Radio Link Control and Logical Channel Prioritization in 5G NR

A Project Report

submitted by

ABHIJIT S GUPTA

in partial fulfilment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY AND MASTER OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

MAY 2019

THESIS CERTIFICATE

This is to certify that the project report titled **Radio Link Control and Logical Channel Prioritization in 5G NR**, submitted by **Abhijit S Gupta**, to the Indian Institute of Technology, Madras, for the award of the degree of **Bachelor of Technology and Master of Technology**, is a bona fide record of the research work done by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. Krishna Jagannathan
Project Guide
Associate Professor
Dept. of Electrical Engineering
IIT-Madras, 600 036

Place: Chennai

Date: 10/05/2019

ACKNOWLEDGEMENTS

I would like to take this opportunity express my sincere gratitude to my guide Dr. Krishna Jagannathan for his encouragement and timely guidance. I thank him for the opportunity he has given me to explore cutting-edge telecommunications from a highly practical viewpoint as part of the prestigious 5G testbed project.

I would also like to thank Jaswanthi Mandalapu and the other members of the IITM testbed team for helping me learn new things in such a vast field every day. I would also like to acknowledge the help and knowledge provided by the IITH and CEWiT teams which helped me understand the engineering and implementation aspects of mobile communications.

A special thanks to my friends for adding great fun and making my life in the institute a memory to be cherished for a lifetime. Finally I would like to thank my parents for supporting me in all my endeavors and being there me in the toughest of times and making me into who I am today.

ABSTRACT

KEYWORDS: 5G; Radio Link Control;Logical Channel Prioritization; NR Protocol Stack.

With 4G bringing in enhanced mobile internet services, 5G aims to improve services by several multiples, while hosting features to cater to a wide variety of future services such as the Internet of Things. Innovations and modifications are being done all across the 5G stack to implement the new architecture.

This work discusses the improvements and implementation aspects of Layer 2 in the 5G Protocol Stack that comprises the RRC, SDAP, PDCP, RLC, and MAC layers, all of which are now designed to handle services with a wide range of QoS requirements and at the same time able to be sliced and split to increase the flexibility.

In particular, this work explores and elaborates on the implementation of two modules with regard to contributing to the 5G Testbed Project as well as the 3GPP RAN2 standards, namely Radio Link Control, a sublayer that handles segmentation, reassembly, and retransmission of packets, and Logical Channel Prioritization, a module within the MAC layer that performs intra-UE uplink priority handling across different control and user plane logical channels.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

3GPP	3rd Generation Partnership Project
5GC	5G Core Network
5QI	5G QoS Indicator
AM	Acknowledged Mode
AMF	Access and Mobility Management Function
ARQ	Automatic Repeat reQuest
BCCH	Broadcast Control Channel
BSD	Bucket Size Duration
BSR	Buffer Status Reporting
CCCH	Common Control Channel
CE	Control Element
C-RNTI	Cellular Radio Network Temporary Identifier
DCCH	Dedicated Control Channel
DL	Downlink
DPDK	Data Plane Development Kit
DRB	Data Radio Bearer
DTCH	Dedicated Traffic Channel
eMBB	Enhanced Mobile Broadband
eNB	Evolved NodeB
e-UTRA	Evolved Universal Terrestrial Radio Access
GBR	Guaranteed Bit Rate
gNB	next Generation NodeB
IITM	Indian Institute of Technology, Madras
IMT	International Mobile Telecommunications
IP	Internet Protocol
IoT	Internet of Things
ITU	International Telecommunication Union
LCID	Logical Channel ID

LCP	Logical Channel Prioritization
LTE	Long Term Evolution
MAC	Medium Access Control
MIMO	Multiple Input Multiple Output
MMTC	Massive Machine-Type Communication
NAS	Non-Access Stratum
NR	New Radio
PBR	Prioritized Bit Rate
PCCH	Paging Control Channel
PDB	Packet Delay Budget
PDCP	Packet Data Convergence Protocol
PDU	Protocol Data Unit
PER	Packet Error Rate
PS	Protocol Stack
PUSCH	Physical Uplink Shared Channel
QFI	QoS Flow ID
QoS	Quality of Service
RAN	Radio Access Network
RLC	Radio Link Control
RRC	Radio Resource Control
SCS	Subcarrier Spacing
SDAP	Service Data Adaptation Protocol
SDU	Service Data Unit
SI	Segmentation Info
SMF	Session Management Function
SN	Sequence Number
SO	Segment Offset
SRB	Signaling Radio Bearer
TB	Transport Block
TM	Transparent Mode
TR	Technical Report
TS	Technical Specification
TSN	Time Sensitive Networking

TTI	Transmission Time Interval
UE	User Equipment
UL	Uplink
UM	Unacknowledged Mode
UPF	User Plane Function
URLLC	Ultra-Reliable Low-Latency Communication
VoIP	Voice over Internet Protocol

CHAPTER 1

Introduction to 5G NR

1.1 Evolution of Mobile Communications

Over the last four decades, there have been four generations of mobile communication systems. With 1G introducing mobile telephony to the world in the early 1980s, 4G has focused on enhancing mobile broadband from the foundation established by 3G.

