Implementing immediate forwarding for 4G in a network simulator

Magnus Vevik Austrheim

Thesis submitted for the degree of
Master in Programming and Networks
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Autumn 2018

Implementing immediate forwarding for 4G in a network simulator

Magnus Vevik Austrheim

Implementing immediate forwarding for 4G in a network simulator

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Contents

Chapter 1

Introduction

As the world of networking continues to move towards wireless solutions, more and more challenges emerge as a result of using and patching up older technologies that could not have anticipated either the scale or circumstances of today's realities in their design.

Mobile networks have specific challenges, particularly rapidly varying bit rate and frequent high levels of transmission losses due to interference in highly unpredictable signal conditions. Increasingly more mobile bandwidth is having less and less effect, because delay has become a critical factor limiting performance. Long Term Evolution (LTE) is a standard for high-speed wireless communications for mobile networks, more commonly marketed as 4G LTE, that addresses these challenges by providing features such as lower data transfer latency and a simplified system architecture with better and more robust support for a large number of users.

While there is still room for improvement and with 5G on the horizon for the near future, the system architecture and procedures that mobile network standards like LTE use to tackle its own problems can also impose underlying challenges for the network transport protocols that passes over these mobile network systems. An essential attribute of the Tranmission Control Protocol (TCP) protocol, one of the most used protocols in digital network communications, is that its throughput performance relies on in-order delivery of its data packets to a significant extent. As frequent packet losses and re-transmissions are in the nature of mobile networks, the current method of ensuring in-order delivery in respect to TCP is to use a buffer for storing packets until losses have been repaired, causing delay. With the rapid widespread deployment of TCP RACK (Linux, iOS and Windows), this main assumption of in-order buffering on which mobile data networking was built can be reassessed to allow the opportunity to remove reordering delays and associated buffer memory.

In addition, there is thought that delay can be further reduced by using technologies such as Active Queue Management (AQM) and Excplicit Congestion Notification (ECN) to limit queuing delay in the network elements.

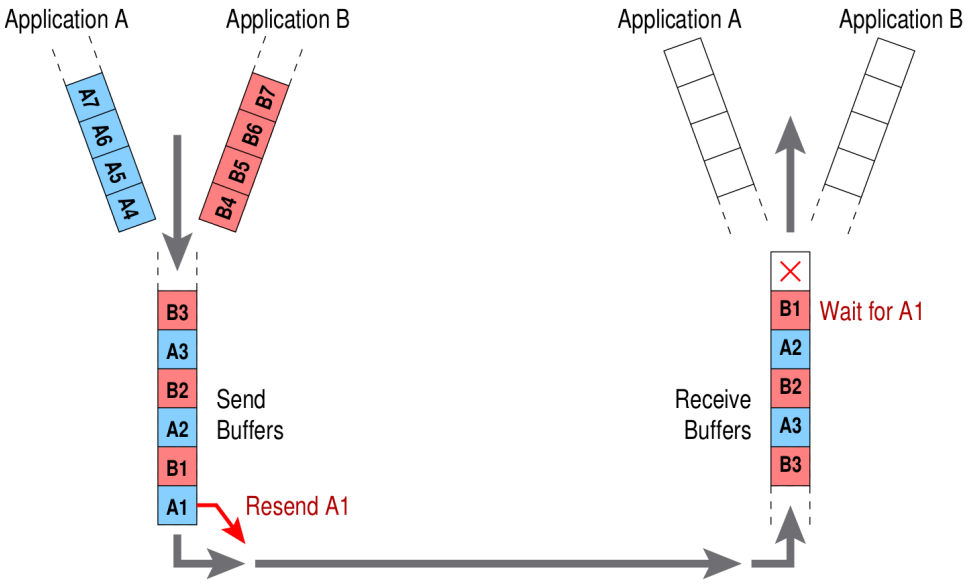

Figure 1.1: Current solution to HoL blocking through buffering

1.1 Hypothesis

TCP RACK is a loss detection algorithm that, instead of counting sequences of duplicate acknowledgements, uses the notion of time for repair of packet losses. While the most common approach is to count three duplicate acknowledgements, RACK uses a time window of one-quarter of the Round Trip Time (RTT), allowing out-of-order sequential reception. If any lost packets do not arrive to fill the entire sequence by this time, the packet is deemed lost. This radically alters a number of assumptions on which lower layer protocols have been designed.

One such assumption is found on the link-layer, where buffering is used to make sure that data segments are sent in sequentially correct order in respect to loss detection algorithms at the transport-layer. This process of reordering data segments causes Head-of-line blocking (HOL), illustrated in figure 1.1. This problem can induce delay, and is implemented with the assumption that the link must send the data in-order to upper-layer to work in conjunction with the assumed sequence based loss detection algorithms.

The hypothesis is that while using TCP RACK, the assumption that the link has to reorder data sequentially in the receive buffer can be rolled back. Instead, the link will send data segments out-of-order to bypass HOL blocking delay, and let the link resend any lost data segments in time to fill any gaps in the TCP receive window within one-quarter on the RTT. The proposed change from the current schematic in figure 1.1 in illustrated in figure 1.2 below.

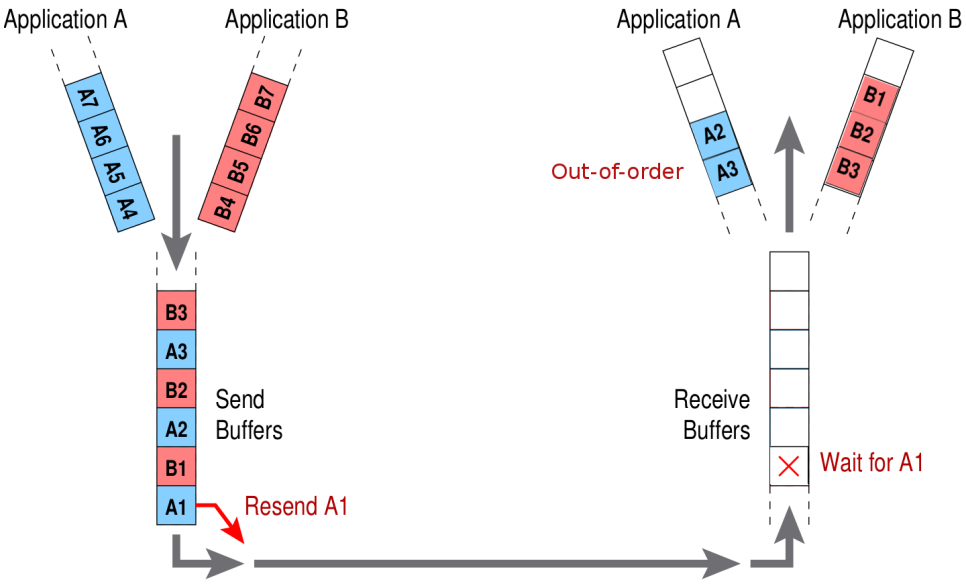

Figure 1.2: Proposal: Bypass HoL blocking with RACK

1.2 Experimental Approach

In order to test the proposed hypothesis, we need to simulate a mobile network with functionality that reaches the technological requirements and specifications. Using the ns-3 network simulator, we can simulate the error-prone wireless environment, and modify the simulators open-source files to implement the desired forwarding scheme.

This thesis aims to solve the following problems:

1. Investigate the potential for forwarding IP packets out-of-order during reorering procedures on the radio link layer, and further measures to favor link-layer retransmissions.
2. Implement the desired scheme in a network simulator.
3. Simulate and compare the performance of a TCP Cubic flow using the default link reordering scheme versus forwarding packets out-of-order.

1.3 Limitations

The focus of this thesis is limited to a single TCP flow transversing the simulated mobile network. While in reality such a flow would compete with other flows in terms of bandwidth and queueing, the performance of such other flows using different protocols in the proposed modified scheme are not considered within the scope of this thesis.

The error models are limited to LTE induced errors, and assumed a perfect link from the end of the mobile network to the remote host. This

enhances the focus on the effect of radio spectrum errors, while limiting the scope to not account for external transmission errors. To further the focus on the proposed scheme, the TCP flow is limited to only transverse one radio link on the network path.

Certain aspects such as data collection, implementations of specifications or Linux kernel version and protocol availability are to different degrees limited to the current version of the ns-3 simulator and its subprojects.

It is also noted that the concept of randomness in regard to error models are limited to computed randomness variables that can never be truly random and therefor not a perfect representation of random errors in the real world.

1.4 Research Method

This thesis follows the *design* paradigm described in [10], which consists of four steps to solve a given problem:

1. State requirements
2. State specifications
3. Design and implement the system
4. Test the system

The system designed and implemented in this thesis is a mobile network simulator with a modified link repair scheme to forward TCP packets out-of-order to bypass reordering delay. The simulated sender and receiver are extended to use a real networking stack in order to enable the required RACK loss detection algorithm.

1.5 Main Contributions

- A ns-3 network simulation program, simulating a mobile user communicating with a remote server over a mobile network using TCP and RACK.
- Modification of the ns-3 implementation of the RLC protocol to forward complete IP packets out-of-order on the radio link.
- Modification to the ns-3 3GPP HTTP Applications to support using the Linux kernel stack.
- Tools for simulator setup, execution and data analysis in the form of various BASH and Python scripts.
- Extension to the current version of the LibOS Linux kernel library to support the full RACK draft specifications in [33].
- Performance evaluation of reordering versus forwarding, including:

1.6 Outline

The thesis consists of the following parts:

- Chapter 2: an overview of the technological background relevant to the thesis.
- Chapter 3: describes the design of the hypothesis.
- Chapter 4: breaks down the implementation of the design.
- Chapter 5: shows the simulation results and evaluation.
- Chapter 6: summerizes the work and draws the final conslusion.
- Chapter 7: lists the future work to build on this thesis.

Chapter 2

Background

Term	Definition
Segment	Unit of data in the *Transport layer*
Packet	Unit of data in the *Network layer*
Frame	Unit of data in the *Link layer*
SDU	Service Data Unit in the *LTE Radio protocol stack*
PDU	Protocol Data Unit in the *LTE Radio protocol stack*

Table 2.1: Cross-layer terminology used.

2.1 IP

The Internet Protocol (IP) is the principal protocol for networking, its purpose is to relay packets through the network from the source to the destination host. The protocol uses IP addresses to identify each host in the network.

Internet Protocol version 4 (IPv4) is currently the most dominant protocol in the internet. Its upgraded successor IPv6, which dealt with the exhaustion of the IPv4 address space, is being deployed incrementally deployed.

2.2 TCP

The Tranmission Control Protocol (TCP) protocol is a transport protocol used for communication between applications, it complements the IP protocol and thus the suite as a whole is often also called TCP/IP. Its purpose is to provide reliability of data transfer through ordering and error checking of packets.

File transferring, e-mail or browsing are examples of applications that uses TCP for its reliability, re-transmitting any lost packets to ensure all data is delivered. For purposes such as video streaming that is not as

dependent as much on reliability, User Datagram Protocol (UDP) is a faster alternative.

In essence, a TCP works by the communicating parties acknowledging successfully transmitted packets. When a sender sends one or more packets, the receiver must respond with an acknowledgement for each packet received. When the sender does not receive an acknowledgement for a packet, it means that there is a hole in the transmission and the packet must be re-transmitted.

2.2.1 Congestion Control and Avoidance

As a statefull connection communicating back and fourth, TCP has limited throughput as opposed to a stateless protocol like UDP which does not regard reliability. In order to utilise the available bandwidth, TCP wants to send as many packets as possible for each iteration, without congesting the network causing these packets to be lost. If there are buffers on the network route that are congested or does not have enough capacity to buffer a large number of TCP packets, the buffer will overflow and the packets will be dropped. In 1986, a congestion control algorithm was added to TCP to enforce conservation of sending packets.[32]

Congestion controls works by keeping a Congestion Window (CWND) variable at the sender which limits the amount of data TCP can send.[7] When initiating a flow, TCP enters *slow-start*, which starts by sending one packet and then exponentially increase the number of packets per Round Trip Time (RTT).[32] At the first sign on congestion, normally by packet loss, the CWND is reduced by half to compensate for the *overshoot* that caused congestion. Now that a reasonable CWND size is determined, TCP enters the state of *congestion avoidance*, conservatively probing for more bandwidth by gradually increasing the CWND until a congestion signal (packet loss) lowers it again. Some of these states are illustrated in figure 2.1.

One form of congestion signal in TCP is by using timers, for example is a packet is not acknowledged before a timeout. Another congestion signal highly relevant to this thesis is duplicate acknowledgements, or dup-acks. An acknowledgement is considered a duplicate when[7]:

- The acknowledgement number is equal to the greatest acknowledgement received on the given connection.
- The advertised window in the incoming acknowledgement equals the advertised window in the last incoming acknowledgement.

in addition to erroneous properties like outstanding/no data or SYN and FIN header bits being off.

Say by way of example a sender sends five packets in sequence, SN(1) through SN(5), but SN(2) is lost in transmission. The receiver receives SN(1) and responds with ACK(1), now expecting SN(2) next. However, since SN(2) was lost the next two packets to be reveived are SN(3), SN(4)

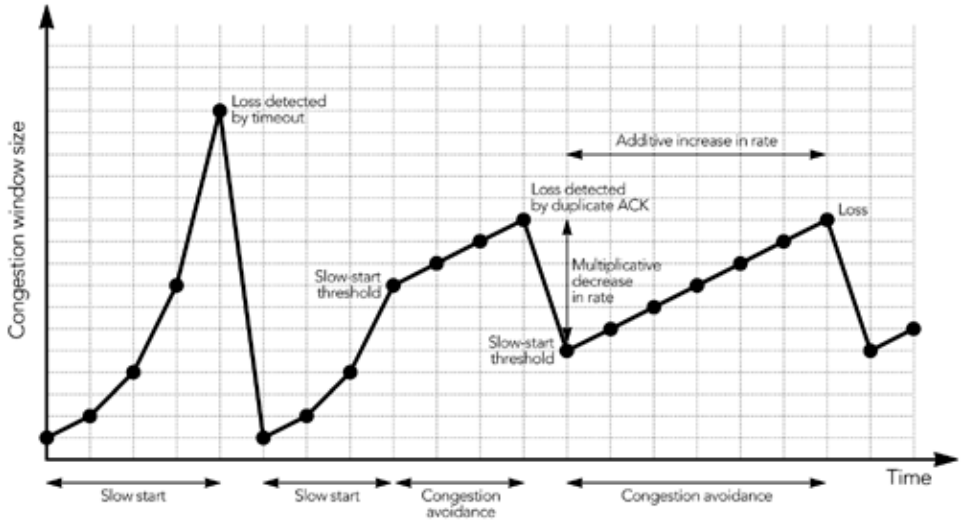

Figure 2.1: Graph showing different TCP states over time, from [24].

and SN(5). Since these sequence numbers are greater than the expected sequence number, the receiver will regard these ad dup-acks and respond with ACK(1) for each of the last three packets to signal that SN(1) was the last packet that was received in-order. This will make the sender count three dup-acks which is the standard threshold and trigger *fast-retranmission*, in which the sender performs a re-transmission of what appears to be the missing segment, without waiting for the re-transmission timer to expire.[7] Different TCP algorithms have their own procedures for handling congestion signals such as this, with the default being a state of *fast-recovery* which waits for acknowledgement of the entire transmission window.

The problem is that TCP cannot determine whether dup-acks are caused by actual congestion or packet disordering.

2.2.2 Selective Acknowledgement (SACK)

Selective Acknowledgement (SACK) is an optional elements for TCP that appends additional information to the otherwise ambiguous dup-ACK. The limited information provided by dup-ACKs means a TCP sender can only learn about a single lost packet per round trip time.[20] As a result, other packets may end up being transmitted multiple times. If both end systems support and enable the SACK option, each acknowledgement will include a range of non-contiguous packets received. Figure 2.2 illustrates an example similar to the previous one, where SN(2) is lost, but with SACK informing which packets have been received out of order. This allows the sender to re-transmit only SN(2) knowing the following sequence numbers have been delivered, where otherwise it could not have know and would have to send all packets starting from SN(2) again.

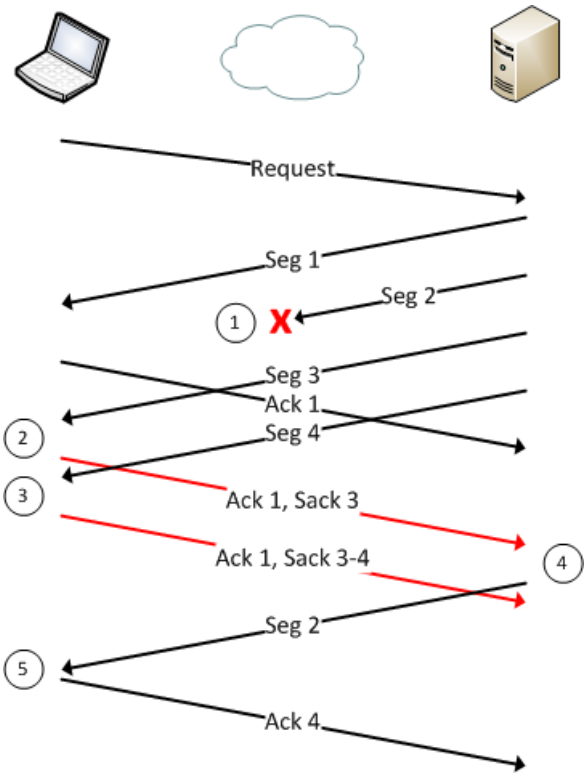

Figure 2.2: Illustration of a TCP connection using SACK.[29]

DSACK is an extension to the SACK specification that enabled TCP to report duplicate segments, which is otherwise not specified in the RFC.[19]

2.2.3 Recent Acknowlegment (RACK)

Recent Acknowlegment (RACK)[8] is a sender-only TCP loss detection algorithm developed by Google, which uses the notion of time instead of the conventional packet or sequence counts to detect losses for modern TCP implementations. It supports per-packet timestamps and the Selective Acknowledgement (SACK) option. It is intended to replace the conventional dup-ACK threshold approach and its variants.

RACK is designed to, among other things, combat three common loss and reordering patterns that has been observed in today's Internet:

1. **Lost re-transmissions.** Traffic policers and burst losses often cause re-transmissions to be lost again, severely increasing TCP latency.

2. **Tail drops.** Structured request-response traffic turns more losses into tail drops. In such cases, TCP is application-limited, so it cannot send new data to probe losses and has to rely on a Retransmission Timeout (RTO).

3. **Reordering.** Link layer protocols or routers' internal load-balancing

can deliver TCP/IP packets out-of-order. The degree of such reordering is usually within the order of the path round trip time.

RACK timestamps all transmitted and re-transmitted packets. It also keeps a *reordering window* and a *minimum RTT* value which are calculated based on the packet relative timestamps. When a packet is received out of order, the packets sent chronologically before that were either lost or reordered. In this case, RACK conducts time-based inferences instead of inferring losses with packet or sequence counting approaches like the standard TCP loss recovery mechanism[7] or Forward Acknowledgment (FACK)[21]. Today's prevalent reordering patterns and lost retransmissions are making packet-counting approached more and more unreliable. Maintaining a duplicate threshold can be difficult as it needs to be constantly adjusted to allow big packets bursts and low enough to detect real packet losses all using sequence space logic. Higher speed links and faster technology in the Internet are also problematic for sequence counting, as rapid volleys of packets can hit the *DupThresh* fast when there are gaps in the sequence due to loss or reordering.

Where a *DupThresh* approach would deem a packet lost at for instance three duplicate acknowledgements, RACK deems a packet lost if some packet sent sufficiently later has been (S)ACKed. 'Sufficiently later' is later by one reordering window, for which the default is set to one quarter of a Round Trip Time, *min_rtt* / 4, for which the minimum RTT is calculated per packet acknowledgement. As of draft *01*, the default is an algorithm that adapts the reordering window to the reordering degree, but the draft used in this thesis uses on quarter of the minimum RTT. For packets that are reordered in the network, there is no need to initiate re-transmission if the packets eventually arrive within the reordering window. The choice for using one quarter of minimum RTT is because Linux TCP uses the same factor in its implementation to delay early re-transmit to reduce spurious loss detection in the prescience of reordering, in addition to that the authors claim through experience that this works reasonably well.[8]

Say by way of example you want to transmit three packets SN(1), SN(2) and SN(3), but reordering on the network path causes SN(3) to be delivered first. If SN(1) and SN(2) arrive within *min_rtt* / 4 to fill the reordering window, they are not deemed lost. If they do however arrive 'sufficiently later' after SN(3), they will be falsely deemed lost and initiate fast re-transmit.

