The Impact of RLC Delivery Sequence on FTP Performance in UMTS

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

[(1)] Department of Communication Technology
Aalborg University, Denmark
{oumer, mb, tbs}@kom.aau.dk

[(2)] Nokia Networks R&D
Aalborg, Denmark
{Jeroen.Wigard, Preben.Mogensen}@nokia.com

***Abstract*— The Radio Link Control (RLC) protocol of Universal Mobile Telecommunication System (UMTS) provides an option for in- or out-of-sequence delivery of Service Data Units (SDUs) to upper layers. In this paper, the impact of this setting on the performance of File Transport Protocol (FTP) sessions is investigated. The investigations are carried out using a real-time emulation platform, and Transmission Control Protocol (TCP) with and without Selective Acknowledgement/Forward Acknowledgment (SACK/FACK) options. The results show that out-of-sequence delivery has a negative impact at higher bandwidths and TCP FACK exacerbates the problem. This is mainly due to the aggressive retransmission strategy employed by FACK, which considers the gaps in a SACK report caused by reordering as lost segments and retransmits them immediately.**

Key words: RLC, TCP, packet delivery order, UMTS, Network Emulation

1 INTRODUCTION

In UMTS, the RLC layer provides a reliable link layer data transmission, by providing selective request (SR) retransmission mechanisms [1]. The RLC receives SDUs from upper layers (TCP/IP), segments them into RLC Protocol Data Units (PDUs), schedules the PDUs for transmission and then stores them in its (re) transmission buffer. Each Transmission Time Interval (TTI), the sender transmits a given number of PDUs depending on the instantaneous allocated bit rate on the air interface. The receiver positively acknowledges (ACKs) the correctly received PDUs and negatively acknowledges (NACKs) the ones that are not received properly, by sending status messages to the sender. The sender removes the ACKed ones from its retransmission buffer and retransmits the NACKed ones.

There are mechanisms that enable both the sender and the receiver to control the status reporting process, and hence the RLC retransmission. On the sender side, a poll bit can be set in some of the PDUs that are sent to indicate to the receiver that a status report is needed. This polling can be sent either periodically or using some PDU/SDU counters. When the receiver gets these polling requests it complies by sending a status message reflecting the status of its receive buffer. On the other hand, the receiver can control the status reporting by using a periodic timer or instantaneously when out-of-order PDUs are received. In order not to send the polling and status requests at a rate higher than required, there are timers both at the sender and receiver that control the time spacing between two consecutive polling requests and status reports, respectively. The reader is referred to [1] and [2] for a detailed description of the RLC retransmission mechanisms.

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. The *in-sequence delivery* option can be used to specify whether the RLC delivers the assembled SDUs in- or out-of-sequence. If out-of-sequence delivery is enabled, it will increase the possibility of TCP receiving out-of-order packets, in addition to the reordering that might happen on the Internet.

Packet reordering has recently gained a lot of attention in the research community. Measurements performed in [3], [4], and [5] show that packet reordering is not uncommon on the Internet, and the probability of a reordering happening in a TCP session can be as high as 90%, depending on the network load. The effects of packet reordering on TCP performance are studied in [3] and [4], the results showing that reordering results in performance degradation by making TCP resort to too many fast retransmit/recovery invocations. However, a recent study in [6] suggests that packet reordering can turn out to be beneficial from a network point of view, although the conditions for this to happen were quite limited (for example only long lived flows with uniform reordering). In all the quoted studies the focus was on the wired Internet, where the main reasons behind packet reordering are multi-path routing and local parallelism (see [3] for details), and where TCP is the only protocol in the protocol stack that is providing reliability.

The focus of this paper is to investigate the impact of out-of-sequence delivery caused by errors on the air interface, on FTP sessions over a UMTS network; where apart from TCP, the RLC also provides reliability. The impact of using TCP FACK, which is intended to improve the congestion control behavior of TCP SACK, is also investigated. The remainder of this paper is structured as follows. Section 2 gives a brief overview of the SACK and FACK TCP enhancements. Section 3 gives a detailed description of the emulation scenarios

and the different parameters. The results from the emulation studies are given in section 4. Finally, section 5 summarizes the main conclusions from this study and gives some pointers regarding future work.

2 TCP SACK AND FACK

The performance of many TCP versions is very bad when multiple packet losses occur within a window of transmitted data. In TCP Tahoe [7], multiple packet losses might lead to the resending of packets that are already received successfully. On the other hand, in TCP Reno and New-Reno [8], the sender can only retransmit at most one lost packet per round trip time (RTT) (details on the problems of many different TCP versions with multiple packet losses can be found in [9]). TCP SACK was proposed mainly to avoid such problems [10].