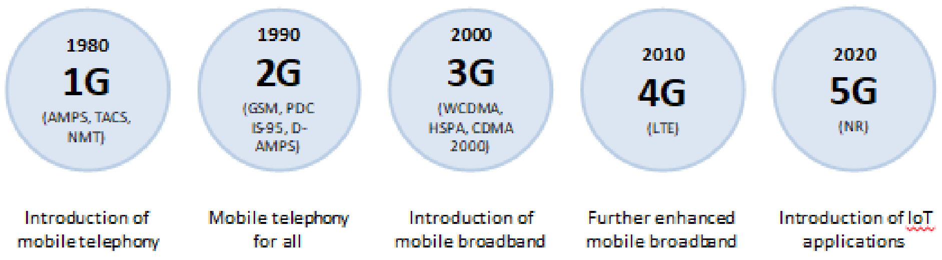

1980
1G
(AMPS, TACS, NMT)
1990
2G
(GSM, PDC IS-95, D-AMPS)
2000
3G
(WCDMA, HSPA, CDMA 2000)
2010
4G
(LTE)
2020
5G
(NR)
Introduction of mobile telephony
Mobile telephony for all
Introduction of mobile broadband
Further enhanced mobile broadband
Introduction of IoT applications

Figure 1.1: Evolution of mobile communication technologies

5G NR (New Radio) is the key Radio Access Technology being developed by 3GPP for the 5G mobile network, which will be designed to meet IMT-2020 requirements put forth by ITU.

Discussions on 5G began in 2012, with the aim of commercializing it in the early 2020s. However, 5G is not restricted to just the specific radio-access technology but rather to a wide range of new services in the future and how technologies throughout the stack should be developed to cater to these requirements.

1.2 Use Cases of 5G

There are three main classes of 5G use cases, namely:

1. **E**nhanced **M**obile **B**road**b**and (***eMBB***): This set of use cases refer to a straightforward enhancement of broadband services from LTE, enabling larger volumes of

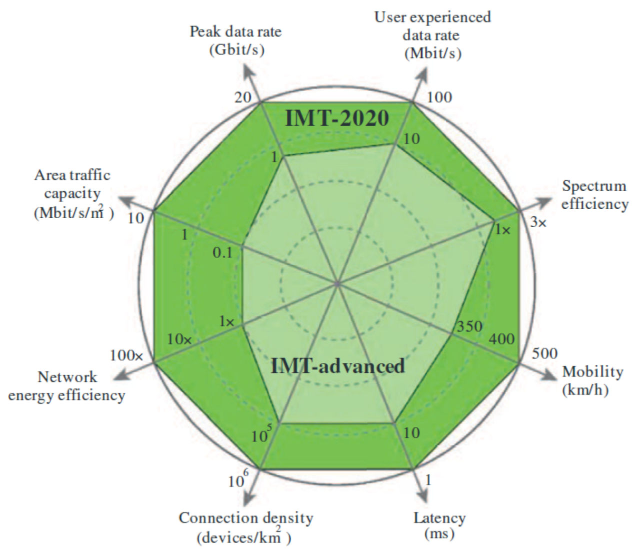


Figure 1.2: IMT 2020 Requirements for 5G relative to IMT-Advanced Requirements for 4G