While allowing a small degree of reordering is one of the key innovations of RACK. It also has an advantage in being able to detect losses of retransmissions that the conventional dup-ACK approaches do not when there is an insufficient number of dup-ACKs to trigger fast retransmission. Take SN(1), SN(2) and SN(3) again, this time the first two are actually lost and SN(3) arrives. The receiver sends back a SACK notifying the sender that SN(1) and SN(2) are missing, RACK then eventually deems these lost and re-transmits them as R(1) R(2). Now re-

transmission R(1) is lost again but R(2) arrives and is SACKed, RACK yet again deems R(1) lost and re-transmits it again. Such lost retransmissions are common with low rate limits.

Tail Loss Probe

In the world of congestion control, the worst case scenario is an RTO which induces seconds of delay. While this can be avoided using the re-transmission schemes previously presented, they rely on acknowledgements of subsequent packets to investigate packet losses. This introduces an important edge case, tail loss. When the tail (last few packets) of a transmission is lost, there is not enough subsequent acknowledgements to indicate a loss, which will lead to an RTO timeout. Tail Loss Probe (TLP)[12] is a sender-only algorithm that allows the transport to recover tail losses through fast recovery as opposed to lengthy re-transmission timeouts. TLP works by repeating the last unacknowledged data segment (probe) with an aggressive timer in order to force some feedback from the receiver. If a tail segment was lost, the probe will feed back an acknowledgement to the sender to signal that the tail segment was lost and allowing fast recovery instead of an RTO timeout. An example is illustrated in figure 2.3.

TLP is a supplemental algorithm but works naturally well with RACK to further reduce RTO recoveries. TLP gives RACK additional (S)ACKs to detect losses and in turn RACK relaxes TLP's requirement for using FACK and re-transmitting the highest sequence packet.[8] Given the dramatic delay from an RTO, good use of TLP is important. According to the TLP specification[12]: measurements on Google Web servers show that approximately 70% of retransmissions for Web transfers are sent after the RTO timer expires, while only 30% are handled by fast recovery.

2.3 Long Term Evolution (LTE)

Long Term Evolution (LTE) is a standard for mobile broadband networks for mobile devices and terminals, developed by the Third Generation Partnership Project (3GPP). LTE evolved from the previous mobile network architecture 3rd Generation (3G). The terms 4G and LTE are often interchanged and can be a cause for confusion. The International Telecommunication Union (ITU) intended for the term 4th Generation (4G) to be used for system that meet certain requirements like peak download speeds achieved in different environments etc, using their own *IMT-Advanced* to model the target requirements. LTE did not meet these requirements initially, and the engineering community came to describe it as 3.9G.[11] Needless to say, this did not stop the markets from labelling LTE as a 4G technology, and eventually the ITU gave in, allowing LTE to be described as a true 4th Generation (4G) technology.

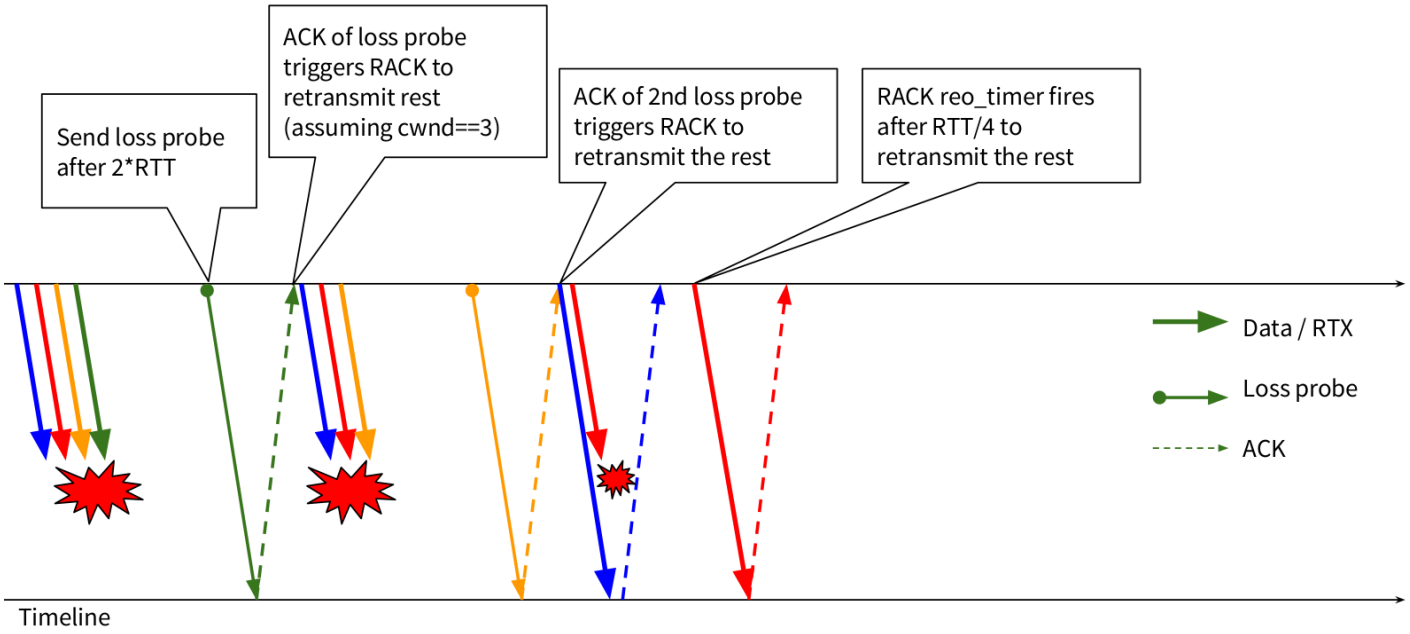

Figure 2.3: RACK + TLP timeline from [9]

2.3.1 The need for LTE

There were various motivations for evolving the revolutionary and dominant 3G system, such as latency issues and giving designers a fresh start from complex 3G specifications as a result of keeping backward compatibility support for earlier devices.[11]

The leading motivation however that pushed the need for LTE was the growth of mobile data. For many years, voice calls dominated the traffic in mobile telecommunication networks. The growth of mobile data was initially slow, but in the years leading up to 2010 its use started to increase dramatically.[11] 2.4 shows how the increase in mobile data usage went to overtake the more stable voice usage.

In the older systems like Global System for Mobile Communications (GSM), voice calls were highly prioritises in the design ans data services were only possible over the circuit switched connection, yielding very low data rates. Later the Universal Mobile Telecommunication System (UMTS) went to improve the data rates by emulating a circuit switched connection for real time services and a packet switched connection for datacom services in the access network, but data packets still relied upon the circuit switched core in the form of allocating IP addresses from establishment and release of data services.[22]

The LTE specifications by the 3GPP describes an evolved design of the older GSM and UMTS systems into a system optimised for packet switching with lower complexity and higher data rates whilst ensure the continuity of competitiveness of the 3G system for the future.[22]

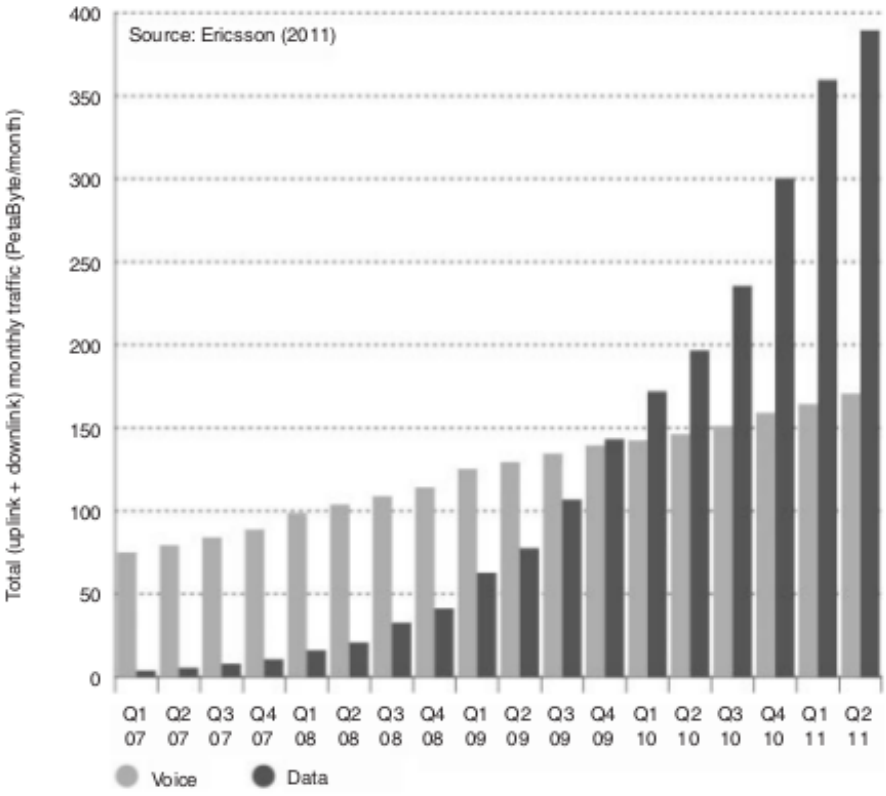

Figure 2.4: Worldwide mobile network traffic and data 2007-2011.[13]

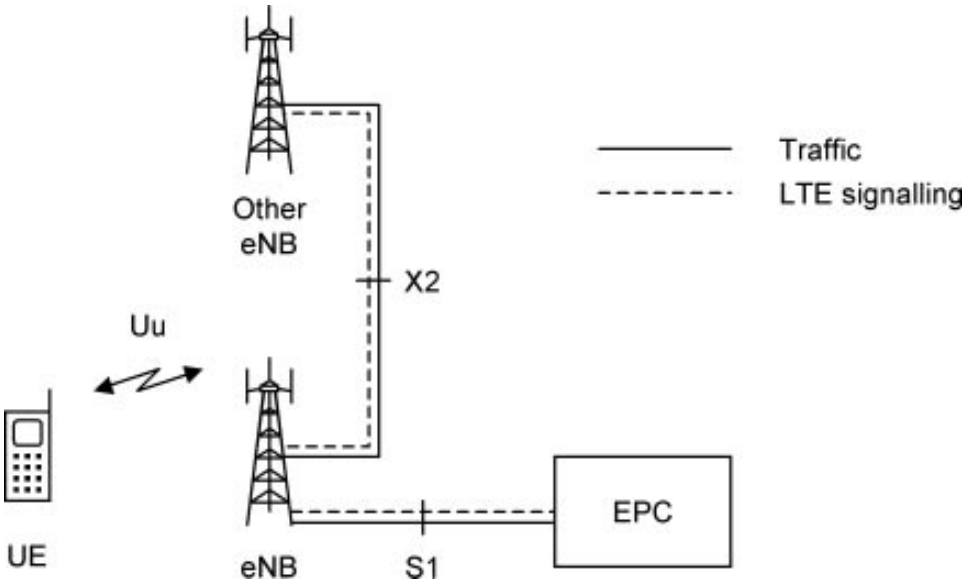

Figure 2.5: Architecture of the evolved UMTS terrestrial radio access network. [11]

2.3.2 LTE Architecture

There are three main components of the LTE architecture, the User Equipment (UE), the uetran and the epc, each containing its own architecture within to fulfil their purpose.

UE

The UE depicts the device that is communicating through the LTE mobile network. It is typically illustrated as a mobile phone, which easily conveys the aspect of mobility and hardware limitations of a UE. In order to communicate with the mobile network, the UE uses the radio link, or wireless transmissions, that are in range of a radio base station that serves as a component of the next LTE component, the E-UTRAN.

E-UTRAN

The Evolved Universal Terrestrial Access Network (E-UTRAN) handles radio communications between the UE and the EPC and just has one component, the Evolved NodeB (eNB), which is the radio base station. Multiple EPCs are used to extend coverage, and the UE only communicates with one eNB at the time, then being the *serving eNB*. If the E-UTRAN contains more than one eNB base station, the base stations can be interconnected through the optional *X2* interface. When a UE moves out of range of one base station into the range of another, a process called *handover*, the base stations can organise via the *X2* interface to move the user session between the two base stations. The data packets are sent from the eNBs to the EPC via the *S1 interface*.

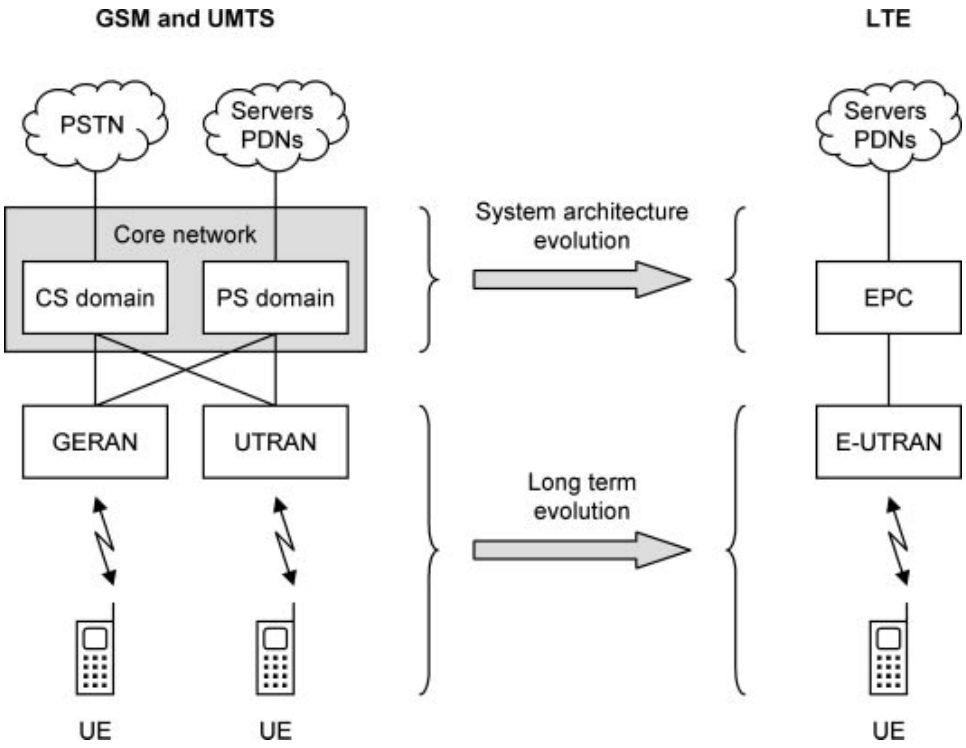

Figure 2.6: Evolution of the system architecture from GSM and UMTS to LTE. [11]

EPC

The Evolved Packet Core (EPC) is the last main component of acrshort-lte, where data packets coming over the *S1* interface from the E-UTRAN is routed to the end of the LTE network and towards the destination end system. The EPC broke free from the circuit switching ways of its predecessors, and is purely based on the IP transport protocol for both real-time and data services. A big factor that allowed the circuit switching core to be replaced was Voice over IP (VoIP) which proved a reliable alternative for real-time voice communications over a packet switched network.

Within the EPC there are again three main components, the Mobility Management Entity (MME), the Serving Gateway (S-GW) and the Packet Data Network (PDN) gateway (P-GW). The *S1* link from the E-UTRAN is split using two logical links, the *S1-U* for traffic going to the gateway and *S1-MME* for signalling messages. The S-GW gateway acts as a router, forwarding traffic fro the base stations to the P-GW via the *S5/S8* interface. The P-GW acts as the end point of the LTE network, connecting with the internet using its *SGi* interface. The MME handles high-level operations such as security issues or unrelated data streams. The MME only uses signalling to manage its network elements.

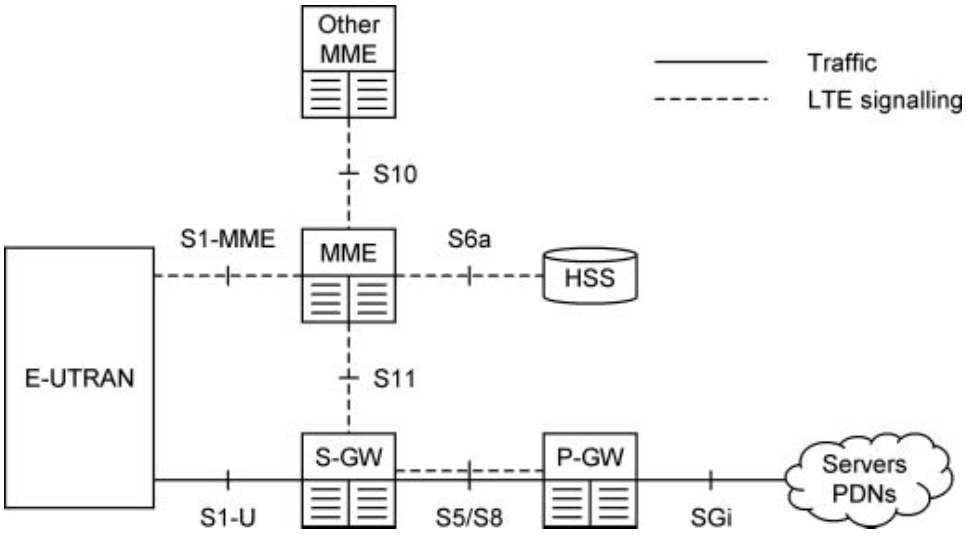

Figure 2.7: Main components of the EPC.[11]

2.3.3 LTE Air Interface

The OSI model standardises communication functions into layers, where (logically) data flows down through the layers, and each layer is responsible for communicating with its corresponding layer on the receiving side. For example application that wants to send a data to a destination host, will send the data to the *Transport layer* where transport protocols such as TCP or UDP are used to determine in what fashion the two parts should communicate. Down towards the lower layers the *Network layer* handles where the data should be sent or routed through the IP protocol, and the *Link layer* and *Physical layer* undertake the physical addressing and actual bit transmissions of the data.

In LTE Air Interface the physical and data link layers, also referred to as layers 1 and 2 respectively, are different from those of their wired counterparts. Layer 1 (PHY) divides data bulks up in smaller pieces that is more manageable and safer to transmit over a wireless radio link, and layer 2 controls these low-level operations and offers functionality to resolve transmission errors before data is forwarded to the *Network layer*. Also, in the higher layers before entering layer 2 of the air interface, there is a separation between the user plane and the control plane. The user plane is where the actually data flows as in the OSI model described above, whereas the control plane handles control signals used to manage the users session with its serving base station in the LTE network. Altogether the layers make up the *Air Interface Protocol Stack*, illustrated in 2.9, while layers 1 and 2 alone is referred to as the *LTE Radio Protocol Stack*.

LTE Channels

Data going through the LTE air interface may be grouped into different *channels*, namely the *Physical channels*, *Logical channels* or the *Transport chan-*

Physical channels	
Physical Uplink Shared Channel (PUSCH)	Uplink data transfer.
Physical Downlink Shared Channel (PDSCH)	Downlink data transfer.
Physical Uplink Control Channel (PUCCH)	Uplink control signals.
Physical Downlink Control Channel (PDCCH)	Downlink control signals.
Transport channels	
Uplink Shared Channel (UL-SCH)	Uplink data transfer.
Downlink Shared Channel (DL-SCH)	Downlink data transfer.
Logical channels	
Dedicated Traffic Channel (DTCH)	Transmission of user data.
Paging Control Channel (PCCH)	Paging information.
Broadcast Control Channel (BCCH)	Broadcats system information to users.
Dedcated Control Channel (DCCH)	User specific information.
Common Control Channel (CCCH)	Setting up new connections.

Figure 2.8: Main LTE channels.

nels. These channels are required to segregate different sorts of data in an organised manner. The various channels types of each of these groups are again sorted into categories of down link and up link, as well as broadcasting, multi-casting, paging etc.

The physical channels are the main transmission channels that handles user data and control signals, while the transport and logical channels are used within the layer 2 protocols. For example when the RLC protocol has data to send, it passes the data down to the MAC layer on a logical channel. The MAC then sends this data as a transport block on a transport channel to physical layer, which in turn encodes this data to signals and transmits this data on a physical channel. The location of these channel groups are pinpointed in figure 2.9.