With SACK, the receiver can give the sender detailed information regarding the received data by using a bitmap of the packets that has arrived successfully, instead of just sending cumulative ACKs as in the case of the aforementioned TCP versions. This makes it possible for the sender to retransmit only the segments that have actually been lost. The basic underlying congestion control mechanisms are unchanged, and the only difference from the other versions occurs when multiple packets are lost within a window of transmitted data.

The SACK option that is sent by the receiver contains a series of out-of-order blocks that are successfully received. The cumulative ACK is also included in each TCP packet as in the other versions. The sender is thus aware of the holes in the receiver's buffer, and when it is time to retransmit packets due to duplicate ACKs, it will retransmit only the packets that are not SACKed. Simulation results in [9] have shown that the performance of SACK is much better than that of other TCP versions when it comes to coping with multiple packet losses within a window.

TCP FACK tries to improve the congestion control algorithm and is designed to be used with TCP SACK [11]. Instead of just assuming that each duplicate ACK that is received represents one packet that has left the network (this is the assumption in TCP Reno)[1], FACK explicitly calculates the amount of outstanding data in the network. Whenever this outstanding data is less than the current congestion window size, TCP is able to send data, new ones or retransmissions (still obeying the normal TCP sending algorithm that first checks to see if there is space left in the receive window).

When it comes to retransmitting data, TCP FACK tries to avoid the unnecessary delay of waiting for three duplicate ACKs. The unacknowledged holes between SACK blocks are considered as lost by TCP FACK and retransmitted immediately. Simulation results in [11] have shown that FACK is less bursty (due to its accurate estimation of outstanding data) and can recover from episodes of heavy loss better than TCP Reno with SACK extension.

[1] Note that TCP SACK does not modify the congestion control behavior of TCP, except for disabling the retransmission of already SACKed packets. And as such, it can be used either with Reno or FACK.

3 INVESTIGATED SCENARIOS

The investigations are carried out using RESPECT, a Real-time Emulator for Service Performance Evaluation of Cellular neTworks [12]. Emulation removes some of the limitations of pure simulation, as emulation makes it possible to use real world traffic from applications such as FTP (as compared with traffic generators in simulations), on top of a mixture of real world and simulated protocol layers.

RESPECT provides a link-level, real-time emulation of a UMTS network on a standalone computer running Linux. Every packet that is arriving to or departing from the machine where the emulator is running is diverted into the emulator (see Figure 1), which then passes it into an emulated UMTS Radio Access Network (UTRAN) protocol stack, making it experience link delays, errors and link level retransmissions. The RLC protocol is fully implemented in RESPECT as specified in [1]. The overall effect of this setup is to emulate an end-to-end connection between a User Equipment (UE) and an application server where the last hop is the air-interface of a UMTS network.

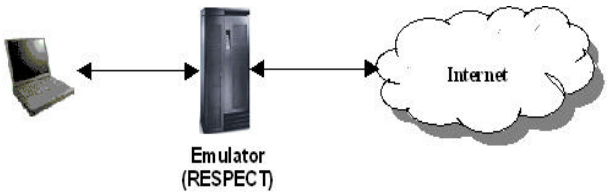

Figure 1. *Emulation Setup*

For the investigations, FTP file download sessions of a 100Kbyte file are used. During these sessions, it is assumed that a dedicated channel (DCH) is already setup, and as such DCH setup and termination times are not considered. The DCH in the uplink (UL) is considered to be error free, while the downlink (DL) experiences a constant uncorrelated frame erasure rate (FER) of 1, 10, or 20%. The bandwidth of the DCH both in the uplink (32kbps) and downlink (32, 128 or 384 kbps) are fixed for a given FTP session. Based on the recommendations in [13], the one-way latency from the UE to the UTRAN (i.e. the total processing time required in all the network entities for a frame of data, excluding the transmission time) is set to 57.5ms. No limitations were set on the RLC buffer sizes, and the

polling/status timers and related parameters were set to the optimal values found in the studies described in [2].

The TCP implementation from the Linux 2.4 kernel distribution is used, whose congestion control strategies are described in [14]. TCP SACK/FACK options are enabled and disabled to see the impact of out-of-sequence delivery on them. The other TCP parameters such as initial window size and maximum window size are set to the default values in the Linux TCP implementation. Table 1 gives the summary of the main emulation parameters.