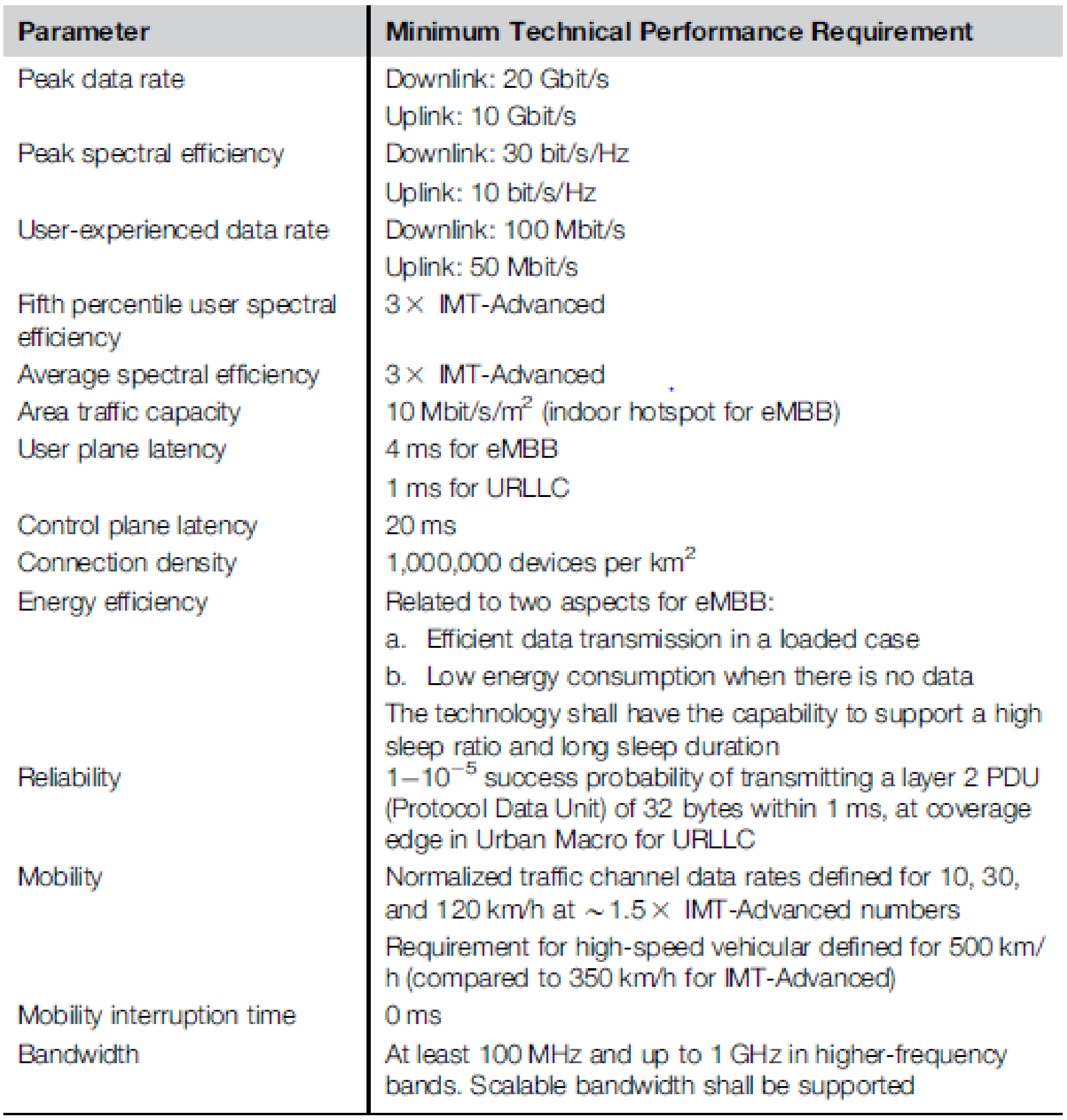
data and enhanced user experience (higher reliability, lower latency, better rates etc). Some use cases enabled by 5G under this umbrella include virtual reality live streaming, 4K real-time gaming etc.

2. **M**assive **M**achine-**t**ype **C**ommunication (***mMTC***): These use cases refer to services by and for a large number of devices, mainly designed for the Internet of Things. This includes remote sensors and actuators to monitor specific machines or environments. The important requirements for these services include low device cost and low power consumption since they are generally left in remote places for a long time. However, high data rates are not required as only small data packets are generated at certain periods.

3. **U**ltra-**r**eliable and **L**ow-**l**atency **C**ommunication (***URLLC***): This caters to mission-critical services where packets need to be sent with very low latencies and high reliabilities. Some examples include Industrial IoT, traffic safety, and vehicle-to-everything (V2X) communication.

1.3 Overall System Architecture

There are two key components in the 5G architecture, namely the Core Network (5GC) and Radio Access Network (RAN). The 5GC handles network functions like authentication, charging, and setting up end-to-end connections, and the RAN handles all radio-related functionalities such as scheduling, radio resource management, transmission and coding protocols, MIMO schemes etc.