Table 2.8 lists some of the most commonly referred to channels:

Radio Resource Control (RRC)

The RRC protocol is used by the UE and eNB as part of the control plane in the air interface. The RRC manages different states of a connecting such as being idle, connected or during handover, and configures the lower layers accordingly using control signals. Functions include mobility, configuration of point to point radio bearers and broadcasting system information to higher control plane protocols that handle functions such as authentication. While the RRC is mostly impartial regarding the data plane, it's Quality of Service (QoS) functions does affect resource scheduling on lower layers.

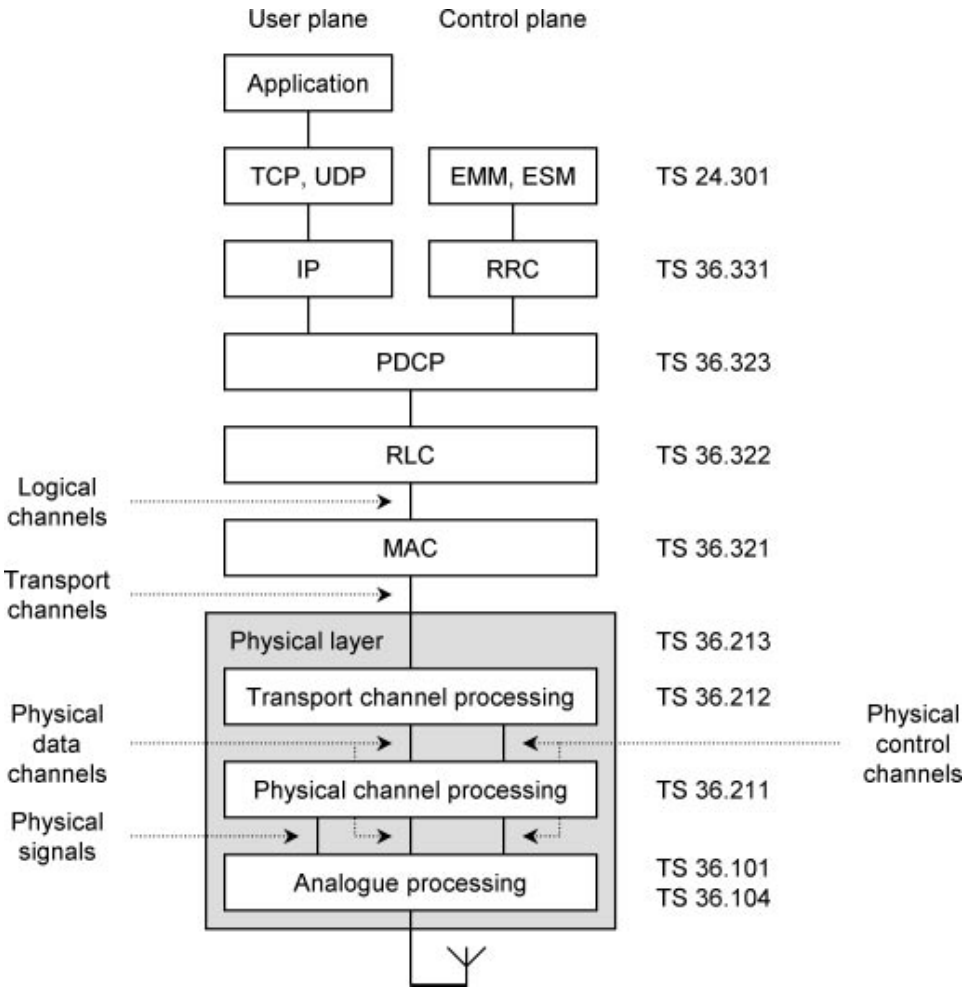

Figure 2.9: Architecture of the air interface protocol stack.[11]

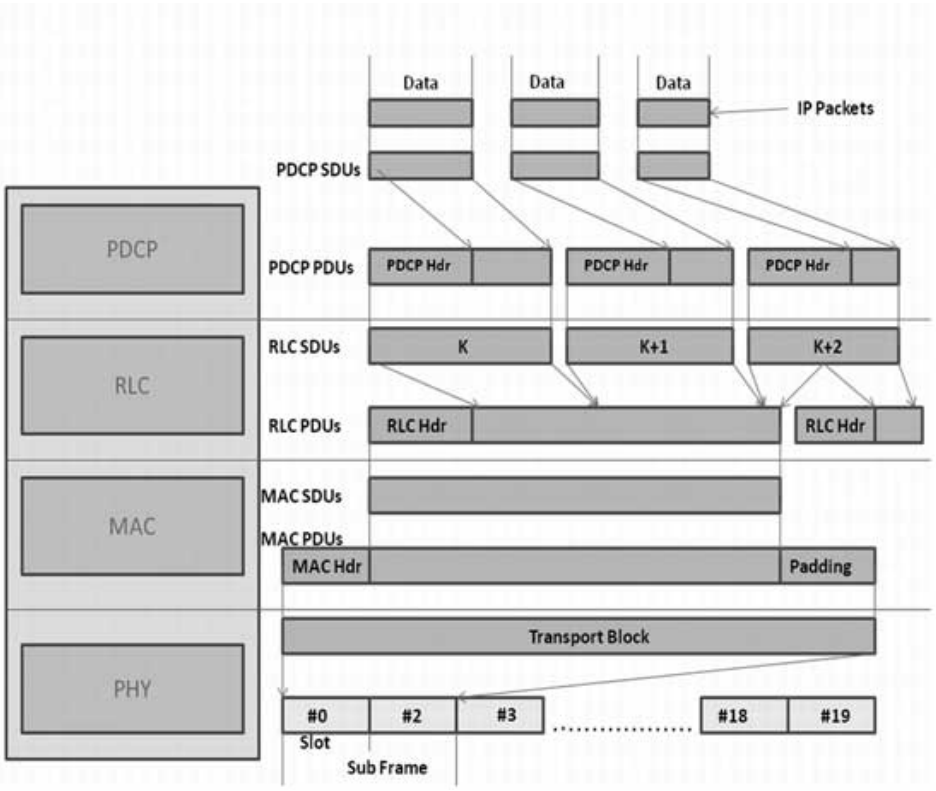

Figure 2.10: Data flow through the LTE radio protocol layers, from[31]

2.4 LTE Radio Protocol Stack

Both user data coming from the user plane and control signals from the Radio Resource Control (RRC) on the control plane pass through the entire *Radio Protocol Stack*. The relevant functions of each the Packet Data Convergence (PDCP), Radio Link Control (RLC), Medium Access Control (MAC) and Physical layer are described below in order. Note that *transmitting* in the context of all but the physical layers means to submit the data to lower layers.

When data moves down the layers, each non-physical layer adds its own header segment to the data in order to communicate with the corresponding layer on the receiving side. A data packet that is received from a higher layer is call a Service Data Unit (SDU), whereas when it has added its own header to the SDU(s) it is called a Protocol Data Unit (PDU), which is sent toward lower layers. The flow of data units through the LTE radio protocol stack is illustrated in figure 2.10.

2.4.1 Packet Data Convergence (PDCP)

The Packet Data Convergence (PDCP) protocol ensures that *handover* procedure is relatively seamless by preventing packet loss by sending status reports between the old and new eNB. The PDCP layer makes sure that packets pending transmission in one cell to terminal connection, are

transmitted in the new cell to terminal connection. This way all packets will be ensured to be delivered to the endpoint.[15]

PDCP also ensures that all packets are delivered in order up to higher layers.[15]

Header Compression

Sending data over the air interface can be expensive in terms of time and resources, so it is desirable to not send more then necessary. Large header sizes can become a problem when reaching the PDCP layer, especially when considering the large 128 bit address spaces for IPv6, and when attached to each data packet the transmission of headers across become a significant portion of the radio link bandwidth. To combat this, the PDCP protocol takes use of the RObust Header Compression (ROHC)[27]. In essence, ROHC sends the full header on the first transmitted packet, but then only sends any differences in the header for later packets. This greatly reduces the overall overhead of the user data.

2.4.2 Radio Link Control (RLC)

The Radio link control layer is a layer 2 protocol in the Air interface, located below the PDCP layer and above the MAC layer with the responsibility of transfering data between these two layers. Main RLC functions as specified in [5] include error correction, in-sequence delivery of upper layer PDUs, duplicance avoidance and reordering.

In comparison its neighbouring layers the PDCP and MAC, the RLC hold a great responsibility in how to handle packets in regard to the overall data flow. The only service RLC expects from the MAC layer is data transfer, meaning RLC will have to account for any sequence misordering as a result of HARQ operations on the MAC. However it isn't given that the end-to-end data flow favors the delay that would come with reordering and error correcting procedures, and would rather benefit from fast delivery of the delivered packets and discardment of those lost. Therefore the RLC contains three modes of operation, Transparent Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM) each of which use their own RLC entities found on this sublayer. The mode is configured per session by the RRC.

Transmission Mode (TM)

In RLC TM, no header information is added by the TM entity (passing transparently through the RLC layer). Typical usage of this mode includes RRC and cell updates on logical channels like BCCH/PCCH/CCCH. It's architecture illustrated in figure 2.11 is very simple.

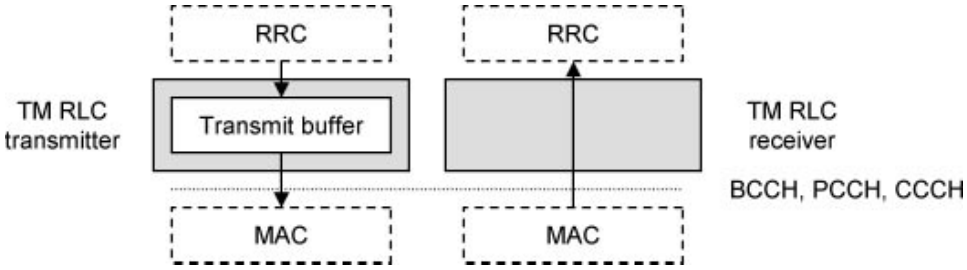

Figure 2.11: Transmission Mode (TM) architecture, from [11]

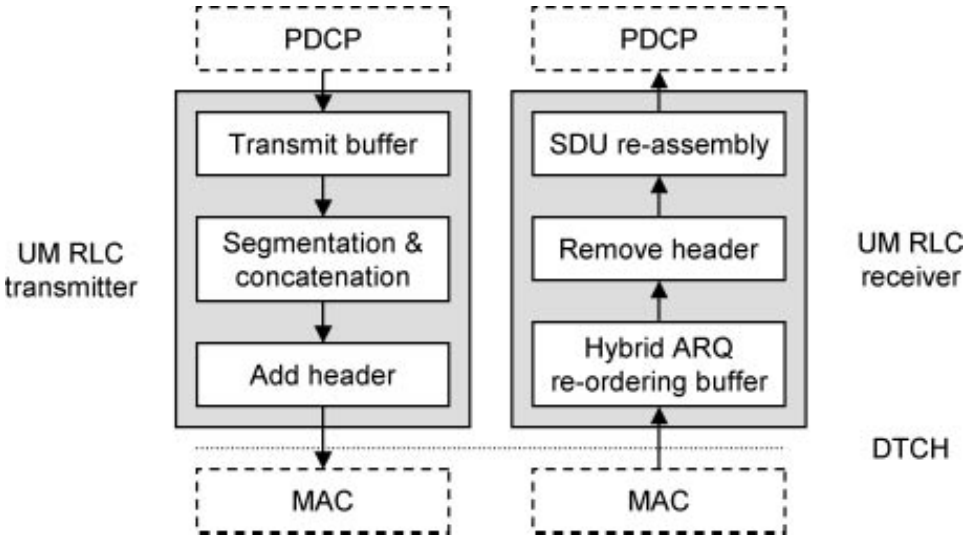

Figure 2.12: Unacknowledged Mode (UM) architecture, from [11]

Unacknowledged Mode (UM)

RLC UM is the most simplistic of the two data transfer modes. Its architecture illustrated in 2.12 shows the three main functions enabled on this mode for both the transmitting and receiving entity. PDCP PDUs coming from the PDCP are put in a transmission buffer. The RLC communicated with the MAC layer through control signals and waits for the MAC to report a transmisson opportunity on the link large enough to send either one, multiple or segmented RLC SDUs. When such a payload is prepared the RLC header is added to make a RLC PDU and is sent to the link. On the receiver the RLC may recieve these RLC PDUs out of order due to the Hybrid-Automatic Repeat Request (HARQ) retransmission scheme on the link. Since the RLC needs to deliver the received RLC SDUs in order to the PDCP, the received data units are buffered and reordered before they are stripped of their RLC headers and possibly re-assembled if they are segmented.

Acknowledged Mode (AM)

RLC AM implements the same functions as RLC UM but with much more extended functionality. While RLC UM does reorder out-of-order PDUs

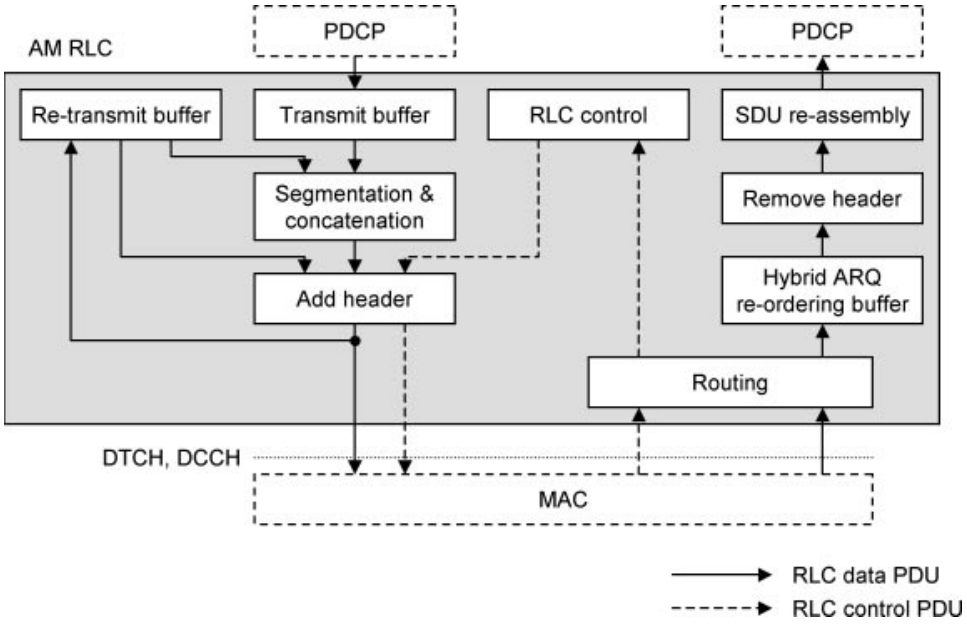

Figure 2.13: Acknowledged Mode (AM) architecture, from [11]

received from the MAC it does not account for lost PDUs. The RLC AM mode provides reliability by accounting for such losses by implementing Automatic Repeat Request (ARQ) functionality to signal the transmitting RLC AM entity that lost PDUs should be retransmitted. In order to front the communication between the RLC AM transmitting and receiving entity, RLC AM differentiates between data and control PDUs through setting a dedicated bit in the RLC header. Data PDUs are handled as normal while control PDUs are used for internal information. An RLC AM entity can request a stats PDU (type of control PDU) from the entity it is communicating with by setting a *polling bit* in the header, which is done at regular intervals, also called *polling*. The architecture of RLC AM is illustrated in figure 2.13. In addition to the tranmission buffer, the RLC AM contains a re-transmit buffer, where copies of transmitted PDUs are temporarily stored. If a control PDU signals that a data PDU must be retransmitted, it is extracted from the retransmission buffer.

For applications using TCP for purposes like emails or web browsing, RLC AM is desireable for the robustness and reliability provided by the ARQ function. However this induces delay from the control overhad as opposed to cutting your losses. For applications like video streaming or voice using protocols like VoIP or UDP, RLC UM is much more desireable for such RTT sensitive flows that would rather skip the ARQ.

RLC Automatic Repeat Request (ARQ)

The RLC ARQ uses much overhead from the control signaling as opposed to other ARQs, but the principle of the error correcting scheme remains

the same. The RLC ARQ is a case of a *selective repeat* ARQ, meaning that lost frames will be selectively NACKed so that the transmitted only needs to retransmit the selected ones as opposed to ARQ instances like *Go-Back-N* or *Stop-and-Wait* where the transmitter needs to retransmit all frames higher than the one that was last acknowledged. This is very similar to TCP retranasmissions using Selective Acknowledgement (SACK) versus cumulative acknowledgements.

As an example, a transmitting RLC AM entity has a sending window of five and sends data PDUs with SN(1) to SN(5). Out of these five SN(3) and SN(4) are lost in transmission, the receiving RLC AM entity acknowledged the highest SN(5) but also sends a NACK for SN(3) and SN(4). The tranmitter advances the sending window by two since it know SN(1) and SN(2) were delivered, then retransmits SN(3) and SN(4) along with new PDUs SN(6) and SN(7) since the window advanced.

One might wonder why HARQ reordering is done on the RLC layer and not on the MAC layer itself. This is a key feature of the Evolved Universal Terrestrial Access Network (E-UTRAN) architecture in LTE, in Universal Terrestrial Access Network (UTRAN) reordering is done on the MAC layer. The disadvantage of reordering on the MAC is additional buffering of packets. Since the RLC already keeps track of sequence numbering and uses transmission buffers, offloading such buffering from the MAC to the RLC reduces the overall architecture complexity.

Segmentation and Concatenation

Complete PDCP PDUs in the transmission buffer go through the process of segmentation or concatination if necessary before being transmitted. Higher layer data unites such as IP packets and PDCP PDUs maintains a "static" size of one packet per data unit, whereas the RLC is at the mercy of link layer transmission opportunities in addition to the size of the PDCP PDUs. When the quality of the radio link is good, there are large transmission opportunities from the MAC and the RLC can send multiple RLC SDUs and does so by concatenating them together into one RLC PDU. When the quality of the radio link is poor, the transmission opportunities may be smaller than the size of a PDCP PDU, and the RLC needs to segment the RLC SDUs into incomplete segments and transmit them, needing the receiver to re-assemble the segments back to a complete SDU. This is illustrated in figure 2.10 from the previous section.

2.4.3 Medium Access Control (MAC)

The Medium Access Control (MAC) protocol, specified by the 3GPP in [3], schedules transmissions that are carried out on the air interface and controls the low-level operation of the physical layer.[11] Data is received from the RLC layers in the form of MAC SDUs. These data units can then be combined through multiplexing and reassembled through demultiplexing, similar to the segmentation and concatination process

on the RLC. Important features of the MAC layers include buffer status reporting, multiplexing/demultiplexing of data units and transmission scheduling.

Buffer Status

The MAC layers on an UE device needs to inform the base station of how much data is available for transmission. It does so by sending a *buffer status report* of the current state of the transmission buffers. This is done both periodically, and for cases regarding the transmission buffers. A status report is sent if new data becomes ready for transmission, a higher priority channel has new data to send or for timeouts for pending transmissions.

Multiplexing and Demultiplexing

In similar fashion to the other layers, the MAC takes one or more RLC PDUs as MAC SDUs and appends its header to make a MAC PDU for transmission, displayed in figure 2.10. In addition to the MAC SDUs the payload contains control elements, such as buffer status report, power signals, timing advances etc. The MAC protocol data unit differs from that of the other layer in that the payload may contain padding at the end, in order to round the size of the data unit up to the current Transport Block (TB) size. In addition each SDU in the payload is assigned its own header that hold information of the logical channel the SDU originated from and the its size.

Scheduling

The MAC uses a scheduling algorithm, for which there are many different versions, to distribute the resources among the users. A base station MAC scheduler needs to gather information about its UEs like buffer status reports and Quality of Service (QoS) to decide how to distribute resource allocation. There is scheduling for uplink and downlink, where both output values like transport block sizes permitted to UEs and modulation schemes. MAC SDU sizes are calculated on the downlink scheduling.