The results from the investigations are evaluated using the average goodput, number of TCP retransmissions per FTP session and the RTT.

Table 1: Main emulation parameters

Parameter	**Value(s)**
RLC Parameters	
Maximum retransmission retries	10
Timer based polling	300ms
Poll last transmitted or retransmitted PDU	True
Missing PDU indication	True
Status prohibit timer	100ms
Buffer sizes	Unlimited
RLC PDU size	40 bytes
TTI	10ms
PHY Layer	
UL/DL FER	0/ (1, 10, 20%)
UL/DL bit rate	32/(32,128,384kbps)
Other	
#Users	1
TCP version	Linux 2.4 distribution with and without SACK/FACK
File Size	100Kbytes

4 RESULTS

Figure 2 shows the normalized goodput for different bit rates and error rates, where the normalization for each curve is done by dividing the individual goodput values by the maximum goodput for each bit rate. TCP with FACK/SACK options disabled (note that these options are enabled by default in Linux). As can be seen from the figure, for the 32 kbps connection, the in-sequence and out-of-sequence cases are indistinguishable. For the 128kbps, there is a small difference; the in-sequence case performs marginally better. For the 384 kbps case, the difference is more significant, in the order of 5%.

Although this small difference is not that substantial, it shows a clear trend that the effect of out-of-sequence delivery becomes more significant as the bit rate increases, This is because for low bit rates, the possibility of an SDU with higher sequence number being assembled properly before another one with a lower sequence number is very small, even if some of the PDUs of the lower sequence numbered packet are lost, as the time it takes to transmit one full packet over the air interface is very high.

For example, with a 32kbps connection, and a PDU size of 320bits, a 1500 bytes IP packet needs 380 ms (i.e. (1500*8 bits)/(320 bits per TTI)) just to be transmitted (i.e. leave the PHY layer). And during this time, there is a high probability of receiving a retransmission request from the receiver of the lost PDUs from previous SDUs (considering that missing PDU indication is configured, the status prohibit timer is 100ms and that the one way latency is only 57.5 ms). As retransmitted PDUs have a higher priority than PDUs that are being transmitted for the first time, the lost PDUs will be retransmitted and reach the receiver before the out of order SDU has all its PDUs received.

However, in the case of higher bandwidth, there are many outstanding packets (and hence PDUs) and the possibility of SDU reordering becomes higher. For a 384kbps connection, for example, a 1500 byte IP packet will require only 40 ms (i.e. (1500*8 bits)/(3840 bits per TTI)) to be transmitted, which is too short for retransmission requests of lost PDUs from previous SDUs to arrive from the receiver. And as the number of reordered packets increases, so will the number of duplicate acknowledgments received by the sender, putting TCP into unnecessary fast retransmit/recovery states.

The same tests are performed, but this time with the FACK/SACK option enabled. Figure 3 shows the normalized goodput for this case, where the normalization is done as in Figure 2, but with the new maximum goodput values for this case. As can be seen from this figure, the differences between in- and out-of-sequence cases are more pronounced than in Figure 2, even for the lower bit rates.

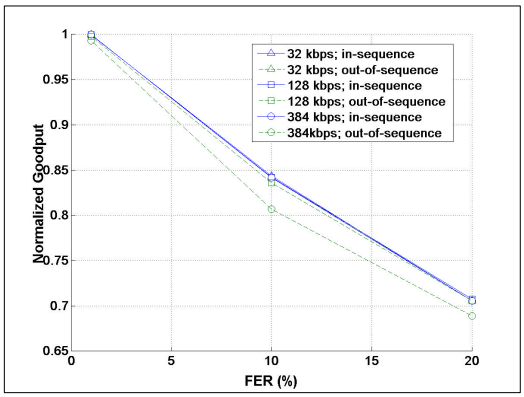

Figure 2. *Normalized goodput with in- and out of sequence delivery (FACK disabled)*

Table 2 compares the number of TCP retransmissions per FTP session for a 384kbps connection, for both cases. When in-sequence delivery is used, it can be seen that the use of FACK decreases both the mean (up to 20%) and the variance (up to 15%) of the number of TCP retransmissions. On the other hand, the reverse is true when considering TCP FACK with out-of-sequence delivery. The mean and standard deviation of the number of retransmissions per session are increased in this case by up to 68% and 18%, respectively.