Parameter	Minimum Technical Performance Requirement
Peak data rate	Downlink: 20 Gbit/s
	Uplink: 10 Gbit/s
Peak spectral efficiency	Downlink: 30 bit/s/Hz
	Uplink: 10 bit/s/Hz
User-experienced data rate	Downlink: 100 Mbit/s
	Uplink: 50 Mbit/s
Fifth percentile user spectral efficiency	3 × IMT-Advanced
Average spectral efficiency	3 × IMT-Advanced
Area traffic capacity	10 Mbit/s/m^2 (indoor hotspot for eMBB)
User plane latency	4 ms for eMBB
	1 ms for URLLC
Control plane latency	20 ms
Connection density	1,000,000 devices per km^2
Energy efficiency	Related to two aspects for eMBB:
	a. Efficient data transmission in a loaded case
	b. Low energy consumption when there is no data
	The technology shall have the capability to support a high sleep ratio and long sleep duration
Reliability	$1-10^{-5}$ success probability of transmitting a layer 2 PDU (Protocol Data Unit) of 32 bytes within 1 ms, at coverage edge in Urban Macro for URLLC
Mobility	Normalized traffic channel data rates defined for 10, 30, and 120 km/h at ~1.5 × IMT-Advanced numbers
	Requirement for high-speed vehicular defined for 500 km/h (compared to 350 km/h for IMT-Advanced)
Mobility interruption time	0 ms
Bandwidth	At least 100 MHz and up to 1 GHz in higher-frequency bands. Scalable bandwidth shall be supported

Figure 1.3: Detailed Table of IMT-2020 Requirements

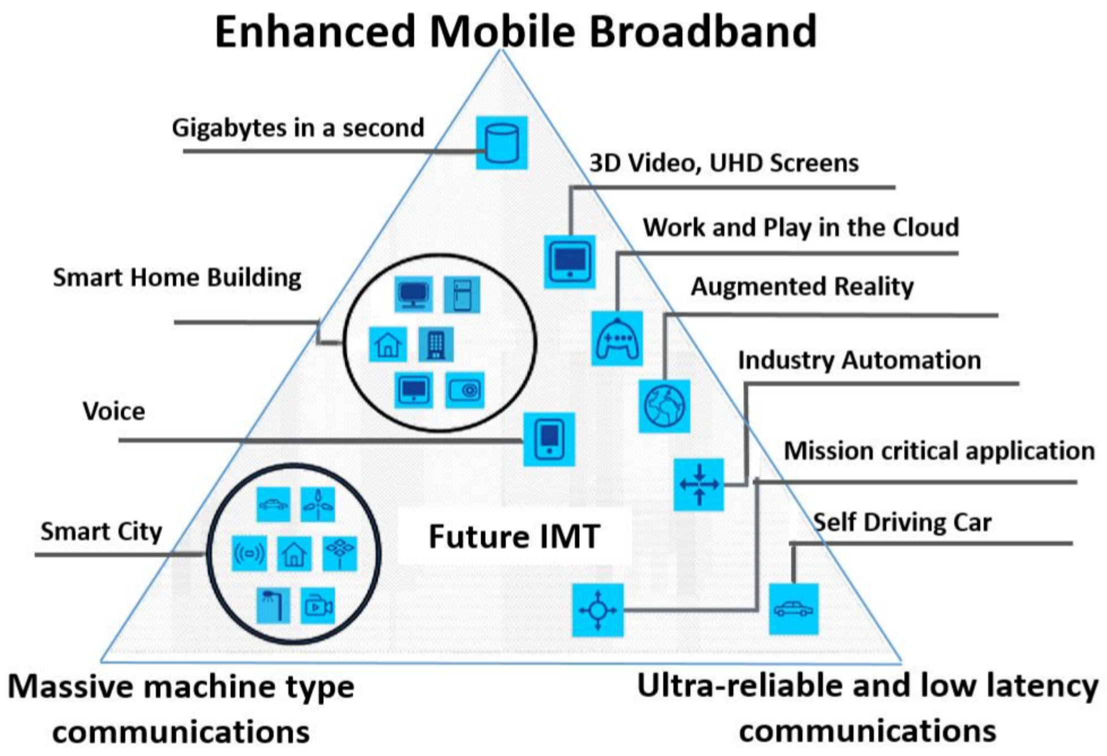

Enhanced Mobile Broadband
Gigabytes in a second
3D Video, UHD Screens
Work and Play in the Cloud
Augmented Reality
Industry Automation
Mission critical application
Self Driving Car
Smart Home Building
Voice
Smart City
Future IMT
Massive machine type communications
Ultra-reliable and low latency communications