Different MAC scheduling algorithms vary between acheived throughput and fairness.[11] A high throughput scheduler will allocate resources to UEs with high signal-to-noise ratios to allows them to utilize the highest data rates, maximizing cell throughput but aldo proves highly unfair to UEs experiencing poor signal conditions. Schedulers aiming for fairness such as a *round-robin* approach giveas each UE the same data rate, being very fair but very inefficient in terms of cell throughput. Other algorithms attempt to find a balance point inbetween these two extremes.

Hybrid-Automatic Repeat Request

The MAC performs error correction on the link, and does so very efficiently through the Hybrid-Automatic Repeat Request (HARQ) mechanism. The

following describes the HARQ operation and the concepts around it.

HARQ is a technique which combines features from Forward Error Correction (FEC) and Automatic Repeat Request (ARQ) [17]. FEC adds a *code word* to the data. The code word is larger than the actual data, and the two are intertwined in such a way that even when it arrives at the receiver with errors the receiver is still able to extract the actual data from it. A modulation scheme is needed to decide the comparative size of the code word to the data, which makes an important tradeoff between throughput and error management. Adding more bits for error checking reduced the available bandwith for the actual data but provides greated error correction, and less bits for error checking gives more data bandwidth but reduces the probability of correcting errors. Error management in ARQ checks for errors but without the means to fix them. The ARQ takes the data and computes a Cyclic Redundancy Check (CRC), a bit sequence based on the data it was computed from. The CRC is attached to the data and transmitted. At the receiver, the receiver calculates a CRC from the transmitted data and compares it with the attached CRC. If any of the CRC bits are different, the transmission has errors and a retransmission is requested. **TODO: UM vs AM** "In HARQ, the actual data is encrypted with a FEC code, and parity bits are sent upon request or immediately sent along with the message when erroneous data is detected" [17]. In other words, HARQ detects erros upon reception, identifies which parts of the data was erroneous and retransmits only these missing bits instead of retransmitting the message as a whole.

The HARQ operation is either *synchronous* or *asyncrhonous*. Synchronous HARQ processes retransmissions periodically over a certain time interval. Asynchronous HARQ retransmissions have no timing constraints and are triggered via explicit signaling, giving better scheduling flexibility at the expense of increased overhead due to signaling. HARQ can also be either *adaptive* or *non-apaptive*. Adaptive HARQ dynamically adjusts scheduling parameters such as code rate or resource allocation, allowing the scheduler to properly adapt to the state of the radio link but again at the expense of information overhead. Non-adaptive HARQ uses fixed sizes and formats, having less overhead but may be sub-optimal for scheduling. When the HARQ receives erronous data, the valid bits of the data are buffered until the parity bits are retransmitted to be combined with the buffered data. *Chase combining* is a type that uses less error-correcting coding, correcting errors for good signal conditions, but in poor signal conditions there are not enough error-correcting bits to repair the errors and a retransmission is requested. *Incremental redundancy* uses more error-correcting bits, but each retransmission contains multiple sets of code bits from the same data bits, using different redundancy versions of the sets. This makes it sufficent for the receiver to repair losses and retransmissions are only made in poor signal conditions and if the data was not decoded correctly.

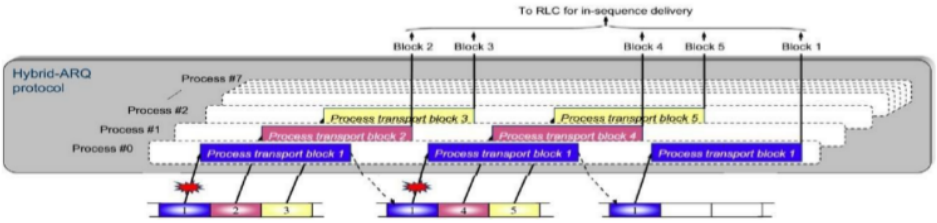

Figure 2.14: HARQ protocol with eight processes, from [17].

ACKs and NACKs are sent in the *stop-and-wait* fashion, waiting for an (n)acknowledgement before proceeding, which induces delay. To reduce delay, multiple HARQ processes can run in parallel, processing one transport block each. There are eight HARQ processes in the uplink and up to eight on the downlink. This is illustrated in figure 2.14. As a result of using multiple unsynchronized HARQ processes, MAC SDUs may be delivered out-of-order to higher layers. The RLC ensures in-sequence delivey and accounts for this misordering.

2.4.4 Physical layer (PHY)

The layer 1 physical layer is the lower endpoint of the LTE protocol stack. The LTE PHY is a highly efficient means of conveying both data and control information between an enhanced eNodeB and mobile UE [35]. The physical layer needs to transmit data in a fashion that satesfies the QoS dictations from the RRC on layer 3, as well as adapting to the quality of the channel. While higher layers transport data in sequence of packets, LTE PHY transports data in frequence-domain to meet its requirements of efficiency and resource distribution.

Since there usually are multiple UE devices connected to a eNB base station, the LTE PHY uses functions like the Orthogonal Frequency Division Multiplexing (OFDM) to communicate with several mobile devices at the same time, while also being a powerful way to minimize interference errors [11]. An Orthogonal Frequency Division Multiplexing (OFDM) transmitter transmits a data stream over multiple sub-streams instead of one single stream, each of these data sub-streams on a different frequency. This helps protect the data from interference, as interference on a given frequence will only affect one sub-stream and instead of knocking out an entire single data stream. As OFDM divides the bandwidth in frequency-domain, LTE frames are divided in time-domain. An LTE frame has a length of 10ms and cosists of ten sub-frames, each of which with a length of one ms [23]. This time-domain structure is illustrated in detail in figure 2.15, and how this fits into the overview of the LTE stack in figure 2.10 from previously.

These alloted frequencies, also called *sub-carriers*, and time-domain data slots are then scheduled using a Resource Block (RB), illustrated in figure 2.16. The LTE bandwidth is a result of how many RBs are allocated, which

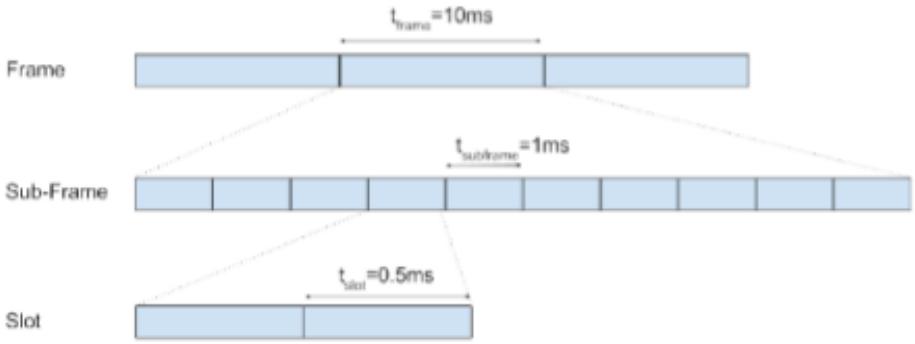

Figure 2.15: LTE frame structure, from [23]

can be up to 100 RBs.

2.5 Noise and Interference

Digital transmission over a wireless medium are prone to variable and often intense error rates and delay. From the source of the transmission antenna and towards the destination receiver, signals are exposed to external interference that can obstruct the encoded information. Atmospheric conditions such as noise on the transmission channel or factors like path propagation and fading due to large distances and blocking objects can cause severe error rates for wireless links. While technologies like Wireless Local Area Networks (WLAN) are susceptible to wireless induces errors, LTE and *Satellite* are even more affected given the scale of factors such as distance and mobility changes. Following are some of the key generic characteristics of wireless systems.

2.5.1 Signal transmission and reception

The data in need of transmission are represented by bits. On the physical layer these bits are encoded by a *modulator* into a radio wave signals. This is done by computing the bits into *symbols* that represent the amplitude and phase of the radio wave. On the receiver, the radio wave is picked up by an analogue antenna and the *symbol* are extracted from the amplitude and phase of the radio wave. However during symbol extraction it isn't given that the radio wave hasn't been tampered with by noise of interference. For distorted waves, the intended symbol may not be clear given the amplitude and phase. With variable levels of confidence the receiver will have to decide whether the symbol might be a 0 bit or a 1 bit. For heavy noise and interference, the receiver might misinterpret a 0 bit for a 1 bit or vice versa, causing errors.

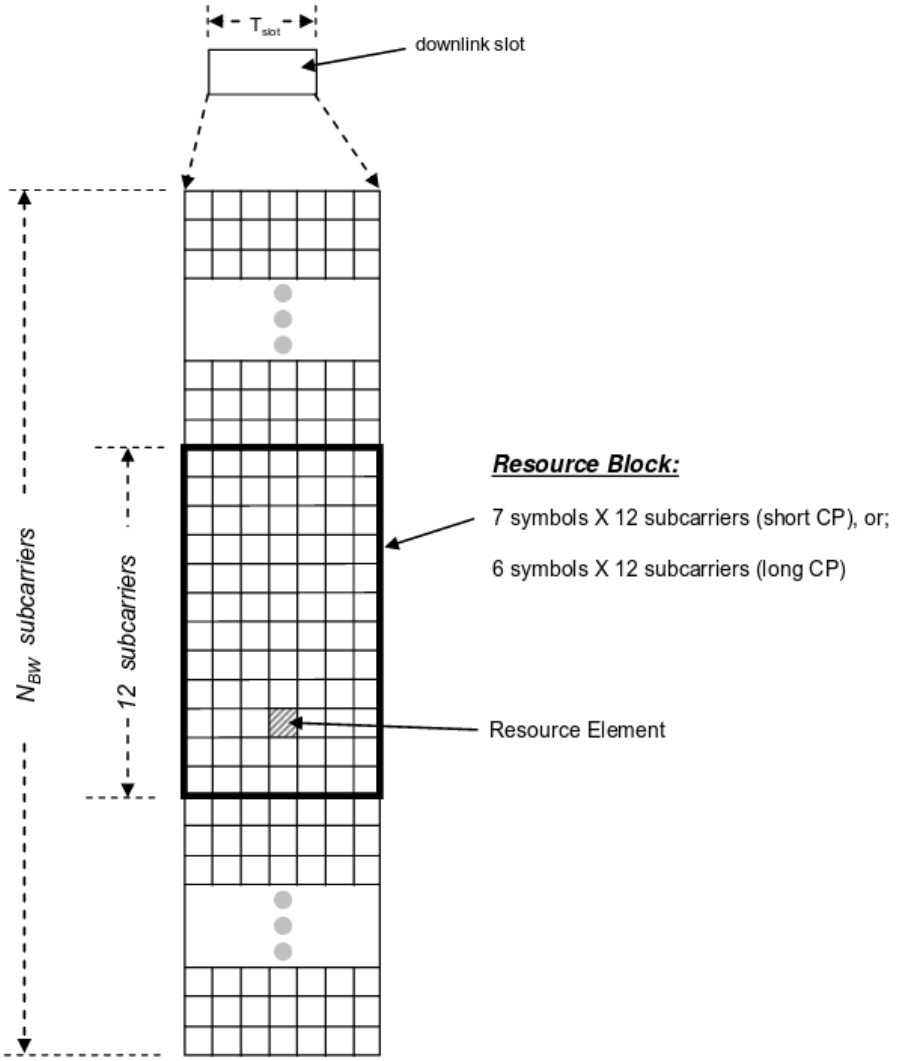

Figure 2.16: LTE downlink resource grid, from [35]

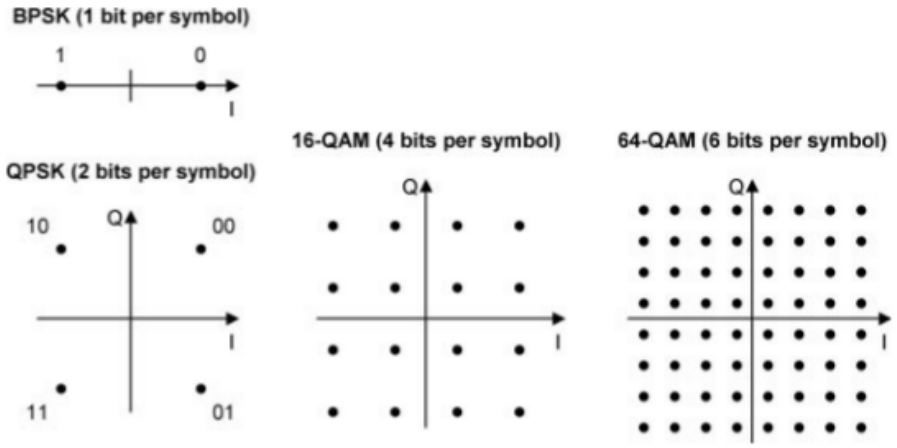

Figure 2.17: LTE modulation schemes, from [11]

2.5.2 Modulation schemes

A modulation scheme converts the digital bits into the form of a radio wave. The simplest example is to represent one bit per symbol, having starting phases of 0° and more as a 0 bit and 180° and more as 1 bit. The downside with this example is that is highly inefficient since transmission delays can be costly, and it would be more efficient to represent more bits per symbol. Figure 2.17 shows the modulation schemes used in LTE, where the Quadrature Amplitude Modulation (QAM) schemes are normal for data transmission. While a high number of bits per symbol is desirable, it comes with a disadvantage of being highly error prone, as noise affected radio waves can cause the receiver to misinterpret not just on bit but a whole bit sequence. Because of this, LTE dynamically changes its modulation scheme based on the condition of the radio link, using 64-QAM when there is good signal conditions and schemes more tolerant to errors like QPSK for poorer conditions.

2.5.3 Propagation

As a signal is emitted on the wireless link, the signal spreads out as it travels away from the transmitter. This causes the received power of the signal to be less than when it was when it was initially transmitted. The ratio of the two is called the *propagation loss*, also called *pathloss*. The signal expands in a spherical fashion, making the pathloss proportional to the $radius^2$ [11].

$$\text{Pathloss} = \frac{Power(Transmitted)}{Power(Received)}$$

A signal can also be reflected or blocked by object along its path, which affects the propagation loss.

2.5.4 Fading

While an ideal signal travels straight from the transmitter to the receiver, signals can reflect and "bounce" of to different paths and eventually end up at the destination receiver. The receiver may therefore receive colliding signals, that either reinforce each other as *constructive interference* or cancel each other out as *destructive interference*. Destructive interference causes the signal power to drop significantly, know as *fading*. Fading increases error rates as the low power signals cause errors for the modulation scheme.

2.5.5 Metrics

Various metrics for measuring noise and interference is needed both for performance analysis and channel quality indications for devices. The following sections describe some measurement quantities used for LTE.

Noise Figure

Noise figure is a measure of how much a device (such an amplifier) degrades the Signal to Noise Ratio (SNR). For radio signals that go through multiple hops via amplifiers, the amplifiers will amplify he input signal but in the process also the noise from the input signal in addition to the noise figure of the device itself.[25]

Signal to Interference and Noise Radio (SINR)

The error rate depends on the Signal to Interference and Noise Radio (SINR) at the receiver.

Chapter 3

Design

3.1 Hypothesis

TCP deems a packet lost after a certain time with no delivery at all (RTO) or after receiving a number of subsequent dup-ACKs. These cases have to deem packet loss because the transport layer cannot ever know if the packet was actually lost or not, and not just delayed by a re-transmission or reordering. In contrast, the ARQ on the link layer can identify actual losses and the exact causes such as congestion or re-transmission errors. With the addition of RACK fast recovery to TCP, RACK reduces the risk that a packet is deemed lost incorrectly due to reordering in contrast to dup-ACKs. The link layer ARQ is better suited to deal with transmission errors on poor radio link, but the ARQ process is a cause of reordering itself, requiring delay inducing reordering timers to avoid triggering false packet loss assumptions at the transport layer.

This thesis proposes a scheme for RACK and link layer ARQ to complement each other. RACK should ease the pressure on the ARQ to satisfy in-order delivery by giving it more time to fix errors detected on the link. In response, the ARQ needs to feed RACK with packets to avoid triggering RACK timeouts, which can be done by immediately forwarding complete packets out of order instead of holding them in the transmission windows for reordering. In theory, this scheme hopes to improve throughput and reduce latency for situations where packets would otherwise be deemed lost.

3.1.1 Throughput

RACK reduces the number of spurious congestion reductions due to packet reordering as opposed to using dup-ACKs. This also allows the ARQ more time to repair losses, letting the ARQ reduce congestion reductions due to actual transmission losses. Combined these two aspects should improve throughput by avoiding unnecessary congestion windows reductions in the TCP flow.

3.1.2 Latency

On the link layer, forwarding packets out of order without waiting for the ARQ to put them in order does not deliver the in sequence packets to the application any faster. However, giving ARQ more time to repair transmission losses reduces the delay that would otherwise occur as a result of re-transmitting the packet end to end. This could be particularly useful for short flows, given their completion time is more sensitive to individual losses.

3.1.3 Applicability

In the standard dup-ACK scheme will deem that it needs to re-transmit a packet earlier than RACK will, given the notion of time versus sequence counting. So if a packet loss was in fact due to congestion in the network, dup-ACK have re-transmitted the packet faster than RACK. But if the loss was falsely deemed lost, RACK and ARQ will be the victor.

The proposed scheme is therefore aimed at networks that experience a significant level of errors and reordering that will end up producing false negatives for the TCP congestion control algorithm. Wireless systems that deploy ARQ functionality are prime candidates for this, both due to the erroneous nature of a radio link but also the cause of reordering from the ARQ. This thesis focuses on simulating an LTE network, but this theory could also be applied to other radio link systems such as Wi-Fi and satellite.

3.2 Related Works

3.2.1 XLR8

The concept of forwarding buffered packets out of order using RACK is not a new idea. It was proposed in *XLR8: Accelerating Beyond 5G*[23], which addressed the struggles of transport protocols like TCP or QUIC over radio channels. The objective was to deliver enduring solutions to slow internet protocol acceleration for better utilisation for a varying radio channel, and to enable data flows to accelerate in constant time up to whatever peak rate the underlying radio technology can deliver.

While the XLR8 proposal addresses a larger scope of TCP related bottlenecks in LTE such as capacity variability and receive window, it also addresses transmission losses as a significant reason for poor end to end TCP performance, due to its inability to distinguish transmission losses from congestion losses and HARQ timeout limitations.

XLR8 proposes to use Active Queue Management (AQM) to integrate internal buffers for better control and to reduce the aggregated queuing delay, illustrated in 3.1.

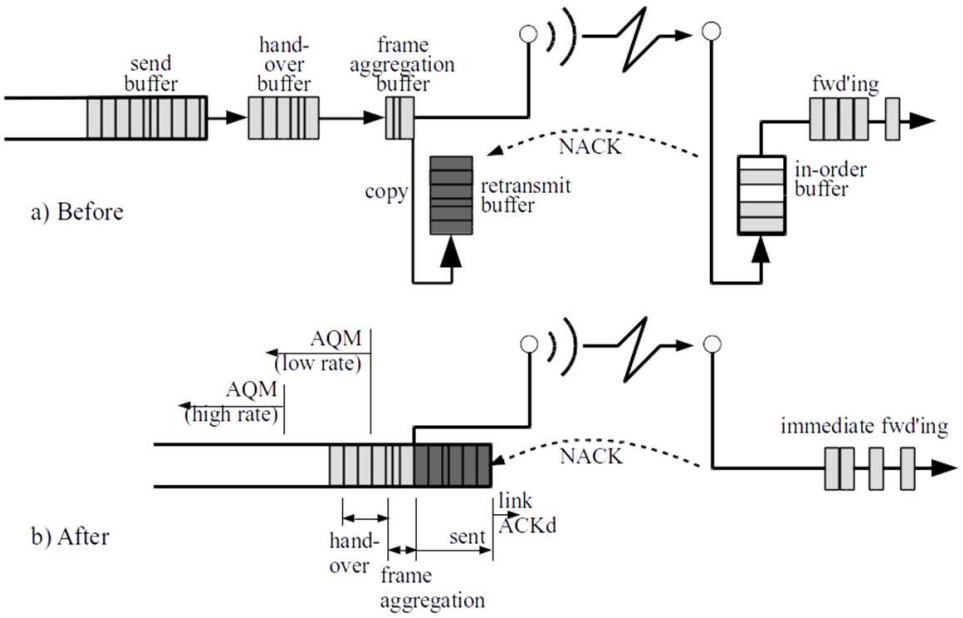

Figure 3.1: a) Today; b) XLR8: Integrated Buffer Control and Active Queue Management.[23]

As seen in figure 3.1, the *in-order buffer* is proposed removed and immediately forward packets on the receiving side (proposed for 5G++). XLR8 recognises the possibility of using RACK to comply with immediate packet forwarding.