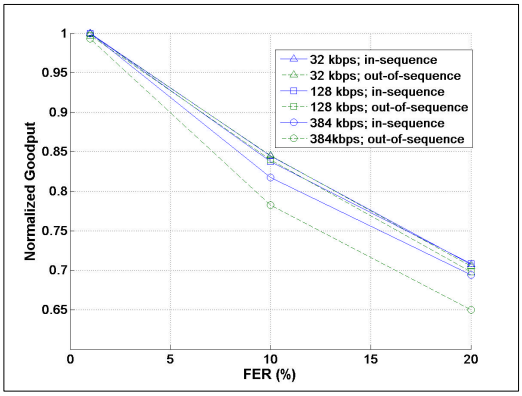

Figure 3. *Normalized goodput with in- and out of sequence delivery (FACK enabled)*

Figure 4 shows the goodput for the 384kbps connection for all the considered cases. For the case of almost no error (1% FER), the FACK/SACK performs slightly better than the no FACK/SACK case, for both in- and out-of-sequence delivery. When the error rate is increased, we see the performance of the FACK/SACK case worsens much faster than the no FACK/SACK if we are using out-of-sequence delivery. For the in-sequence delivery case there is a similar pattern of low performance of the FACK/SACK cases than the no FACK/SACK cases, though the difference between the two seems to be less at 20% FER than at 10%.

Figure 5 shows the RTT for the 384kbps connection for all the considered cases. From the figure it can be seen that for all cases, out-of-sequence delivery leads to lower RTT values than in-sequence delivery. This is because, for the out-of-sequence case, packets are released immediately to upper layers, regardless of their order. For the higher error rates, for both the in- and out-of sequence cases, FACK performs better while for the low error rate FACK leads to increase in RTT.

The increase in the number of retransmissions and hence the goodput performance degradation is due to the aggressive nature of the FACK retransmission strategy. A TCP sender that is using FACK considers the gaps in a TCP SACK report as lost, and retransmits them immediately. This does not necessarily put

Table 2: The Mean and standard deviation of TCP retransmissions within a 100 Kbytes FTP download session, for a 384kbps connection.

	FER	Mean	Std
		Without FACK	
In Sequence	10	0.34	0.6
	20	0.5	0.75
Out of Sequence	10	3.05	1.83
	20	3.83	2.25
		With FACK	
In Sequence	10	0.27	0.51
	20	0.49	0.7
Out of Sequence	10	5.14	2.16
	20	5.4	2.48

TCP in the fast recovery/retransmit state, and hence does not reduce the congestion window size. However, it will lead to simultaneous retransmissions at the link and TCP layers (as the RLC is still trying to recover the lost PDUs of the "lost" packet), wasting valuable air interface bandwidth.

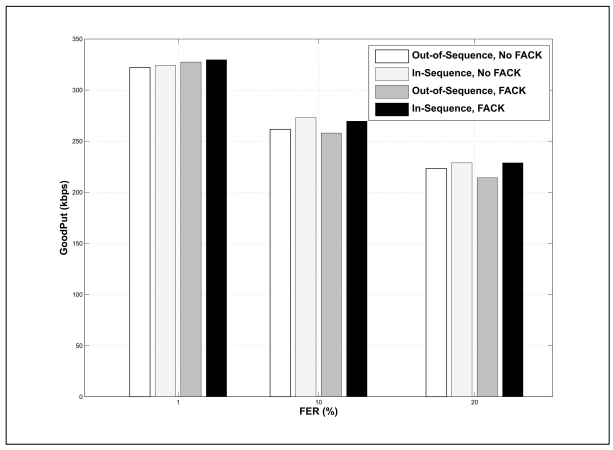

Figure 4. *Summary of goodput for the 384 kbps connection for all the considered cases*

5 CONCLUSION

The use of out-of-sequence delivery at the RLC layer degrades the performance of FTP at higher bit rates. As the air interface bandwidth is expected to grow in future cellular/wireless networks, the usage of out-of-sequence delivery at link layers will have a bigger impact, and thus should be discouraged. When in-sequence delivery is set, TCP FACK performs slightly better than TCP without FACK. When out-of-sequence delivery is used, on the other hand, TCP FACK leads to performance degradation. This is due to TCP FACK's aggressive

retransmission mechanism, which leads to redundant simultaneous retransmissions at both the link and TCP layers. Though there is a little improvement in the RTT due to out-of-sequence delivery, it comes with a penalty of goodput, and hence the download time, the main factor affecting the perceived quality of an FTP session.