Figure 1.4: 5G Use Cases and Applications

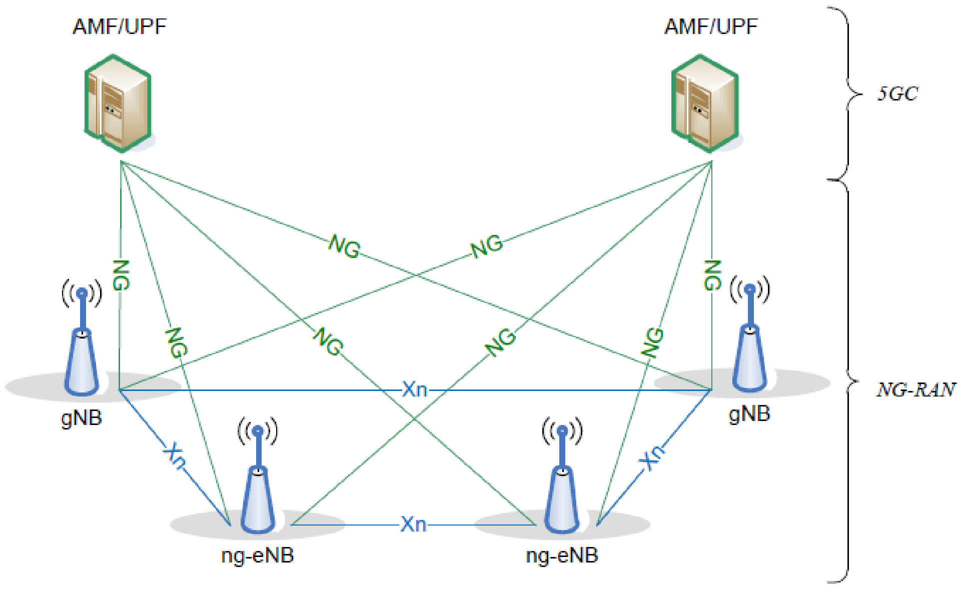


Figure 1.5: 5G Use Cases and Applications (As per TS 38.300)

1.3.1 5G Core Network (5GC)

The 5GC builds upon 4G's EPC (Evolved Packet Core) by introducing a service-based architecture, network slicing, and user/control-plane split, all of which are crucial to the design of the 5G RAN as well.

A service-based architecture is especially important to 5G since it caters to a huge number of use cases, with each having varied demands and Quality of Service (QoS) requirements. This is a natural step forward from 4G since most of the EPC today is already highly virtualized.

Network slicing is often mentioned in the context of 5G innovations. A network slice is a logical subset of the network that is created to cater to a specific user or business need, and this comprises all key functions from the service-based architecture to function in line with those specific needs, be it a higher data rate, lower latency, a guaranteed data rate, or a mission-critical application, These slices run from the application layer to the physical layer. While they all share the same physical layer and radio network, they appear to be separate networks from the point of view of the applications. This implies that the lower layers of the 5G stack should be designed to handle a wide variety of slices simultaneously based on user requirements.

The user/control plane split is done to further modularize the 5G system so that each of them can be independently scaled and altered without affecting the other.

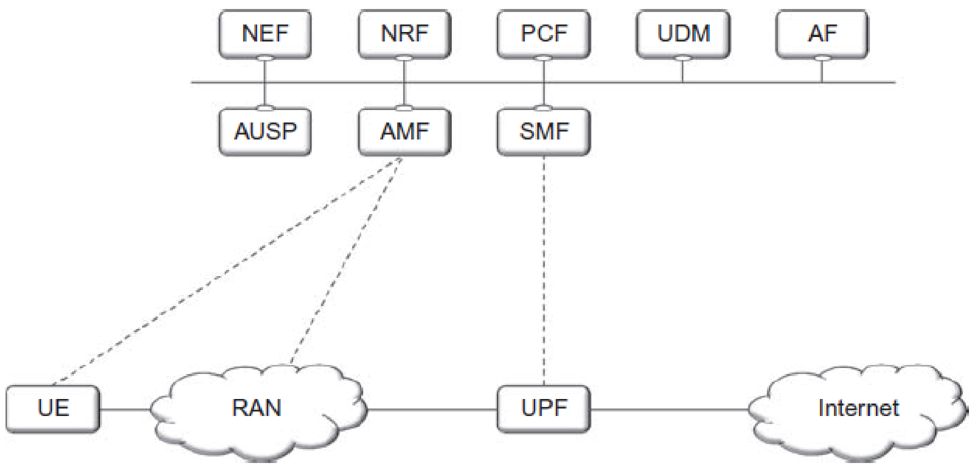


Figure 1.6: High Level 5G Core Network Architecture (As per TS 38.300)

In the above architecture, the User Plane Function (UPF) connects the (R)AN to the Internet. A key differentiator between 5G and 4G is its support for non-radio access networks (WiFi, for instance). The Session Management Function (SMF) handles session-related functionalities like IP Address allocation and policy enforcement. The other 5GC modules handle functions ranging from security and authentication to application management.