On the subject of RACK, XLR8 states that: "It would seem that the radio link cannot yet exploit this advance, because it will not be able to depend on RACK being in use until it has become universally deployed in TCP (and a similar approach in other reliable transports such as SCTP and QUIC). This could take decades."[23] However the proposal mandates RACK for all transport protocols that use Low Latency, Low Loss, Scalable Throughput (L4S). Today, RACK is enabled by default in Linux.

3.2.2 Improving TCP Performance Over Mobile Data Networks

Originally designed for fixed(wired) networks, TCP struggles to utilize the throughput potential in 3G and LTE networks that have fundamentally different properties then fixed ones. In recognizing this, [18] quantifies how much TCP congestion control algorithms such as Cubic is able to effectively utilize the bandwidth in mobile networks, and then proposed a protocol optimisation for better bandwidth utilization without any modifications to the TCP implementations or on the end-to-end hosts.

The first step is addressing the precise reasons as to why TCP struggles in mobile networks. [18] outlines three main mobile network characteristics that deviates from the TCP design: bandwidth fluctuations, traffic flow scheduling and large link buffers. Varying bandwidth rates is a natural part of wireless networks and makes it hard for TCP to adapt its transmission rate precisely, even with no competing traffic flows. Mobile networks like

Protocols	Average TCP Throughput (Mbps)		Average UDP Goodput (Mbps)		Bandwidth efficiency
	mean	stdev	mean	stdev	
TCP Cubic	14.2	4.01	33.0	4.9	43.7%
TCP Reno	13.6	3.86			41.4%
TCP Westwood	13.2	3.62			40.0%
TCP Veno	13.3	4.03			40.3%

Table 3.1: Comparison of TCP throughput performance in a LTE network, from [18]

LTE uses MAC layer schedulers to schedule different user traffic flows. In fixed networks, there is no such traffic scheduling in routers and TCP has to enforce fair bandwidth sharing in bottleneck buffers. With base stations already enforcing scheduling, it offloads this purpose from the TCP protocol. Last but not least, base stations typically use large per-user link buffers to support link layer re-transmissions and reduce packet losses due to buffer overflows during large and rapid bandwidth fluctuations.[18] This enabled TCP to send more bytes without causing buffer overflow.

The first experiments did in [18] was measuring the bandwidth efficiency achieved for a set of selected TCP congestion control algorithms. The results are shown in table 3.1, comparing the average TCP throughput to the average UDP goodput. As seen from their results, the performance is sub-optimal, 43.7% efficiency at best for Cubic. The proposed scheme in [18] involves adding a *mobile accelerator* between the internet server and the mobile network. This accelerator intercepts the TCP logic and adds a more opportunistic flavor. Firstly, the original purpose for the TCP receiver window was to avoid overwhelming slow receivers. With todays relatively higher computing power in even mobile devices, the accelerator opportunistically forwards packets regardless of the recever window size (unless zero, at which point forwarding is suspended). TCP sending rate is also by design very conservative in regard to its congestion window. Instead, the accelerator disregards the congestion window and does bandwidth estimation using ACK timings. To make a long story short, [18] also proposed estimations in time-domain, in this case using fixed intervals of seconds and acknowledgement timestamps. Lastly, the accelerator allows the opportunistic transmission of new packets during the recovery phase, where normally TCP suspends new packet transmissions during this phase.

The end results concluded with a 97.6% bandwidth utilisation for accelerated TCP Cubic in LTE, a 53.9% increase from the previous experiments.

3.2.3 Occupancy regulation for ARQ Re-ordering Buffer

Re-ordering delay on the RLC is a product of its error control scheme based on selective-repeat ARQ. The receiving window has to be large enough to buffer a large burst of packets, but also increases the occupancy at the re-ordering buffer. In [28], the effects of windows sizes on RLC throughput and performance is investigated, along with a proposed scheme to regulate the re-ordering buffer occupancy.

[28] proposes to regulate the re-ordering buffer occupancy using a threshold value. Packets arriving that are below the threshold value will be placed in the re-ordering buffer and handled in a selective-repeat fashion as normal. Packets that go over the threshold however (but still inside the window) are handled using *go-back-N* instead, essentially discarding the packet and wait for it to be re-transmitted.

With the approach in this thesis, there shouldn't be an occupancy problem, because it's only buffering a few segments.

3.2.4 Limiting HARQ retransmissions in Downlink for Poor Radio Link in LTE

While this project proposed to increase HARQ re-transmissions if needed, [16] makes some arguments for limiting HARQ retransmissions for poor radio link conditions.

While the HARQ is a powerfull tool for link layer repairs, it also consumes radio resources through both scheduling and overhead of the process. The proposal in [16] to reduce the maximum HARQ retransmission threshold from 3 to 1 or 2 is justified by saving radio resources. When there are multiple UEs communicating with a base station, a UE with poor radio link quality will makes less use of the radio resources compared to a UE with good radio link quality. If the UE with poor radio link quality reduces its number of HARQ retransmissions it will reduce its throughput but free resources that can be better utilized by the other UE with good radio link quality. There is also the case of failure to repair the transmission even after 3 HARQ retransmissions, in which case those retransmissions are considered to be wasted. In regard to robustness for the TCP protocol, failure to repair the transmission on the MAC can still be recovered by the RLC ARQ.

In simulations testing error rates versus SNR for low CQI values, [16] concludes that a HARQ retransmission threshold of 3 or higher does does improve the error rate, but not substansially enough to justify the unfairness in resource allocation that comes with it. As this thesis opens for the opportunity to increase the HARQ retransmission threshold to give the RLC ARQ more time to repair errors, it is noted that doing so could negatively impact other UEs by taking more of the shared radio resources.

3.2.5 Advice to link designers on link Automatic Repeat reQuest (ARQ)

An optimal design for the protocol performance on one layer may be affect protocol performance on other layers. For example, TCP designers need to be aware that the IP protocol operates over a diverse range of underlying subnetworks on its route. [14] outlines some important considerations regarding link ARQs impact on the performance of higher layer TCP or UDP protocols.

Perfect reliability is not required for IP networks for optimal performance, and thus may drop packets due to a number of reasons like queuing, faults etc. It has long been widely understood that perfect end-to-end reliability can be ensured only at or above the transport layer, enter the end-to-end principle.[26] ARQs on the link can be used to make the link comply with the needs of the higher layer protocols such as reordering and delay, to make sure these upper layers do not have their performance degraded by link layer operations. [14] discusses the pros and cons of using different ARQ implementations for transport protocol performance.

ARQs can be implemented with variable persistence. A perfectly persistent (reliable) ARQ, which is the type implemented in the RLC ARQ in LTE, provides reliable service to the upper layers, using high degrees of reordering and retransmissions. If a the ARQ fails to retransmit a lost frame, the ones that are received out-of-order are discarded instead of forwarded in order to avoid inducing congestion avoidance signals at higher layers. Such reliability comes at the expense of high end-to-end delay, which can be desireable for TCP but not for non-reliable protocols such as UDP. Non-reliable transport protocols instead benefit from ARQs with lower persistence, with lower timeout thresholds for reduced delay and less guarantees for reliable delivery. This poses a problem when using multiple flows through the same ARQ, as different protocols on different flows may require different implementation schemes.

In conclusion, [14] advises ARQ designers to consider the implications of their design on the wider Internet, as it can be very hard to generalise ARQ schemes for transport protocols, especially considering that there may be multiple ARQs along the end-to-end network route. Also, the approach of being able to identify flows at the link layer is noted, which would enable ARQs to implement low persistancy but high-persistancy mechanisms for TCP specifically. Algorithms to implement this remain an area of research[14], and is also required to realise this project in its full potential.

3.3 Proposed Scheme

The proposed scheme of using RACK with a modified link error-correcting procedure aims at examining the measures of the effect this scheme will have on the performance of a number of TCP flows in comparison to the default schemes and congestion control algorithms. The design is not in-

tended to be something complete that could be deployed immediately. Rather, it's purpose is to apply the previously presented idea to incrementally focus more packet re-transmission instances from the end-to-end transport protocols to the link protocols. Since the scope of the design of this scheme is narrowed down to such an extent, it would be biased to compare its results directly to default schemes that account for a much wider scope and applicability. Therefore this section also addresses limitations such as technical requirements and use cases that would be needed for this scheme to truly level with comparative schemes.

The key modification required for the proposed scheme is packet forwarding to bypass the delay from the HARQ reordering procedure. Additionally, tuning of timers such as reordering timers and increasing retransmission thresholds are considered. Lastly technical requirements and proposed solutions to extend the applicability of the scheme are discussed.

3.3.1 Immediate Forwarding

The end goal of this scheme is to make sure that complete IP packets are immediately forwarded up to higher layers to reach the TCP protocol, bypassing any HOL blocking. However these IP packets are encapsulated into data units in lower layers going down the LTE protocol stack, meaning that forwarding a lower layer data unit such as a MAC PDU could potentially mean forwarding multiple IP packets, theoretically making all data units from the IP layer and below candidates for forwarding. The problem however arises regarding functions such as segmentation and multiplexing, because there is no guarantee that a data unit on layer 2 or layer 1 are divided in line with the IP packets in the payload. Such data units (PDUs) could instead contain incomplete IP packets as a result of segmentation, and it would be counter-effective to immediately forward these as they would cause errors for the network protocol and again for the remaining parts of the IP packet.

Ideally, in the fashion of forwarding IP packets, one would want to forward all PDUs. However looking down the LTE protocol stack, PDUs that can realistically be forwarded can be identified. For a clear overview of how the IP packets are packed through the LTE protocol stack, see figure 2.10 from the previous chapter. IP packets first arrive at the PDCP layer, where they are PDCP SDUs put into PDCP PDUs. The PDCP compresses the IP header using its ROHC procedure, but PDCP PDUs are still capable of being forwarded since the decompression uses the *COUNT* value which is composed of the PDCP SN [4], and the decompression of packets is not halted by out-of-order sequencing.

PDCP PDUs, containing complete IP packets, are sent to the RLC as RLC SDUs. The RLC layer may concatenate or segment this data, depending on whether the conditions of the radio link are good enough to carry multiple RLC SDUs or so poor that an RLC SDU must be segmented over

multiple RLC PDUs. Even if the signal conditions are good, RLC wants to fill up its transmission opportunity and will segment parts of an RLC SDU to utilise the remaining byte space allowed. This means that RLC PDUs should not be immediately forwarded as a whole because they may contain an incomplete RLC SDU and therefore incomplete IP packet. **However if the RLC PDU contains one or more complete RLS SDUs, these can be forwarded individually and incomplete RLC SDU segments can be buffered until new segments make them complete.** This requires management to make sure that the incomplete RLC SDU segments are reassembled correctly in regard to its sequence number and whether the segment is the first, last or even a middle part of the complete RLC SDU.

MAC PDUs containing MAC SDUs are already forwarded in the sense that they are immediately shipped to the RLC when the MAC PDU is complete because in E-UTRAN HARQ reordering is done at the RLC ARQ instead of buffering locally on the MAC layer. Going deeper to the PHY layer, the MAC PDU is re-assembled by the receiver by time-divided sub-frames. These sub-frames could in theory contain complete IP packets, but because of the time-domain there is no way to confirm that given the structural architecture of the data units. Even if one could detect IP packets within the sub-frames, forwarding these would break the entire structure of the LTE protocol stack in regard to header data and packet control.

Immediately forwarding PDUs must therefore be limited to forwarding completed RLC SDUs on the RLC layer up the the PDCP layer, with mechanisms to correctly re-assemble incomplete RLC SDU segments.

3.3.2 Threshold Tuning

Reordering timers and re-transmission thresholds are by default set to values that are high enough to allow reordering but low enough not to cause too much delay. The main limits in question are the RLC reordering timer and the HARQ maximum re-transmission limit.

When the RLC receives a RLC PDU that is out-of-sequence, it starts a reordering timer. While the reordering timer is running, any duplicate RLC PDUs are discarded and positive or negative acknowledgement is prohibited.[5] Any additional RLC PDUs that are received out-of-sequence while the timer is running are buffered in the reception window. The timer will continue to run until the expected SN arrives or until the timer expires. If the expected SN arrives, the timer is cancelled and the expected RLC PDU is delivered along with any packets of higher SN that fall in order. This is illustrated in figure 3.2, where the RLC reordering timer waits for a HARQ re-transmission.

If the timer expires, the RLC deems the expected PDU lost and start delivering the RLC SDUs that were buffered during the reordering timer. The reason the reordering timer prohibits status reporting when running is to give the HARQ more time for re-transmission before the RLC ARQ starts

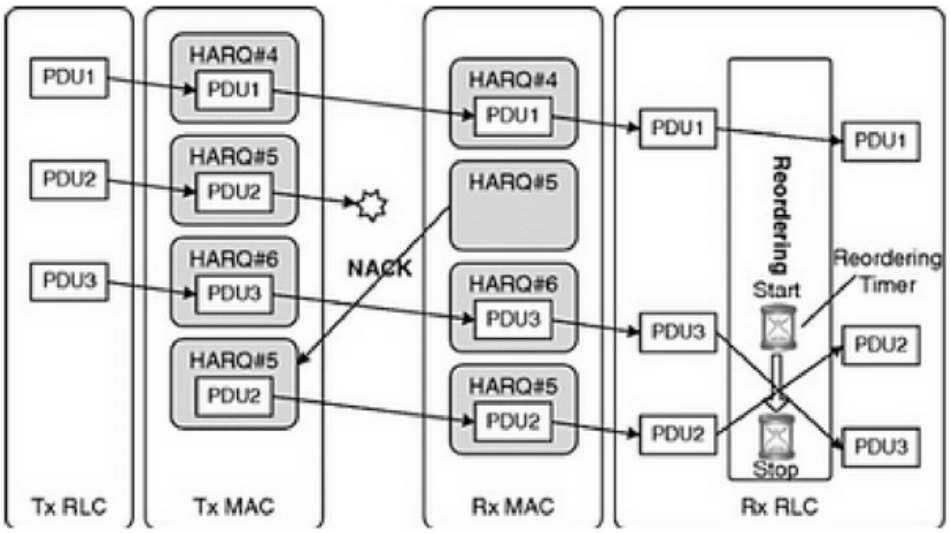

Figure 3.2: HARQ re-transmission with RLC reordering, from [34].

its own (slower) re-transmission. The RLC status prohibit timer must be equal to the RLC reordering timer. If the reordering timer is higher than the status prohibit timer, a status report could preemptively trigger a re-transmission a the transmitter for a PDU that could have arrived within the reordering timer anyways, wasting bandwidth resources. If the reordering timer is lower than the status prohibit timer, PDUs that do need to be re-transmitted after the reordering timeout will be further delayed as it needs to wait for the status prohibit timeout.

This design proposed to increase the HARQ maximum re-transmission limit and therefore the complementary RLC reordering and status prohibit timers, with the purpose of shifting re-transmission delays closer to the link and away from the higher transport layer. When increasing the HARQ re-transmission limit, the RLC reordering timer should be high enough to complement this limit, inducing higher worst case delays while putting putting more trust in the HARQ than the RLC ARQ. The increases must not be so high that they consequently add additional latency or jitter to the end-to-end flow. Ideally one would want the HARQ to wait as long as it needs to if there is in fact a re-transmission process in action, a limit must be set for situations like radio link failure or excessively long duration of interference.

3.3.3 PDCP reordering

The 3GPP specifications for PDCP in [4] states "in-sequence delivery of upper layer PDUs at re-establishment of lower layers" as one of the protocols functions. It will be no use disabling RLC reordering if the PDCP layer still buffers packets into order. Although this is limited to re-establishment of lower layers, which can be triggered by link failure or a failed handover, such a case would void the effect of forwarding out-of-

order RLC SDUs. During this process, PDCP keep a *Reordering_Window* variable with a size half of the PDCP SN space. Disabling reordering in this case could be done by for example disabling such functions that are triggered by such sequencing. However, this project does not cover the handover process, and as will be described in the next chapter the PDCP implementation of the simulator does not support such a function. This means that the experiments will automatically bypass this aspect, but it is noted as a limitation. The RLC reordering scheme will still be in place and out-of-sequence deliveries at the RLC will still trigger the reordering timer, however the out-of-sequence RLC PDUs that are buffered are not held until the reordering timeout or the reception of the expected SN. Instead, completed RLC PDUs are inspected to look for complete RLC SDUs, which are then individually forwarded. When the reordering timer times out or the expected SN arrives and start sending packets in order, the completed RLC SDUs buffered have already been forwarded.

3.3.4 RACK awareness

While immediate forwarding is designed to work for RACK, it would be severely damaging for TCP flows that do not use RACK and does depend on receiving packets in order. Therefore there needs to be a mechanism for the RLC to differentiate between whether data packets should be forwarded or reordered. This is another limitation that is not implemented but is included as theoretical proposals.

One solution could be to peek at the available IP headers from the IP packets stored in a packet payload. RACK compliant packets could be identified using the ECN bits of the IP header, assuming that an end-to-end connecting employing ECN also uses RACK. As the work on RACK progresses, and is already enabled in Linux by default, this is not an improbable assumption to make, although still not completely thorough. These identified RACK data units can then be forwarded while others can not. There is an additional problem with this method regarding the PDCP header compression. If the PDCP compresses the IP headers, lower layers are unable to inspect the IP headers. Disabling PDCP header compression would solve this but is highly undesirable, since it would add severe measures of overhead to the data transmission.

A better but more extensive solution could be to extend the LTE stack functionality. For RACK compliant connections, the RRC which assigns the RLC AM mode, could also configure the immediate forwarding scheme either through additional control variables or a modified duplicate of the AM mode. There would also have to be dynamic changes to the MAC scheduler controlling the HARQ and the mac re-transmission threshold, which could also be configured by higher layers. Packets can be identified above the PDCP layer as they would be classified into the different bearers.

Chapter 4

Implementation

4.1 NS-3

Network Simulator version 3 (NS-3) is an open-source network simulator which is publicly available for research and educational purposes.[2] It is a discrete-event simulator, meaning that it models all events in a simulated system as a discrete sequence of events. Any event occurs within it's own time instance in which it marks any changes it imposes on the system before jumping to the next event.

The simulator, written in C++, revolves around creating and configuring different *nodes* in the simulated network environment, which can represent any network entity such as user equipment, terminals, routers etc. These nodes can be configured by using the ns3 *Model Library*, by for example having a node resemble a smartphone by using the network module to install the required devices and networking stacks and the mobility model to map movement over time. Although a very powerfull tool, there are still limitations in regard to support or functionality of some technological specifications. However, being an open-source project allows users to implement and adapt their own features or extensions to the already existing ns3 implementation.

4.1.1 LTE Module

The ns3 LTE Module is used to simulate an LTE network. An overview of the module is illustrated in figure 4.1. The module is a product of two main components:

- **LTE Model**: includes the LTE Radio Protocol stack (RRC, PDCP, RLC, MAC, PHY). These entities reside entirely within the UE and the eNB nodes. This model simulates spectrum aspects that occurs on a radio link such as interference. To accurately model packet scheduling and interference, the model needs a granularity of one Resource Block (RB).
- **EPC Model**: includes core network interfaces, protocols and entities.

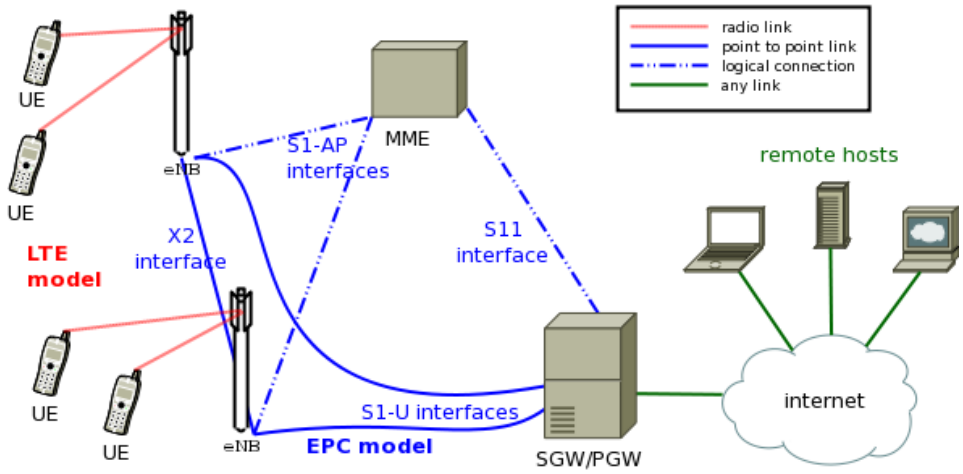

Figure 4.1: An overview of the LTE-EPC simulation model from [2]

These entities and protocols reside within the S-GW, P-GW and MME nodes, and partially within the eNodeB nodes.

