using the *maxDAT* parameter. If a PDU has been retransmitted *maxDAT-1* number of times and still has not been received properly, the SDU that this PDU is part of will be discarded.

3 Emulation Tool and Investigated Scenarios

The investigations are carried out using RESPECT, a real-time emulation platform for UMTS[5]. RESPECT provides a link-level, real-time emulation of a UMTS network using off-the-shelf Linux operating system. Every Internet Protocol (IP) packet that is arriving to or departing from the machine where the emulator is running is diverted into the emulator, which passes it into an emulated UMTS protocol stack, making it experience the effects of a UMTS network.



(a) Status sent due to poll reception

(b) Periodic status reporting

(c) Missing PDU indication

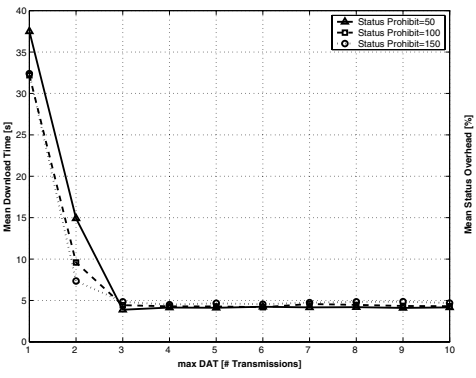
Fig. 2. RLC status reporting mechanisms. In the sub-figures, the 1^{st} row represents the PDUs that are being received during the indicated TTI, the 2^{nd} and 3^{rd} (set) of rows represent the states of the reception and transmission buffers, respectively, at the receiver side at different TTI instances

FTP sessions are started and during these sessions, it is assumed that a dedicated channel (DCH) is already setup. The DCH in the uplink (UL) is considered to be error free with a fixed bandwidth of $32kbps$, while the downlink (DL) has a fixed bandwidth of $384kbps$ with a constant, uncorrelated, frame erasure rate (FER) of 10%. In-sequence delivery is set both in the UL and DL.

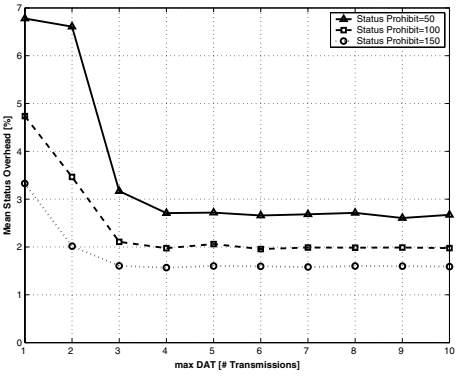
Based on the values given in [6] and [7], one way UMTS processing delay, without considering the transmission time in the air interface, is taken to be $57.5ms$. No limitations are put on RLC buffer and window sizes. *Poll last PDU*, *Poll last Retransmitted PDU* and *Missing PDU indication* are always set, as they help in avoiding many deadlock situations that may arise due to infrequent polling. The emulations were carried out on a machine with the linux 2.4 kernel TCP implementation. The download time, throughput, SDU delay (time taken from the arrival of an SDU till its complete reception and assembly), and status overhead (percentage of status PDUs that are sent as compared with the data PDUs) are used to evaluate the results.

4 Results and Discussion

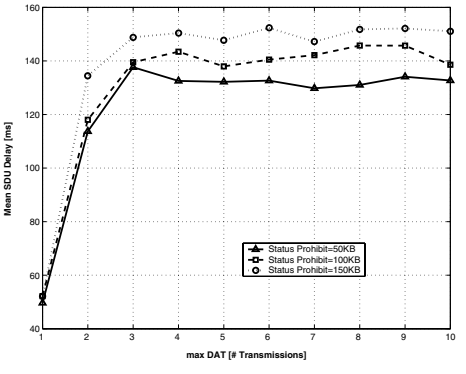
Fig. 3(a) shows the dependency of the download time on maxDAT and status prohibit, for a fixed timer poll value of $100ms$. The emulations were done for a $100KBytes$ file. As can be seen from the figure, for a given status prohibit value, the download time increases as the maxDAT value decreases. For the status prohibit values of $50ms$, $100ms$ and $150ms$, this increase in download