1.3.2 5G Radio-Access Network (RAN)

The 5G RAN provides access to the UEs (user equipment) to the 5GC through a gNB (5G NR Base Station) or through an ng-eNB (to serve 4G LTE UEs). The gNBs are connected to the 5GC through various network interfaces, with each serving specific functions keeping in mind the virtualized core network architecture.

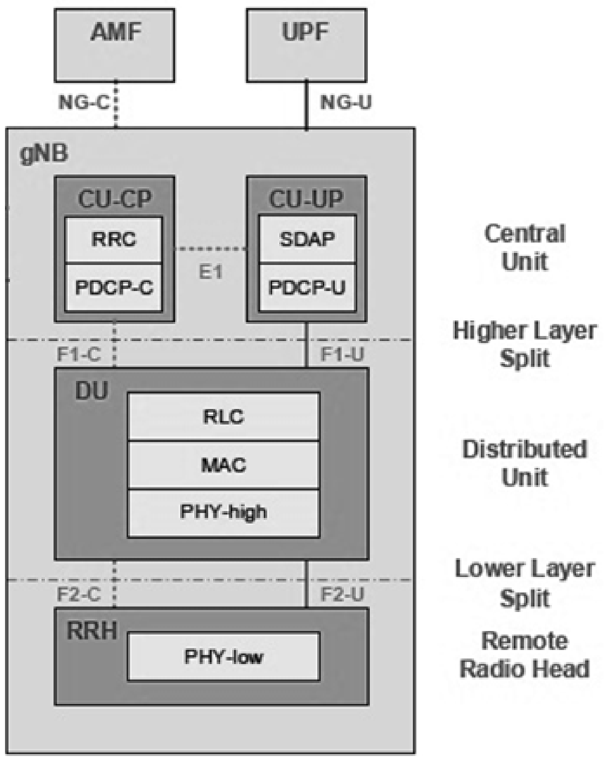


Figure 1.7: NG RAN and all Interfaces

The RAN protocol architecture comprises of several layers, namely the RRC (Radio Resource Control), SDAP (Service Data Application Protocol), PDCP (Packet Data Convergence Protocol), RLC (Radio Link Control), MAC (Medium Access Control), and PHY (Physical Layer), on both the gNB and the UE. All the sublayers of the protocol stack are discussed in detail in the next chapter.

CHAPTER 2

The NR Protocol Stack

As briefly discussed in the previous chapter, the NR protocol stack consists of the RRC, SDAP, PDCP, RLC, MAC, and the PHY layers. Their functionalities are split between the user and control planes, with RRC resting purely on the control plane and SDAP on the user plane.

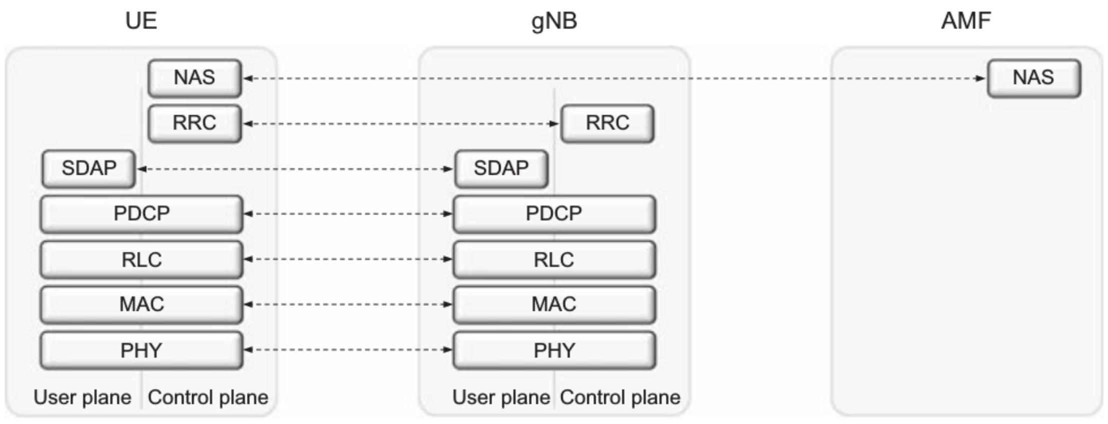


Figure 2.1: Protocol Stack (User and Control Plane) as per TS 38.300

2.1 Key Architectural Elements

2.1.1 5G QoS Model

The core principles of QoS handling in 5G are the same as that in LTE. The network is in charge of the QoS control and only the core network is aware of the service, with it configuring the RAN elements through RRC (Radio Resource Control) to ensure QoS. QoS handling is important in 5G due to its many features to support a wide range of services, network slicing etc.