While previous studies such as the one carried out in [11] showed a clear advantage of using TCP FACK, the results from our studies showed otherwise. This is mainly due to the differences between the simulation scenarios considered in the studies, i.e. a high capacity wired network with no link layer retransmission mechanisms vs. a lower capacity wireless network with link layer retransmissions. The retransmission mechanism employed by FACK was more costly to us than for the simulations in [11], as there is a link layer mechanism that is also trying to recover the corrupted data packets. A recent study comparing performance of different TCP versions for UMTS [15] also have shown that while SACK is better for FTP services, it may lead to performance degradation for HTTP services, where the source traffic is burstier. This clearly shows that there is still a room for improvement when it comes to the inter-working of link and transport layer retransmission mechanisms.

In the future, we would like to study the effect of out-of-sequence delivery for interactive services such as www browsing. In such services, the RTT is also a critical quality metric, and the disadvantage of the lower goodput maybe compensated by the improvement in the RTT, and that has to be investigated through usability studies. If that is the case, the usage of FACK will also be justified for such interactive services, as it improves the RTT whether we are using in- or out-of-sequence delivery.

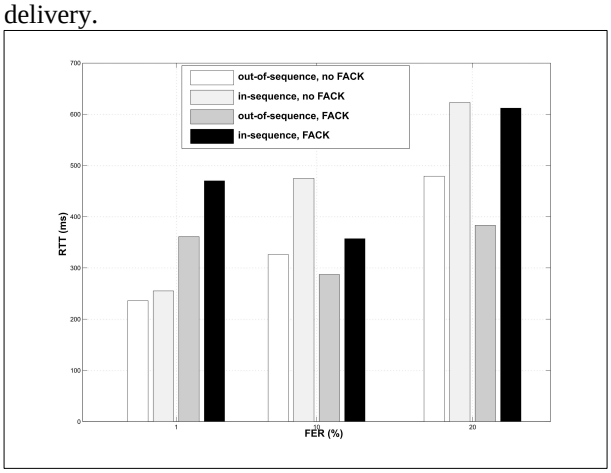

Figure 5. *Summary of RTT for the 384 kbps connection for all the considered cases*

REFERENCES

[1] 3GPP TS 25.322 V6.2.0, "Radio Link Control (RLC) Protocol Specification", December 2004

[2] Teyeb M.O., Boussif M., Sørensen T. B., Wigard J., Mogensen P. E., "Emulation Based Performance Investigation of FTP File Downloads over UMTS Dedicated Channels", *ICN'05*, April 2005

[3] Bennett J. C. R., Patridge C., "Packet reordering is Not Pathological Network Behaviour", *IEEE/ACM Transactions on Networking*, December 1999, vol. 7, No. 6, pp. 789-798

[4] Laor M., Gendel L., "The Effect of Packet Reordering in a Backbone Link on Application Throughput", IEEE Network, September 2002, pp. 28-36

[5] Zhou X., Mieghem P. V., "Reordering of IP packets on the Internet", *Lecture Notes in Computer Science*, , May 2004, Vol. 3015, pp. 237-246

[6] Neglia G., Falletta V., Bianchi G., "Is Packet Reordering Always Harmful?", *12th IEEE symposium on modelling, analysis and simulation*, October 2004, pages 87-94

[7] Jackobson V., "Congestion Avoidance and Control", *ACM SIGCOMM Proceedings*, August 1988

[8] IETF RFC 2001, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms", January 1997.

[9] Fall K., Floyd S., "Simulation-based Comparisons of Tahoe, Reno and SACK TCP", *ACM Communication Review*, July 1996, pp. 5-21

[10] IETF RFC 2018, "TCP Selective Acknowledgment Options", October 1996

[11] Mathis M., Mahdavi J., "Forward Acknowledgement: Refining TCP congestion control", *ACM SIGCOMM Proceedings*, August 1996

[12] Teyeb M.O., Boussif M., Sørensen T. B., Wigard J., Mogensen P. E., "RESPECT: A Real-Time Emulation Platform for Service Performance Evaluation of Cellular neTworks", *VTC Fall 2005*, accepted for publication

[13] 3GPP TS 25.853 , "Delay Budget within the Access Stratum", March 2001

[14] Sarolahti P., Kuznetsov A., "Congestion Control in Linux TCP", *In Proceedings of Usenix 2002/Freenix Track*, June 2002, pp. 49-62

[15] Dubois X., "Performance of Different TCP Versions over UMTS Common/Dedicated Channels", Masters Thesis, University of Namur Computer Science Institute, June 2005.