Together the LTE-EPC simulation model generates a network route from a UE through whatever node is connected to the LTE gateway. Note that in this model the S-GW and P-GW are combined into a single SGW/PGW node that hold the functionality of both. This eliminates the need for the S5/S8 interface specified by the 3GPP, being the only major but harmless deviation from the LTE specifications. The other interfaces and protocols follow 3GPP specifications, even though the implemented code might be tailored differently for the scope of the simulator but generating the desired output. However some implementations of protocols may be limited by only being partially implemented.

The S1-AP and S11 interfaces are used by the control plane of the EPC model, used by the MME to control its connected base stations and gateways. On the data plane, the X2 interface connects base stations which in turn connect to the gateway using an S1-U interface. The eNB connects to the UE endpoint over the radio link. The S1-U interface between the eNB and the SGW/PGW encapsulates packets over GPRS Tunneling Protocol (GTP), UDP and IP as done in real LTE systems.[2] The overall protocol stack that a data packet transverses through the data plane from end to end is illustrated in figure 4.2.

Propagation and Fading Models

There are many different propagation models available to produce pathloss on the radio link. These models range from simple models such as pure randomness and range models, to more complex models that model environments or buildings. The pathloss is computed for downlink and uplink separately, and only for communication between UE and eNB. The Friis propagation loss mode is the default propagation model.

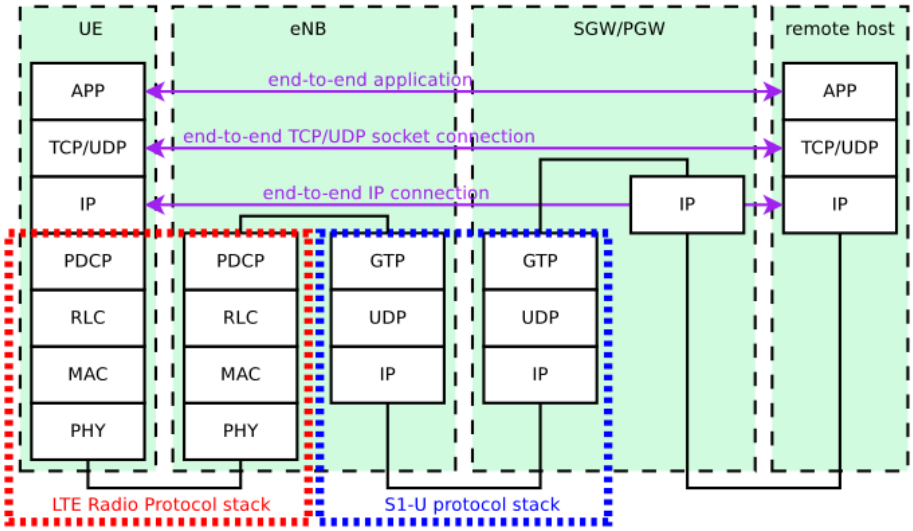

Figure 4.2: LTE-EPC data plane protocol stack, from [2]

Table 4.1: Delay profiles for E-UTRA channel models, from [6]

Model	Number of channel taps	Delay spread (r.m.s.)	Maximum excess tap delay (span)
Extended Pedestrian A (EPA)	7	45 ns	410 ns
Extended Vehicular A model (EVA)	9	357 ns	2510 ns
Extended Typical Urban model (ETU)	9	991 ns	5000 ns

The fading loss model is based on precalculated fading traces. Such traces can be generated with 3GPP specified models from [6], which uses delay profiles in the form of a 'tapped delay-line' characterised by a number of taps at fixed positions on a sampling grid, which are further characterised using delay spread and maximum delay spanned by the taps. These profiles are then used in combination with a maximum Doppler frequency to generate profiles such as the Extended Pedestrian A (EPA), Extended Vehicular A model (EVA) and Extended Typical Urban model (ETU), whose model parameters are shown in table 4.1. These models are selected to represent low, medium and high delay spread environments. In addition the ns3 fading model takes user speed as an important variable, where typical values include 0 to 3 kmph for pedestrian scenarios and 30 to 60 kmph for vehicular scenarios, both being considered urban scenarios. Three such models are provided by ns3:

- EPA: Pedestrian with speed 3 kmph.
- EVA: Vehicular with speed 60 kmph.
- ETU: Typical urban with speed 3kmph.

Spectrum Model

The radio spectrum is modelled using the carrier frequency and transmission bandwidth. The spectrum model is the base for transmission speed and error models when nodes communicate over a radio link. The spectrum model is assigned for every eNB, where different eNBs can have different spectrum models. A UE will automatically use the spectrum model of the eNB it is currently attached to.

Data transmitted over the radio spectrum on the data plane (PDSCH and PUSCH), are sent over resource blocks. The error model used over the spectrum is therefore in terms of Code Block Error Rate (BLER). BLER performance is modelled using BLER curves, that outputs the TB size given the BLER, SINR and MCS.

There are also error models for the control plane, but for this project this model will be disabled to ensure an ideal error free downlink/uplink control channel for simplicity and to avoid tampering with the data error model results.

HARQ

HARQ is implemented with Incremental Redundancy (IR) combined with stop-and-wait processes, and is enabled by default. UL transmissions are synchronous while DL transmissions are asynchronous as per the standard. There are 8 HARQ processes on the MAC layer, controlled by a specified MAC scheduler. These processes are illustrated in 4.3. The HARQ implementation is active on both PHY and MAC layers, despite begin a MAC layer technology. The PHY layers implements HARQ functionality related to its layer to support decodification buffers for IR management.

The MAC schedulers that control the HARQ processes mostly revolve around fairness among the UEs using the eNB, such as Round-Robin, proportionality or token based schemes. Since this project is limited to one UE node, the Maximum Throughput (MT) Scheduler is chosen to maximise overall throughput of the eNB, since it is not a problem in this case that the MT scheduler cannot provide fairness to UEs in poor channel conditions.

RLC

The RLC implementation includes a model for each of the three 3GPP specified entities Transmission Mode (TM), Unacknowledged Mode (UM) and Acknowledged Mode (AM). In addition, ns3 provides a Saturation Mode (SM) model, an unrealistic model for simplified cases.

The implementation interfaces with the PDCP by using sending and receiving PDCP PDUs through its Service Access Point (SAP). Similarly the RLC interfaces with the lower MAC layer by sending/receiving RLC PDUs, but in addition the MAC entity uses *NotifyTxOpportunity* to notify a transmission opportunity to the RLC entity, and the RLC uses *ReportBufferStatus* to report the size of pending buffers in the transmitted peer. This

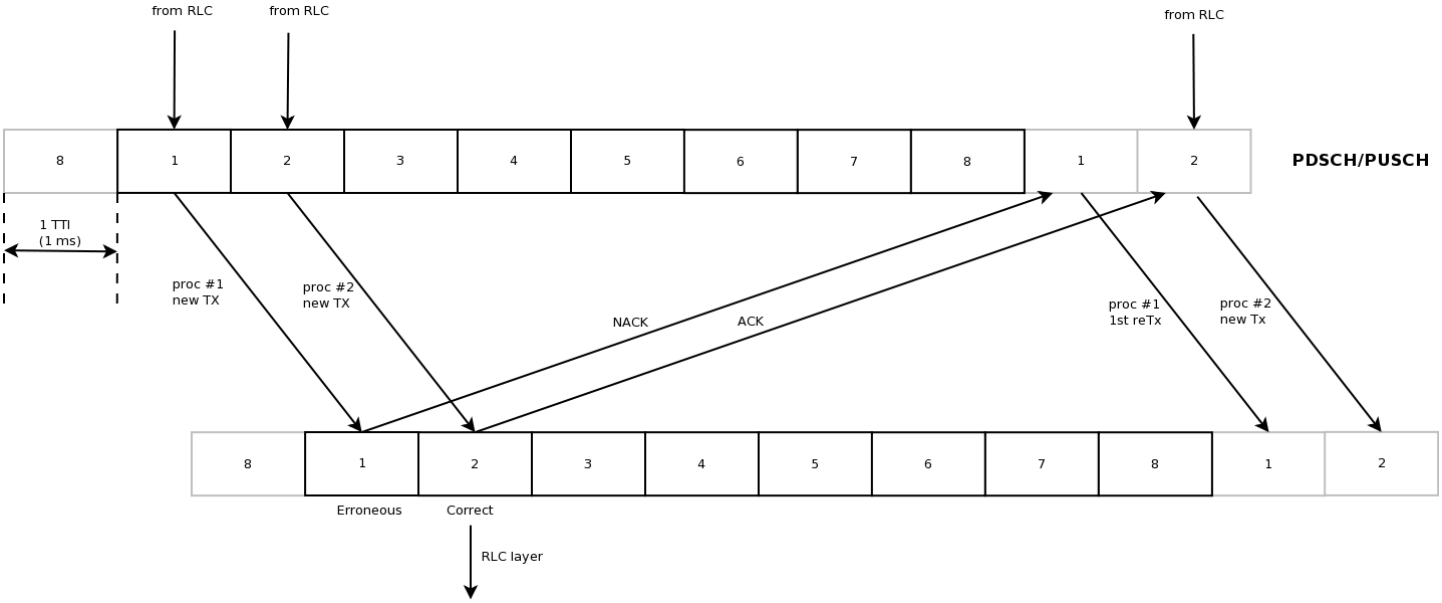

Figure 4.3: LTE HARQ processes scheme, from [2]

project only uses of the Acknowledged Mode (AM) RLC.

The ns3 implementation of AM RLC uses the following three buffers:

- *Transmission Buffer*: queue with RLC SDUs delivered from the PDCP. SDUs are silently dropped when the buffer size is full.
- *Transmitted Buffer*: queue with transmitted RLC PDUs that have not yet been ACKed or NACKed.
- *Retransmission Buffer*: queue with RLC PDUs considered for retransmission, meaning they have been NACKed or timed out.

When a RLC SDU is received from the upper PDCP layer, it is placed in the transmission buffer and a report on the buffer status is sent to the MAC for scheduling purposes. When the MAC scheduler sees a transmission opportunity it notifies the RLC, which in turn creates an RLC PDU using the buffered SDUs in the transmission buffer and copies the data to the transmitted buffer before it is send to the MAC.

In the textitns-3 implementation, AM RLC generally wants to send exactly one RLC PDU per transmission, even if the transmission opportunity is large enough to hold an additional segment. When the transmission opportunity is smaller than the RLC PDU size, AM RLC will segment and concatenate as needed to fill the opportunity. [5] specified that there is no concatenation in the retransmission buffer. The ns3 implementation follows this by not supporting re-segmentation in the retransmission buffer and rather waits for a sufficiently large transmission opportunity instead.

Currently the AM RLC implementation does not support the following procedures in [5]:

- Send an indication of successful delivery of RLC SDU.
- Indicate to upper layers that max retransmission limit has been reached.
- SDU discard procedures.
- Re-establishment procedure.

PDCP

The PDCP implementation in ns3 is simple, and only features the most basic functions needed from this layer. It supports the following procedures in [4]:

- Transfer of data.
- Maintenance of PDCP SNs.
- Transfer of SN status.

Among the procedures not supported from [4] that relates to this project are:

- In-sequence delivery of upper layer PDUs at re-establishment of lower layers.
- Duplicate elimination of lower layer SDUs at re-establishment of lower layer for RLC AM mapped radio bearers.
- Header compression and decompression of IP data flows during the ROHC protocol.
- Duplicate discarding.
- Timer based discarding.

4.1.2 DCE

Direct Code Execution (DCE) is a framework for ns-3 that provides facilities to execute, within ns-3, existing implementations of userspace and kernelspace network protocols or applications without source code changes.[1] For applications that are not already implemented in ns-3, DCE can be used to to run applications on the Linux networking stack and have its data traffic routed through the ns-3 simulator. This project uses the currently latest version DCE-1.9.

Linux Kernel library

DCE uses a kernel compiled as a library. A ns-3 simulation that needs its application to use the Linux network stack does so by linking to this kernel library. By default DCE comes with *net-next-sim*, a simplified and slightly modified kernel that compiles into a kernel library that is compatible with DCE. However *net-next-sim* is now deprecated and does not contain the required networking code for this project, so instead its successor *net-next-nuse* will be used, also called LibOS.

Most desirably one would want to use the latest version of the Linux kernel. There is a work in progress to integrate ns-3 with the Linux kernel library (LKL), which reuses the Linux kernel code more extensively, but as stated in [30] it remains uncertain when this is finished.

4.2 Simulation Setup

The core simulation setup implements a single UE communicating with a remote host over an LTE network.

The setup revolves around four nodes:

1. UE node
2. eNB node
3. SGW/PGW node
4. Remote host node

These nodes are installed and interconnected as needed to represent their intended network entity, explained in more detail below. Figure 4.4 shows an overview of the simulation nodes in action, generated using the ns-3 *NetAnim* module which traces all ns-3 nodes and their actions, outputting an animation of the simulation in *XML* format. The images representing the nodes are added for better visual representation.

4.2.1 User Equipment node

The UE node has a Linux networking stack installed, configuring networking parameter such as congestion control algorithm from the simulation parameters. LTE UE Net devices are install for communicating over a radio link, and is then attached to the eNB node. For network purposes, the node is assigned an IP address and the Linux *ip tables* are configured for routing to the remote host. Lastly the UE starts an *iperf* application, configured as a client targeting the remote node IP address. The UE node uses a constant position mobility model, and by default does not move throughout the simulation, otherwise it moves at a constant velocity.

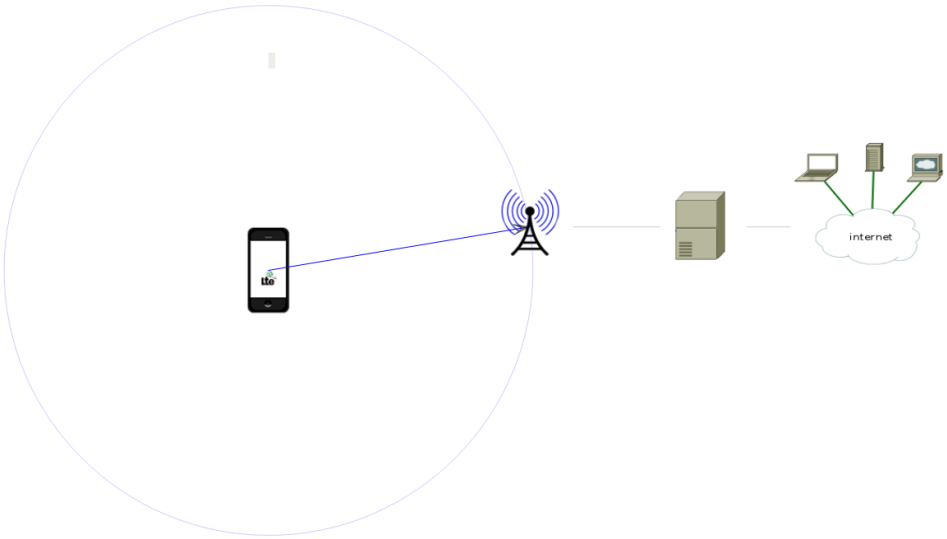

Figure 4.4: NetAnim animation of the simulation setup.

4.2.2 Evolved NodeB node

The eNB node is initialised using the LTE module. This node is responsible for modelling the radio environment, since when the UE node attached itself it inherits these configurations. Simulation parameters can override default values such as noise or spectrum properties. The eNB node sets the desired pathloss and fading model from the simulation parameters, as well as configuring the MAC scheduler.

4.2.3 SGW/PGW node

The gateway node attaches to the eNB node over the *S1-U interface* link. Net devices are installed and IP address is assigned for IP routing. The SGW/PGW node connects to the remote host node over a point-to-point link. For the simulation results to focus on LTE aspects, the point-to-point link from the gateway to the remote host is implemented as an error-free 100 Gigabit per second link to ensure that this is not the network bottleneck.

4.2.4 Remote host node

The remote host is similar to the UE node except from the LTE aspects. The remote nodes uses the Linux stack with RACK enabled by default, net devices for physical links and the point-to-point link to the gateway node.

4.2.5 Traces and Data Collection

The various devices installed on the nodes are enabled to output trace files displaying LTE radio stack operations such as number of transmitted or received bytes, delay, data unit sizes etc. Network packets are captured as *pcap* files for those net devices that allow it. Before every simulation, an

ns-3 *Flow Monitor* is installed that hooks onto packet flows observed under the simulation, outputting end-to-end statistics.

4.3 HARQ Re-transmission Limit

The HARQ retransmission limit in ns-3 is hard-coded to a constant of value 3, the 3GPP specified threshold. The different HARQ schedulers all have their own implementations, all with the retransmission limit hard-coded. To modify the retransmission limit, we must modify the scheduler flavor that is being used in the main simulation program, which in this case is the *Time Domain Maximize Throughput* scheduler.

The retransmission limit is implemented as a simple *if*-statement, which drops a HARQ process if it triggers this case. In order to increase the HARQ retransmission limit, the constant is simply changed to the desired value. This is done for both instances of the *if*-statment, one for the uplink function *TdMtFfMacScheduler::DoSchedUlTriggerReq* and the downlink function *TdMtFfMacScheduler::DoSchedDlTriggerReq*.

The optimal solution to increase the retransmission limit would have been to change the constant into a run-time attribute configured by the main simulation program. However due to simplicity and time constraints, several compiled versions of the simulator with different threshold values from 3 to 8 were implemented.

4.4 RLC AM Immediate Forwarding

The implementation of immediate forwarding completed RLC SDUs is done by modifying the ns-3 source code files for the receiver side of the RLC AM, namely *lte-rlc-am.cc* and *lte-rlc-am.h*. The changes involve modifications in of the *void LteRlcAm::DoReceivePdu* function, and the addition of the *voidLteRlcAm::Forward* function which forwards complete RLC SDUs and handles re-segmentation. Before describing the details, an overview of the original code is presented as a basis for the modifications.

4.4.1 Original implementation

The original code starts by initialising all variables and timers as configured by the RRC and default values. The core functions are *DoReceivePdu*, which handles an RLC PDU received from the MAC, *DoNotifyTxOpportunity* which is called by the MAC to signal an opportunity and *ReassembleAndDeliver* that delivers buffered RLC SDUs from the receiver window. PDCP PDUs (RLC SDUs) received from the PDCP are put into the transmission buffer. When the MAC calls *DoNotifyTxOpportunity* that signals a sufficiently large transmission opportunity, this function takes RLC SDUs from the transmission buffer and packages them into an RLC PDU fitting of the opportunity size and transmits it to the MAC. *DoReceivePdu* takes received RLC PDUs and implements ARQ functionality. Both of these functions differentiate between control PDUs and data PDUs, each with their

own procedure. For each data RLC PDU received with the expected SN, *ReassembleAndDeliver* takes each completed RLC PDU in the window with a SN equal to or higher than expected one (*m_vrR*) and extracts complete RLC SDUs. RLC SDU segments are temporarily kept until its match is found, and discarded if not. This is done by keeping the reordering states shown in 4.1.

Listing 4.1: *lte-rlc-am.h* reordering states

```
SequenceNumber10 m_vrH;                    ///< VR(H)

/**
 * Counters. See section 7.1 in TS 36.322
 */
uint32_t m_pduWithoutPoll; ///< PDU without poll
uint32_t m_byteWithoutPoll; ///< byte without poll

/**
```

For example When a RLC PDU contains an RLC SDU segment, it stores the segments in *m_keepS0* and sets the state to *WAITING_SI_SF*. For the next RLC PDU, if in sequence, will then concatenate its starting segment to the stores one and deliver it as a complete RLC SDU, resetting the reassembling state. If it is out of sequence, the kept segment is discarded and state reset.