For each connected UE, there are one or more PDU sessions, each with one or more QoS Flows and Data Radio Bearers. IP packets are mapped to QoS flows that can

Resource Type	**Priority**	**PDB**	**PER**	**Example Services**
GBR	20	100 ms	10^{-2}	Conversational Voice
GBR	30	50 ms	10^{-3}	Real-time Gaming, V2X Messages, Automation
GBR	7	75 ms	10^{-2}	Mission-critical Push To Talk Voice
Non-GBR	80	300 ms	10^{-6}	Buffered Video Streaming, Email, Browsing

Table 2.1: QoS requirements and configurations for some select services

provide services based on their requirements, which include metrics like Guaranteed Bit Rate (GBR), Packet Delay Budget (PDB), Packet Error Rate (PER) etc. Each packet is marked with a QoS Flow ID (QFI) to assist uplink QoS handling. Then, these packets are mapped to different Data Radio Bearers (DRB) in the SDAP layer based on their QFIs. Multiple QoS flows can be mapped to the same DRB.

As described in the table, 3GPP TS 23.501: System Architecture for the 5G System provides an elaborate list of the configurations for various services (in Table 5.7.4-1, 23.501 V16.0.2). It even specifies the PDBs for just the radio interface. For example, 20 ms should be subtracted from the PDB of conversational voice to account for the delay between a UPF terminating N6 and a 5G-AN, which results in an 80ms PDB for the radio interface.

These parameters, particularly the priority, can be also used to configure various RAN modules through the RRC. One such module, Logical Channel Prioritization, is discussed in detail in Chapter 4.

2.1.2 Radio Bearers

Radio bearers carry packets from SDAP to PDCP and subsequent layers. They are of two kinds: Signaling Radio Bearers (SRB) and Data Radio Bearers (DRB). SRBs carry control signaling information. SRB0 carries Common Control Channel (CCCH) information, and SRBs 1-3 carry other control information and RRC messages.

2.1.3 Channels

Channels are analogous to "data pipelines", and are of three main kinds: logical channels, transport channels, and physical channels. Additionally, RLC channels are used to segregate DRB and SRB packets to different RLC modes (which are discussed in detail in Chapter 3).

Logical channels specify what kind of information is being carried, and are fed into MAC. After performing MAC functionalities, they are mapped to transport channels which carry the data to the PHY layer and specify how the data needs to be sent. Once the PHY layer modifies and adds headers, the data is sent to physical channels which specify where the data needs to go once it reaches the receiver.