(a) Mean file download time



(b) Status overhead



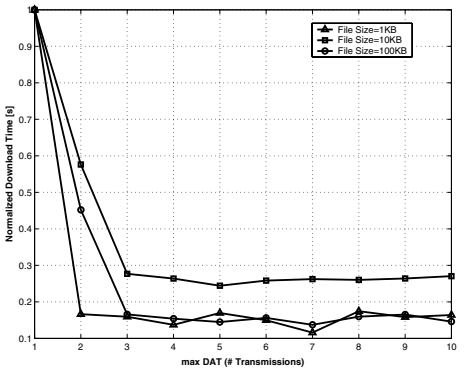
(c) Mean SDU delay

Fig. 3. Results for different maxDAT and Status Prohibit for a $100KBytes$ file download

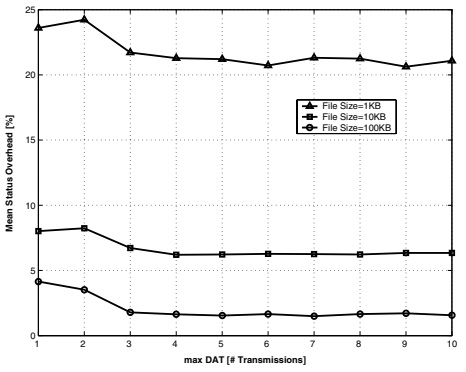
time is 800%, 647% and 589%, respectively, as maxDAT decreases from 10 to 1 [2]. This is an expected result as the main idea behind link layer retransmissions is to decrease the probability of TCP timeouts by retransmitting a subset of the packet, that is the RLC PDUs.

When it is large, maxDAT is the dominating factor and the effect of status prohibit is nullified. This is because even though status prohibit is small and leads to spurious retransmissions as too many status reports arrive due to missing PDU detection, maxDAT is high enough to prevent the untimely discard of the SDU. This conclusion will not hold true if the status prohibit is set to a very

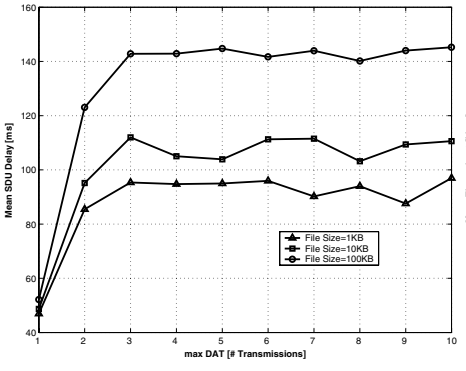
[2] Setting maxDAT to 1 is equivalent to disabling retransmission, and hence nullifying the RLC reliability mechanism



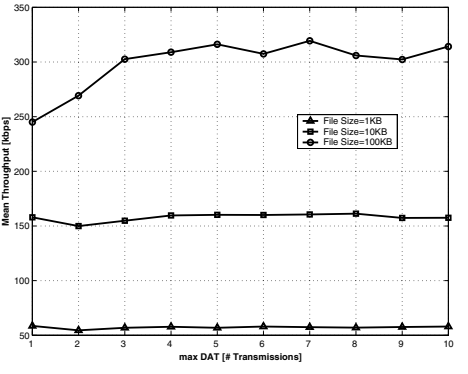
(a) Mean file download time



(b) Status overhead



(c) Mean SDU delay



(d) Mean throughput

Fig. 4. Results for different maxDAT and file sizes

high value, as that would have a considerable impact even if maxDAT is high, by causing status reporting to be delayed for unnecessarily longer periods. For maxDAT values below 3, status prohibit has a noticeable impact on the download time. When the status prohibit value is very low, the frequency of status reporting becomes high, increasing the probability that a PDU is retransmitted. An increase in the retransmission rate will make the retransmission count reach the maxDAT faster, leading to SDU discard. The SDU discard will lead to TCP timeouts, and hence a decrease in the link efficiency, i.e increase in download time.

In fig. 3(b) the effect of the same parameter combinations as the previous case are shown but for the status overhead. This shows that the lower the status prohibit, the higher the status overhead. As maxDAT increases, the status overhead also decreases, in a trend is similar to the download time. This is because for lower maxDAT values, there are a lot of status reports that are wasted completely because the SDUs are discarded anyway even though retransmission requests are coming.