RLC PDUs are stored in a data structure containing the sequence number, byte segments and Boolean variable for whether the RLC PDU is complete or not, given that a RLC PDU may be sent as multiple segments if transmission opportunities are small.

4.4.2 Modification

Rewinding back to the design and motivation, this is where out-of-order IP packets are delayed because out-of-sequence RLC PDUs are buffered in the window waiting for a reordering timeout or another packet to move the reception window. In order to forward out-of-sequence RLC PDUs, a few additions are added to *DoReceivePdu*. In the original implementation, when receiving a RLC PDU, the function first checks if the packet is a duplicate or if its within the receiving window or not, in which it is discarded, and whether the packet is complete or not. For a new RLC PDU that has passed these tests, *DoReceivePdu* checks if the sequence number is the expected one (*m_vrR*), and only if so does it initiates the procedure of delivering the buffered packets. For this immediate forwarding scheme, another test case is added for which the sequence number is higher than *m_vrR*, meaning that at least this one packet is received out of order and forwarding should take place. For every such out-of-order packet received, the forwarding function *Forward ()* is called, which iterates the window and forwards any complete RLC SDUs it can find and re-assembling segments if possible.

The forwarding function is described in detail later. In order to maintain the core structure of the original code, RLC PDUs that have had all their payload forwarded are not deleted from the window. When reordering times out or the expected packet arrives, the RLC moves the windows incrementally delivering packets in order, but to avoid having the RLC try to deliver a packet that has already been forwarded, and additional variable is added to the RLC PDU data structure, *m_delivered*, marking it as forwarded.

It is noted that this may not necessarily be the most optimal approach in regards to performance.

Listing 4.2: *lte-rlc-am.h* RLC PDU data structure

```
 *
 * \param packet the packet
 */
void ReassembleAndDeliver (Ptr<Packet> packet);

bool Forward (SequenceNumber10 seqNumber);

/**
```

PDUs marked as delivered do not invoke the delivery function *ReassembleAndDeliver*, as it has already been processed with *Forward*. If *Forward* has modified the RLC PDU by forwarding complete RLC SDUs but been unable to re-assemble any remaining segments, it will have modified the packet payload by removing the forwarded RLC SDUs, and only the segment will remain in the packet payload under the normal delivery procedure.

Listing 4.3: *lte-rlc-am.cc* expected SN and out-of-order forwarding

```
        }

      // - if x = VR(R):
      //     - if all byte segments of the AMD PDU with SN = VR
          (R) are received:
      //         - update VR(R) to the SN of the first AMD PDU
          with SN > current VR(R) for which not all byte
          segments have been received;
      //         - update VR(MR) to the updated VR(R) +
          AM_Window_Size;
      //     - reassemble RLC SDUs from any byte segments of
          AMD PDUs with SN that falls outside of the receiving
          window and in-sequence byte segments of the AMD PDU
          with SN = VR(R), remove RLC headers when doing so and
           deliver the reassembled RLC SDUs to upper layer in
          sequence if not delivered before;

      if ( seqNumber == m_vrR )
        {
          std::map <uint16_t, PduBuffer>::iterator it =
              m_rxonBuffer.find (seqNumber.GetValue ());
          if ( it != m_rxonBuffer.end () &&
               it->second.m_pduComplete )
            {
              it = m_rxonBuffer.find (m_vrR.GetValue ());
              int firstVrR = m_vrR.GetValue ();
              while ( it != m_rxonBuffer.end () &&
                      it->second.m_pduComplete )
                {
                  if ( !it->second.m_delivered )
                    {
                      NS_LOG_LOGIC ("Reassemble and Deliver ( "
                          SN = " << m_vrR << " )");
                      NS_ASSERT_MSG (it->second.m_byteSegments.
                          size () == 1,
                                     "Too many segments . PDU
                                         Reassembly process
                                         didn't work");
                      ReassembleAndDeliver (it->second.
                          m_byteSegments.front ());
                    }

                  m_rxonBuffer.erase (m_vrR.GetValue ());

                  m_vrR++;
                  m_vrR.SetModulusBase (m_vrR);
                  m_vrX.SetModulusBase (m_vrR);
                  m_vrMs.SetModulusBase (m_vrR);
                  m_vrH.SetModulusBase (m_vrR);
                  it = m_rxonBuffer.find (m_vrR.GetValue ());

                  NS_ASSERT_MSG (firstVrR != m_vrR.GetValue (),
                       "Infinite loop in RxonBuffer");
                }
```

```
                NS_LOG_LOGIC ("New␣VR(R)␣␣=␣" << m_vrR);
                m_vrMr = m_vrR + m_windowSize;

                NS_LOG_LOGIC ("New␣VR(MR)␣=␣" << m_vrMr);
              }

          }
```

4.4.3 Forward ()

The *Forward* function is a modified version of the *ReassembleAndDeliver* function. The original *ReassembleAndDeliver* function takes an RLC PDU as argument and strips it of its RLC header. The header information is used to correctly extract the RLC SDU segment(s) in the payload, putting RLC SDUs and segments into a buffer. The framing information from the RLC headers indicates whether the the payload contains segments that are either the first part or the last part of a complete RLC SDU. If the payload contains the first and last bytes of an already complete RLC SDU. *ReassembleAndDeliver* has to account for a large set of cases, from the framing information to the re-assembling status and whether or not the RLC PDU is in order. For example, if the previous call to the function found an incomplete segment which was kept for the next call to find its missing piece, the next function call might contain such a segment but if the RLC PDU was out-of-order these two segments cannot be concatenated because they do not belong to the same RLC SDU, in which the kept segment has to be discarded and re-assembly status updated. For each complete RLC SDU, it is delivered directly to the PDCP using the Service Access Point (SAP), namely $ReceivePdcpPdu(Ptr < Packet > p)$.

Forward() is a recursive function. Instead of being a void function call that only processes one RLC PDU, it transverses all the completed RLC PDUs in the reception window each time it is called. To indicate the base SN at which to start iterating the window, the function takes this sequence number as an argument, which will be the expected SN (*m_vrR*) plus one. Sequence slots in the window that are incomplete or marked as delivered are simply skipped. The idea behind recursively iterating the out-of-order RLC PDUs is for segment re-assembly, since a newly added RLC PDU might contain segments that complete that of previously processes RLC PDUs which had to keep those segments. Recursion also reduces some complexity in regard to state variables. The recursion ends when there are no more complete RLC PDUs to transverse in the reception buffer.

Instead of decisively removing the RLC header from the packet, *Forward* merely peeks at the information. This is because it is not given that the packet is to be delivered or not, as it may contain segments that do not have a complementary match in the reception window. Therefore a copy is made of the original packet, and a copy is processed. If the copy successfully forwards all RLC SDUs, the original PDU is marked as delivered, and if no changes took place, it is left intact in its original state. If some RLC SDUs were forwarded but there are remaining segments, a new packet is made by adding an updated RLC header to the remaining segments, and the original packet is replaced by the updated one.

Listing 4.4: Forward recursion

```
    {
      NS_LOG_LOGIC (seqNumber << " is OUTSIDE the receiving
          window");
      return false;
    }
}

bool
LteRlcAm::Forward (SequenceNumber10 seqNumber)
{
  Ptr<Packet> packet = 0;

  for ( uint16_t i = seqNumber.GetValue (); i < m_vrH.GetValue
      (); i++)
    {
      std::map <uint16_t, PduBuffer>::iterator it =
          m_rxonBuffer.find (i);
      if ( it != m_rxonBuffer.end () && it->second.
          m_pduComplete &&
           !it->second.m_delivered )
        {
          if ( i == seqNumber.GetValue () )
            {
              packet = it->second.m_byteSegments.front ();
            }
          else
            {
              keepS = 0;
              Forward ( SequenceNumber10 (i) );
            }
          break;
        }
    }
```

The function returns a Boolean value to signal the calling function whether the kept segment was delivered or not. This is demonstrated in Listing 4.5 which is the case for a RLC PDU that has no last byte, meaning it contains an incomplete segment at the end. First it iterates the SDU buffer to forward any completed SDUs extracted. The "tail" segment is temporarily stored and copied to the shared re-assembly pointer *keepS*. However the function does not know if there are any in-order RLC PDU(s) received that can complete the remaining part of this tail segment. Therefore it starts the next recursive iteration at the desired sequence number that can complete the segment. If the function returns true, it meas the tail segment was successfully re-assembled and delivered by the next iteration, and the packet can be marked as delivered. If it returns false, either no such RLC PDU has been received or it was unable to re-assemble the RLC SDU, in which case the tail segment much be kept. If in addition to keeping the segment the *counter* variable (which counts delivered RLC SDUs) is zero, no changes were made and the function should abort.

```
            Listing 4.5:  RLC SDU re-assembly case
          NS_LOG_DEBUG ("(11) Forward SN(" << seqNumber.
              GetValue () << ") Time: " << Simulator::Now ().
              GetSeconds());
          m_rlcSapUser->ReceivePdcpPdu (*itp);
          counter++;
        }

      it->second.m_delivered = true;
      Forward ( seqNumber + 1 );
      return retv;
    case (LteRlcAmHeader::FIRST_BYTE | LteRlcAmHeader::
        NO_LAST_BYTE):
      while ( sdusBuffer.size () > 1 )
        {
          NS_LOG_DEBUG ("(10) Forward SN(" << seqNumber.
              GetValue () << ") Time: " << Simulator::Now ().
              GetSeconds());
          m_rlcSapUser->ReceivePdcpPdu (sdusBuffer.front ());
          sdusBuffer.pop_front ();
          counter++;
        }
      NS_LOG_DEBUG("Tail (1)");
      tailSegment = sdusBuffer.front ();
      sdusBuffer.pop_front ();
      keepS = tailSegment->Copy ();

      if ( Forward ( seqNumber + 1 ) )
        {
          it->second.m_delivered = true;
          return true;
        }
      else
        {
          keepS = 0;
```

Function calls that reach the very end of the function and has not return either true or false, are cases where the packet is modified and must be updated. At this point, any completed RLC SDUs have been forwarded, and the only potentially remaining segments are the head segment and the tail segment. A new RLC header is constructed in regard to whether there are any head and/or tail segments, while also updating head information such as length indicators. The new packet, the segment(s) and the new RLC header, replaces the original packet pointer.

4.5 TCP

The LibOS library operating system contains most of the up-to-date networking code for TCP like the Cubic congestion control algorithm. Although it also supports RACK, the implementation of RACK in LibOS is a simplified version of the algorithm which does not fulfil the entirety of the

[8] draft (draft 03). The current version is only used after recovery starts, and does not detect the first loss.

In order to make use of the full power of RACK, its implementations has to be updated. The current RACK implementation is replaced with the RACK implementation from the Linux kernel library, which is a complete implementation of the first RACK draft [33] (draft 01). In order to incorporate this version to LibOS, various TCP kernel networking files have to be updated as well such as header files or acknowledgement procedures that this RACK implementation depends on.

4.5.1 LibOS Modifications

The following directory tree lists the LibOS kernel files that were modified to support the updated RACK version:

```
include
├── linux
│   └── tcp.h
└── net
    ├── inet_connection_sock.h
    └── tcp.h
net
└── ipv4
    ├── tcp_cubic.c .3 tcp_input.c
    └── tcp_recovery.c
```

linux/tcp.h contains the RACK data structure with information of the most recently (s)acked Socket Buffer (SKB) that handles sent or received packets. Associated RTT with microsecond granularity and ending TCP sequence of the SKB is added to the data structure. In *inet_connection_sock.h* a reordering timeout is defined and added to the function logic when resetting the re-transmission timer. *net/tcp.h* replaces old flags and function definitions from the old implementation to the new one. Also a rate sample data structure is added for various timestamp and counter variables. *tcp_recovery.c* is the RACK source code file and is entirely replaced with the new version. *tcp_input.c* is a core file for the Linux TCP/IP protocol suite. This file contains many additions from the newer LKL file to the older one. These additions include RACK specific functions and calls to these functions, integrating RACK into the overall logic and procedures of TCP/IP. Some data structures and variables have been updated or simplified in the newer LKL versions but remain active in the LibOS version, therefore some tweak have been made in terms of pointers and casting the correct variable types to make sure the new RACK implementation is compatible with the older code. An example would be the acknowledgement timestamp in the SKB structure which was previously a pointer but in new version just a variable, so the same data is

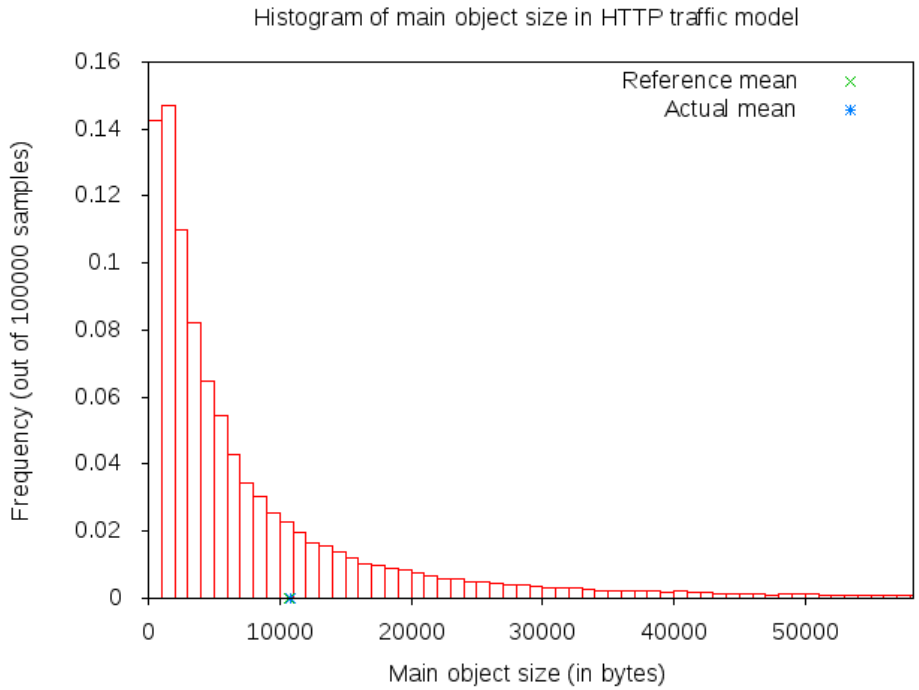

Figure 4.5: Main object size, from [2]

flowing where it should but the correct formats are integrated.

4.6 3GPP HTTP Applications

Often used for network simulations in ns-3 are applications like *BulkSendApplication* or *OnOffApplication* which send data over a variable period of time. These are suitable for simulating long flows for FTP-like traffic.

In order to simulate web traffic more realistically, the *ThreeGppHttpServer/Client* applications simulates web browsing traffic using the Hypertext Transfer Protocol (HTTP). The client models a web browser, requesting web pages from the corresponding server. Responding to client requests, the server will send back either a *main object*(web page) or and *embedded object*(media references by the web page, such as images).

4.6.1 Server

The server application accepts client requests, and keeps every connection open until the client disconnects. The size of the objects sent to the client are randomly determined (see 4.5) and an object may be split into multiple packets due to network limitations. The MTU values used are either 536(low) or 1460(high).

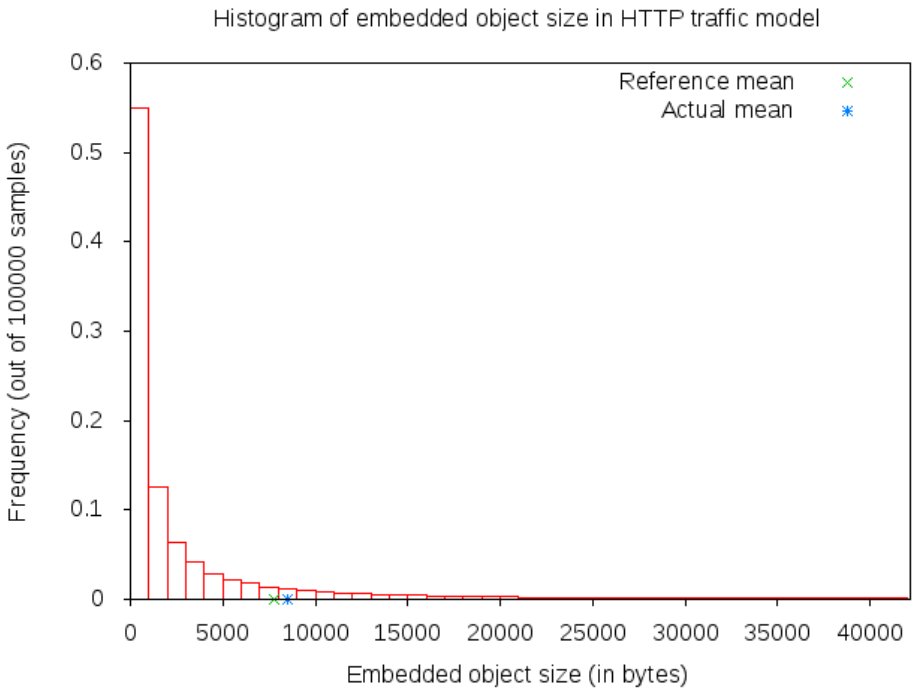

Figure 4.6: Embedded object size, from [2]

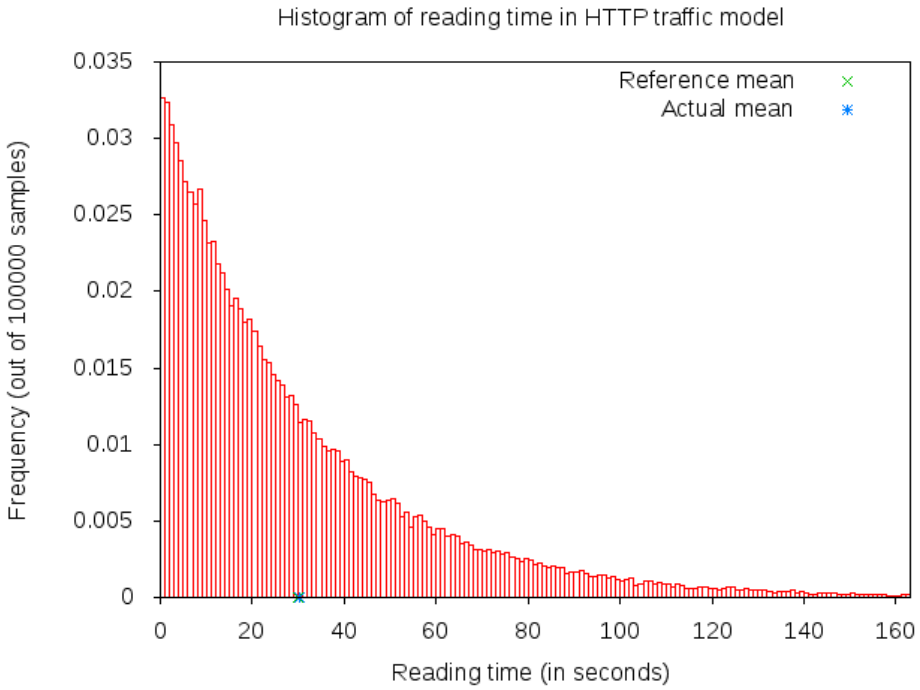

Figure 4.7: Reading time, from [2]

4.6.2 Client

The client application open a connection to the server, and immediately requests a *main object*. Once received, the *main object* is parsed to determine which *embedded objects* to request. This parsing time of the *main object* averages 0.1 seconds, and each request has a constant size of 350 bytes. Once an *embedded object* is determined, it is requested from the server. The next *embedded object* determined will not be requested until the first one is completely received.

Once all objects are received, the client simulates reading time, where no network traffic occurs. Once the reading time expires, another web page is requested. Reading time is randomly determined but is a very long delay, show in 4.7.

Chapter 5

Results

5.1 Experiment setup

The following experiments simulates an UE device communicating with a remote server over an LTE network, with multiple applications on the UE device initiating TCP flows. On the radio link, RLC Acknowledged Mode is used for the entire duration. Results using standard reordering on the RLC ARQ will then be compared to the results using the unordered forwarding implementation. Experiments are run with propagation and fading error models that are partly based on distance, allowing errors to be induced to increasing the UEs distance to the eNB.