Fig. 3(c) shows the effect of maxDAT and status prohibit on the SDU delay. The SDU delay values are very low for small values of maxDAT. This is because when the maxDAT is very low, there are a lot of SDUs that are discarded and the only SDUs that will account for the mean SDU delay value calculation are the ones that are completed with fewer PDU being retransmitted. For example, for the maxDAT value of 1, the only SDUs that are taken into account for the SDU delay calculation are the ones that are received without any of their PDUs being retransmitted. As the maxDAT value increases, the SDU delay increases, as more and more SDUs are being received properly, mostly with some of their PDUs being retransmitted. After maxDAT reaches 3 the SDU delay value remain almost the same. The effect of the status prohibit for this case is the reverse of what is seen for the download time and status overhead. With low status prohibit value, the retransmission requests come to the sender in a quick succession, increasing the rate of retransmission of PDUs, and hence decreasing the total time required to transmit a given SDU. Though this has a good effect from an SDU delay point of view, it can have a negative effect (and it has for the cases that are investigated here) on the overall file download session, as a most of the bandwidth will be used for retransmissions instead of for first time transmissions, and a lot of battery power will be spent by the mobile terminal through frequent status reporting.

Figures [4(a)-4(c)] also show the dependency of the download time, status overhead and SDU delay on maxDAT, for different file sizes, while the timer poll and the status prohibit are fixed at $300ms$ and $100ms$, respectively. The download times are normalized to the maximum for each file size. From the figures it can be seen that the trend is the same as in the previous cases, i.e. an increase in maxDAT leads to a decrease in the download time, a decrease in the status overhead and an increase in the SDU delay, for a given file size.

For small file sizes, as can be seen in fig. 4(b), the status overhead is the greatest as the chances of cumulatively acknowledging a lot of PDUs at once is very low. However, the SDU delay is the lowest for small file sizes as the probability of an SDU being queued in the RLC transmission buffer before we can start sending it for the very first time is very low.

Fig. 4(d) shows the evolution of the mean throughput as a function of maxDAT for different file sizes. It can be seen that the file size greatly affects the bandwidth utilization, the utilization factor ranging from 15% for the $1KBytes$ case up to 81% for the $100KBytes$ case. The main reason behind this is that for small file sizes, the file download is completed before the TCP connection is able to get out of the initial cycles of TCP slow start.

5 Conclusion

In this paper, the performance of FTP file download over a UMTS dedicated channel, under the assumption of constant bit rate and uncorrelated errors has been investigated. It is found that the main determining factor is the maxDAT value, while the status prohibit value plays a minor role when the maxDAT is

not high enough. No disadvantage of setting maxDAT to a higher value is found for the investigated cases. However, a definite conclusion can not be given unless extensive investigations are carried out considering several issues such as correlated errors and advanced TCP retransmission mechanisms such as Selective Acknowledgments (SACK). Such interactions may lead to redundant simultaneous retransmissions at the RLC and TCP layer, therefore diminishing the advantages of high maxDAT values, or even turning it into a disadvantage. Also, the performance may be different if other type of services such as streaming are considered. In the future, we want to consider the aforementioned factors to arrive at a definite conclusion on the effects of the RLC mechanisms and their parameters settings.

References

1. Greg Miller and Kevin Thompson. The Nature of the Beast: recent Traffic Measurements from an Internet Backbone. http://www.caida.org/outreach/papers/1998/Inet98/Inet98.html.
2. Jon Postel. *RFC 793: Transmission Control Protocol*, September 1981.
3. M. Allman and V. Paxson and W. Stevens. *RFC 2581:TCP Congestion Control*, April 1999.
4. 3GGP TS 25.322 v5.4.0. *RLC Protocol Specification*, March 2003.
5. Malek Boussif, Oumer M. Teyeb, Troels Sørensen, Jeroen Wigard, and Preben M. Mogensen. RESPECT: A Real-time Emulator for Service Performance Evaluation in Cellular neTworks. In *Vehicular Technology Conference, Fall*, 2005. submitted for publication.
6. Harri Holma and Antti Toskala. *WCDMA for UMTS, Radio Access for Third Generation Mobile Communications.* John Wiley & Sons, 2004.
7. 3GGP TS 25.853 v4.0.0. *Delay Budget within the Access Stratum*, March 2001.