Good and poor signal conditions are here defined by how the signal conditions affect the RLC ARQ performance. For *good* signal conditions there is little to no buffering on the RLC ARQ. For *poor* signal conditions there is buffering on the RLC ARQ that allows the performance of the immediate forwarding implementation to be exposed.

5.2 Short dynamic flows

Using the 3GPP HTTP Client/Server applications, two clients are initiated on the UE and starts to request Main/Embedded objects from the server at the same time. The size of the objects are compared to the completion time (delay from request to successful reception).

5.2.1 Good signal conditions

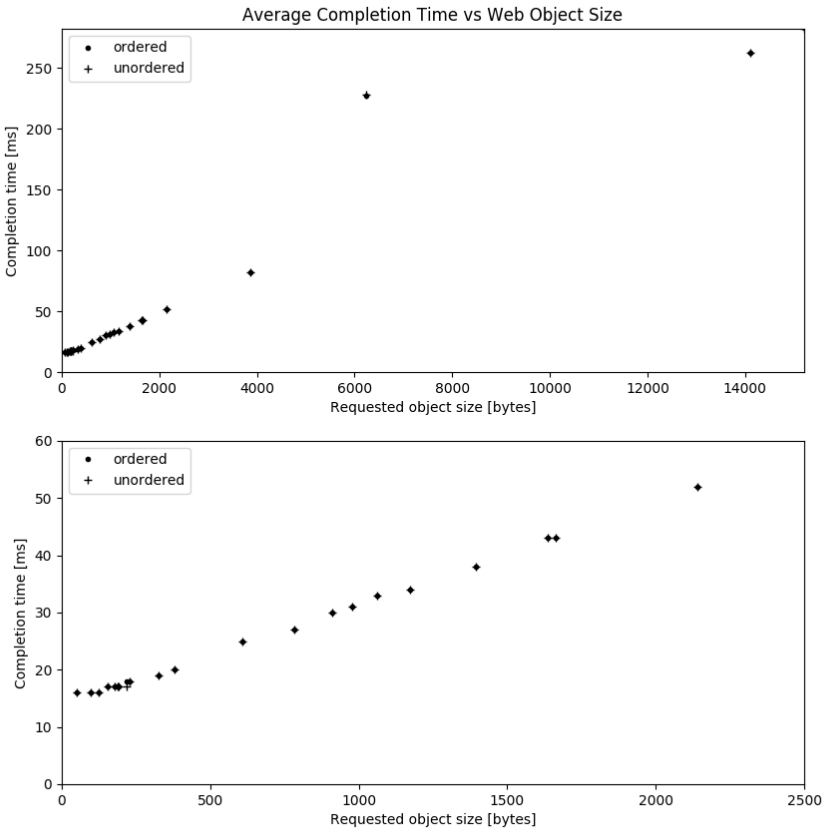

Figure 5.1: Average completion time of Web objects in good signal conditions.

5.2.2 Poor signal conditions

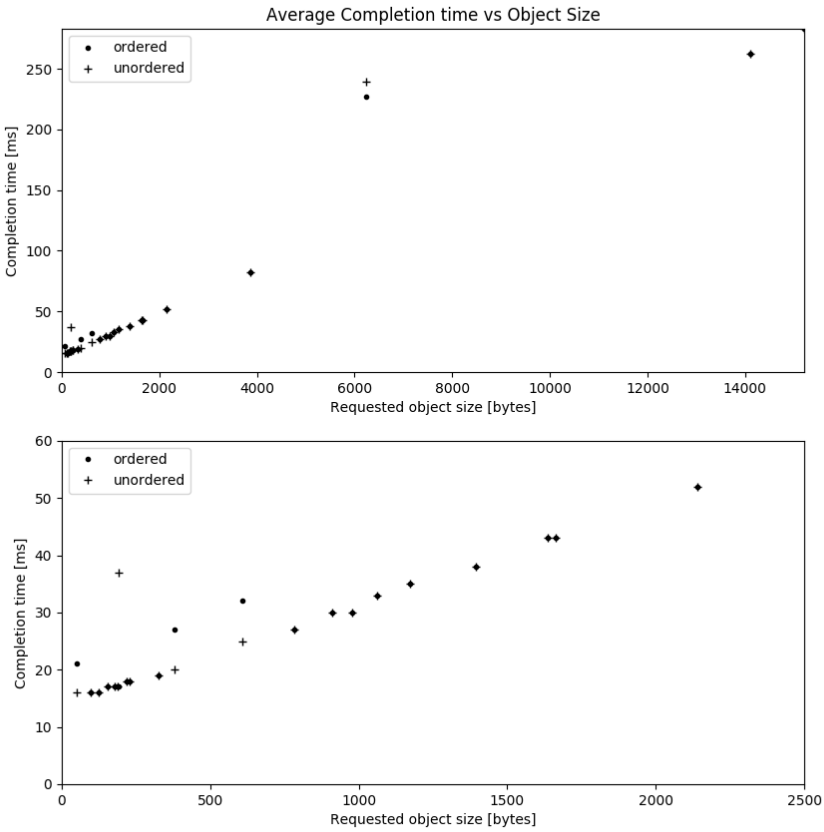

Figure 5.2: Average completion time of Web objects in poor signal conditions.

5.3 Long flows

Three packet sink applications (A, B and C) on the same UE starts receiving constant data from a server to simulate FTP-like traffic.

5.3.1 Poor signal conditions

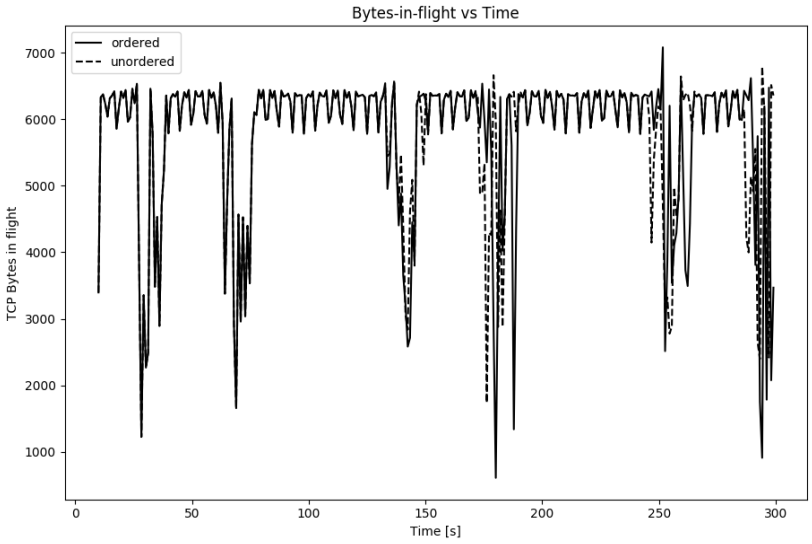

Figure 5.3: Output of long flows in poor signal conditions.

	Ordered	**Unordered**
Application A	0.499542	0.487026
Application B	0.553513	0.555012
Application C	0.470535	0.487026

Table 5.1: Total bytes received for each application.

5.4 Evaluation

For short dynamic flows, unordered forwarding displayed no significant difference from ordered buffering. This was expected since there was little reordering to be done on the RLC ARQ as retransmissions were rarely required and quickly resolved in good signal conditions. In poor signal conditions however, shown in Figure 5.2, there were minor differences between the two schemes. Although being mostly equal, unordered forwarding shows to reduce delay of the completion time for some of the smaller object sizes. However, in the cases where unordered forwarding fails to reduce delay, it shows more delay.

Without more extensive experiments, it is not safe to assume any consistent effect, but these results can hint to the risk factor of immediate forwarding. If TCP packets arrive out-of-order at the receiver, and the RLC ARQ manages to repair the error quickly, there can be improvements in the average completion time of requested objects by reducing HOL blocking. If however the lost TCP packet does not arrive in time, RACK will deem the packet lost and initiate a costly retransmission on the transport layer. In that case, if the packets had not been delivered to TCP unordered had been buffered in the RLC ARQ, the induced delay of buffering might still be smaller than a TCP retransmission.

Even though unordered forwarding was not aimed at improving long flows, the experiments shown in Figure 5.3 was included to see if it had any significant effect on their performance. Even in poor signal conditions, performance was not significantly different between ordered and unordered forwarding. In fact, unordered forwarding seems to have distributed the received bytes more equally across the three applications.

5.5 Limitations

These results are very limited, and they hints more to trends than they do properly quantify the effective results. The main reason for this is the currently limited ability of this simulation setup to produce effective simulations. One such problem is the LTE Radio protocol stack of the ns-3 simulation failing to handle high traffic loads, even when using the default version without any modifications. There could be various reasons for such unexpected behaviour or general rarities. Like DCE configured nodes communicating with ns-3 configured nodes or changes to the 3GPP HTTP applications to support the DCE Kernel stack, all required for the scope of this project but pushing the limitations of how these components work together. Ideas such as increasing the HARQ retransmission limit or timer tuning left out due to the inability to produce good enough simulations for these cases. Insufficient knowledge or misconfigurations may also be the issue of course, anyhow more robust simulations are needed to produce results to fully quantify the ideas presented in this project.

Chapter 6

Conclusions

- Immediate forwarding on the RLC ARQ can reduce HoL blocking in LTE networks, with the risk of not being able to fill gaps in the TCP receiving window withing the RACK specified reordering time.
- RLC segmentation of PDUs limits the effect of immediate forwarding. When the signal conditions are poor, small segmented RLC PDUs are transmitted which can not be immediately forwarded until they form a complete PDU.
- A high reordering timer on the RLC ARQ (10 ms in ns-3) is very persistent a rarely causes any retransmissions on the transport layer. While it may induce HoL blocking it is still a strong alternative to immediate forwarding since it performs well and is very safe.
- Immediate forwarding with TCP RACK does not significantly improve nor deteriorate the performance of long TCP flows in LTE networks.

Chapter 7

Future Work

- Implement the idea of immediate forwarding for Wi-Fi, another error-prone network environment.
- Test immediate forwarding versus TCP Bottleneck Bandwidth and Round-trip propagation time (BBR) in the simulator, another protocol favoring time over sequence counting.
- Investigate effect of reordering of re-establishment procedures as specified by 3GPP.
- Investigate effect of a moving UE device, as opposed to stationary.
- Investigate ways to separate RACK-compliant flows within an LTE network using L4S.

Project code: https://github.uio.no/magnuvau/Masters

Bibliography

[1] Direct code execution, 2018. https://www.nsnam.org/overview/projects/direct-code-execution.

[2] ns-3 home page, 2018. https://www.nsnam.org.

[3] 3rd Generation Partnership Project. Medium access control (mac) protocol specification (3gpp ts 36.321 version 8.6.0). Technical report, July 2009.

[4] 3rd Generation Partnership Project. Packet data convergence protocol (pdcp) specification (3gpp ts 36.323 version 8.6.0). Technical report, July 2009.

[5] 3rd Generation Partnership Project. Radio link control (rlc) protocol specification (3gpp ts 36.322 version 9.0.0). Technical report, February 2010.

[6] 3rd Generation Partnership Project. User equipment (ue) radio transmission and reception (3gpp ts 36.101 version 11.2.0). Technical report, November 2012.

[7] M. Alman, V. Paxon, ICSI, and E. Blanton. Tcp congestion control. Technical report, September 2009.

[8] Y. Cheng, N. Cardwell, N. Dukkipati, and P. Jha. Rack: a time-based fast loss detection algorithm for tcp (draft 03). Technical report, Google Inc, March 2018.

[9] Yuchung Cheng, Neal Cardwell, and Nandita Dukkipati. Rack: a time-based fast loss recovery, 2016. Presentation slides from IETF97: Seoul.

[10] Doublas E. Comer, Peter J. Denning, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a dicipline. Technical report, January 1989.

[11] Christopher Cox. *An introduction to LTE: LTE, LTE-advanced, SAE and 4G mobile communications*. John Wiley & Sons, 2012.

[12] N. Dukkipati, N. Cardwell, Y. Cheng, and M. Mathis. Tail loss probe (tlp): An algorithm for fast recovery of tail losses. Technical report, Google Inc., February 2013.

[13] Ericsson. Traffic and market data report. Technical report, November 2011.

[14] L. Wood G. Fairhurst. Advice to link designers on link automatic repeat request (arq). Technical report, August 2002.

[15] I. Johansson. Congestion control for 4g and 5g access. Technical report, Ericsson AB, October 2015.

[16] Mohammad T. Kawser, Nafiz Imtiaz Bin Hamid, Md. Nayeemul Hasan, M. Shah Alam, and M. Musfiqur Rahman. Limiting harq retransmissions in downlink for poor radio link in lte. Technical report, September 2012.

[17] Sachin K.R, Nikhil Srinath Betgov, Nisarga C, Apeksha S.K, and Smt. Usha M.S. A review of hybrid arq in 4g lte, 2015.

[18] Ke Liu and Jack Y. B. Lee. On improving tcp performance over mobile data networks. Technical report, November 2015.

[19] M. Mathis, J. Mahdavi, S. Floyd, ACIRI, J. Mahdavi, Novell, Pittsburgh Supercomputing Center, M. Podolsky, and UC Berkeley. An extension to the selective acknowledgement (sack) option for tcp. Technical report, July 2000.

[20] M. Mathis, J. Mahdavi, PSC, S. Floyd, LBNL, and A. Romanow. Tcp selective acknowledgment options. Technical report, Sun Microsystems, October 1996.

[21] Matthew Mathis and Jamshid Mahdavi. Forward acknowledgment: Refining tcp congestion control. Technical report, Pittsburgh Supercomputing Center.

[22] Magdalena Nohrborg. Lte overview, 2018. http://www.3gpp.org/technologies/keywords-acronyms/98-lte.

[23] University Carlos III of Madrid, Simula Research Laboratory, Nokia, Orange, LiveU, Software Radio Systems, IMDEA Networks Institute, University of Oslo, and Court of the University of Aberdeen. Xlr8: Accelerating beyond 5g. Technical report.

[24] Colin Perkins. *RTP: Audio and Video for the Internet*. Pearson Education and Addison-Wesley Professional, 2003.

[25] Iulian Rosu. Understanding noise figure. http://www.qsl.net/va3iul.

[26] J. H. Saltzer, D. P. Reed, and D. Clark. End-to-end arguments in system design. Technical report, 1981.

[27] K. Sandlund, G. Pelletier, Ericsson, and L-E. Jonsson. The robust header compression (rohc) framework. Technical report, March 2010.

[28] Woo Cheol Shin, Jin Kyung Park, Jun Ha, and Cheon Won Choi. Occupancy regulation for re-ordering buffer at 3gpp's arq. Technical report, 2004.

[29] Jeremy Stretch. Tcp selective acknowledgments (sack), 2010. http://packetlife.net/blog/2010/jun/17/tcp-selective-acknowledgments-sack/.

[30] Hajime Tazaki. dead or alive: Linux libos project in 2016, 2016. https://github.com/thehajime/blog/issues/1.

[31] tutorialspoint.com. Lte layers data flow, 2018. https://www.tutorialspoint.com/lte/lte_layers_data_flow.htm.

[32] Michael J. Karels Van Jacobson. Congestion avoidance and control. Technical report, 1988.

[33] N. Dukkipati Y. Cheng, N. Cardwell. Rack: a time-based fast loss detection algorithm for tcp (draft 01). Technical report, Google Inc., March 2017.

[34] SeungJune Yi, SeungDuck Chun YoungDae Lee, SungJun Park, and SungHoon Jung. *Radio Protocols for LTE and LTE-Advanced*. John Wiley & Sons, 2012.

[35] Jim Zyren. Overview of the 3gpp long term evolution physical layer. Technical report, NXP, July 2007.

Glossary

3G 3rd Generation. 14, 15, 37

3GPP Third Generation Partnership Project. 14, 15, 26, 46, 53

4G 4th Generation. 14

ACK Acknowledgement. 10, 11, 38

AM Acknowledged Mode. 24, 25, 48, 49

AQM Active Queue Management. 3, 36

ARQ Automatic Repeat Request. 25, 28, 35, 36

BBR Bottleneck Bandwidth and Round-trip propagation time. 73

BCCH Broadcast Control Channel. 20

BLER Code Block Error Rate. 48

CCCH Common Control Channel. 20

CQI Channel Quality Information. 39

CRC Cyclic Redundancy Check. 28

CWND Congestion Window. 10

DCCH Dedcated Control Channel. 20

DCE Direct Code Execution. 50, 51

DL-SCH Downlink Shared Channel. 20

DSACK Duplicate SACK. 12

DTCH Dedicated Traffic Channel. 20

dup-ACK Duplicate Acknowledgement. 11–13, 35, 36

E-UTRAN Evolved Universal Terrestrial Access Network. 17, 18, 26, 42

ECN Excplicit Congestion Notification. 3, 44

eNB Evolved NodeB. 2, 17, 20, 22, 29, 46, 48, 51, 52, 65

EPA Extended Pedestrian A. 47

EPC Evolved Packet Core. 17, 18

ETU Extended Typical Urban model. 47

EVA Extended Vehicular A model. 47

FACK Forward Acknowledgment. 13, 14

FEC Forward Error Correction. 28

FTP File Transfer Protocol. 62

GSM Global System for Mobile Communications. 15

GTP GPRS Tunneling Protocol. 46

HARQ Hybrid-Automatic Repeat Request. 24, 27–29, 36, 48

HOL Head-of-line blocking. 4, 41, 69

HTTP Hypertext Transfer Protocol. 62

IP Internet Protocol. 9, 19, 46

IPv4 Internet Protocol version 4. 9

IPv6 Internet Protocol version 6. 9, 23

IR Incremental Redundancy. 48

ITU International Telecommunication Union. 14

L4S Low Latency, Low Loss, Scalable Throughput. 37

LTE Long Term Evolution. 1, 3, 14, 15, 17–19, 26, 29, 30, 32, 36–38, 45, 46, 51, 65, 71, 73

MAC Medium Access Control. 1, 22, 26, 48

MCS Modulation Coding Scheme. 48

MME Mobility Management Entity. 18, 46

MT Maximum Throughput. 48

MTU Maximum Transmission Unit. 62

NS-3 Network Simulator version 3. 45

ns-3 Network Simulator 3. 5, 6, 51, 53, 62, 69, 71

OFDM Orthogonal Frequency Division Multiplexing. 29

OSI Open Systems Interconnection. 19

P-GW Packet Data Network (PDN) gateway. 18, 46

PCCH Paging Control Channel. 20

PDCCH Physical Downlink Control Channel. 20

PDCP Packet Data Convergence. 1, 22, 23

PDSCH Physical Downlink Shared Channel. 20

PDU Protocol Data Unit. 22

PHY Physical layer. 1, 29, 48

PUCCH Physical Uplink Control Channel. 20

PUSCH Physical Uplink Shared Channel. 20

QAM Quadrature Amplitude Modulation. 32

QoS Quality of Service. 20, 27, 29

RACK Recent Acknowlegment. 1, 12–15, 35–37

RB Resource Block. 29, 30, 45

RLC Radio Link Control. 1, 22, 23

ROHC RObust Header Compression. 23, 41

RRC Radio Resource Control. 20, 22, 29, 44, 53

RTO Retransmission Timeout. 12, 14, 35

RTT Round Trip Time. 4, 10, 13, 25

S-GW Serving Gateway. 18, 46

SACK Selective Acknowledgement. 1, 11, 12, 26

SAP Service Access Point. 48, 58

SDU Service Data Unit. 22

SINR Signal to Interference and Noise Radio. 33, 48

SKB Socket Buffer. 61

SN Sequence Number. 10, 11, 58

SNR Signal to Noise Ratio. 33

TB Transport Block. 27, 48

TCP Tranmission Control Protocol. 3, 9–13, 19, 35–38

TLP Tail Loss Probe. 14, 15

TM Transmission Mode. 23, 24, 48

UDP User Datagram Protocol. 10, 19, 38, 46

UE User Equipment. 2, 17, 20, 27, 29, 46, 48, 51, 52, 65, 66, 68, 73

UL-SCH Uplink Shared Channel. 20

UM Unacknowledged Mode. 24, 48

UMTS Universal Mobile Telecommunication System. 15

UTRAN Universal Terrestrial Access Network. 26

VoIP Voice over IP. 18

WLAN Wireless Local Area Networks. 30