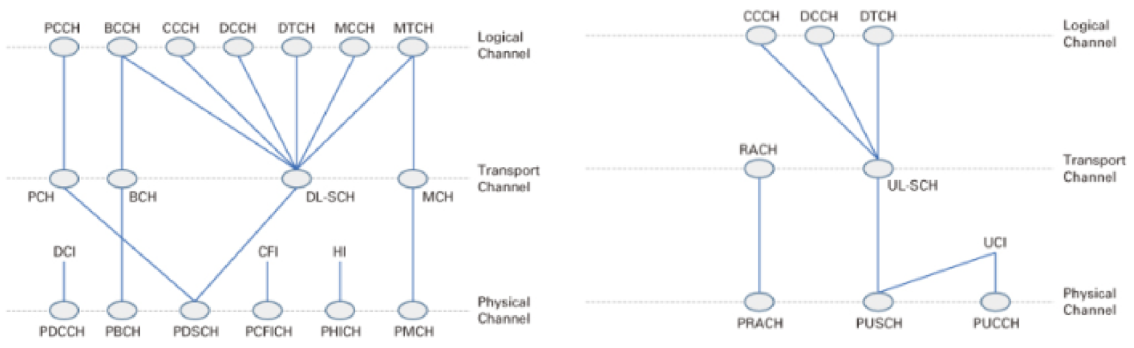
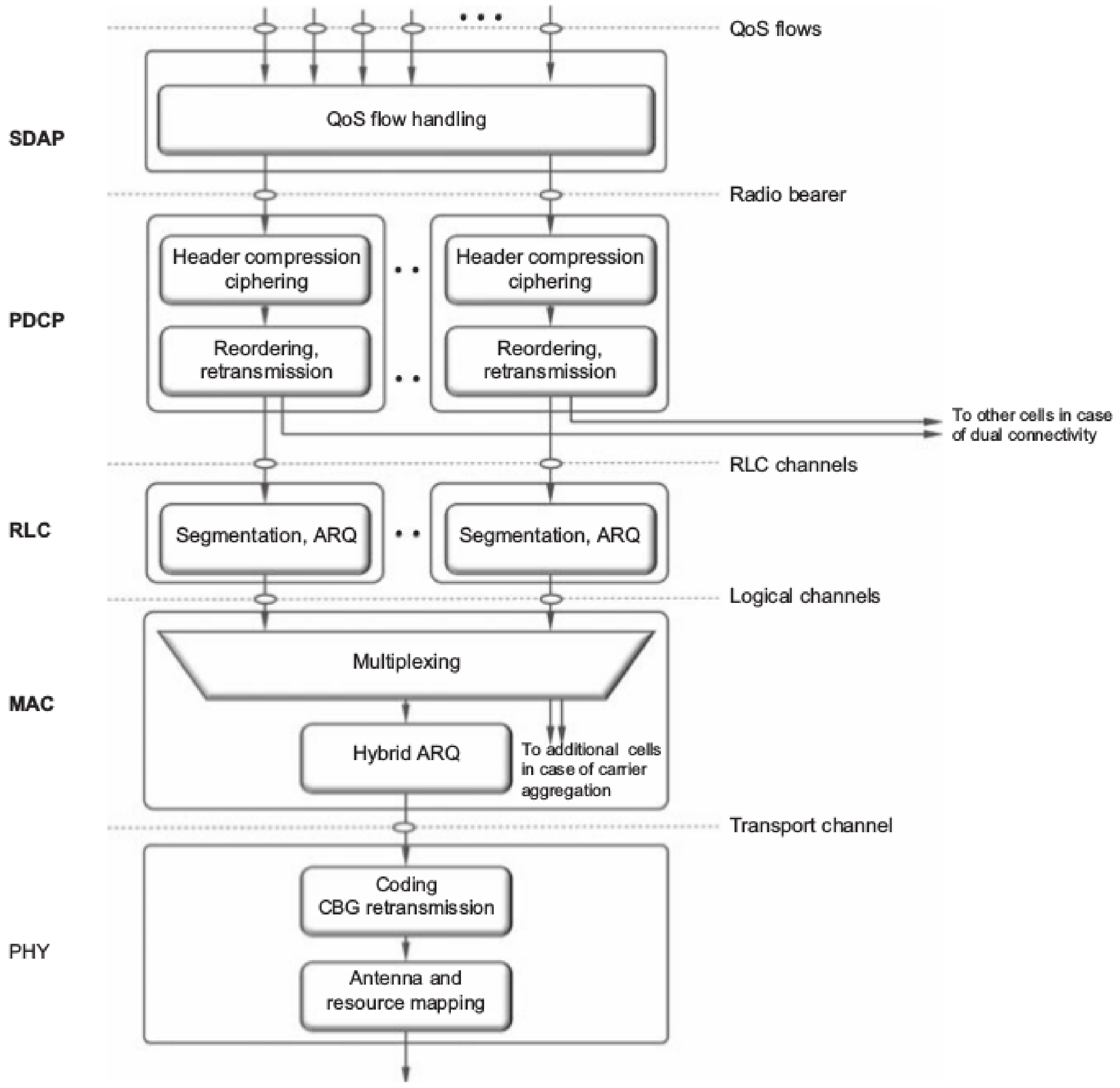


Figure 2.2: Channel Mappings (DL and UL)

2.2 User- and Control-Plane Protocols

Most of the NR user-plane protocols are similar to their LTE counterparts but with certain tweaks in order to handle the different 5G enhancements. For example, the different protocols are designed keeping in mind the QoS flow model as described in the previous chapter. Additionally, the SDAP layer is newly added in NR in order to map the QoS flows to the corresponding radio bearers. The other sub-layer specific enhancements are discussed in the subsequent subsections. The figures below illustrate the protocol stack structure in detail, along with providing details on the data flow.



QoS flows
SDAP
QoS flow handling
Radio bearer
PDCP
Header compression ciphering
Header compression ciphering
Reordering, retransmission
Reordering, retransmission
To other cells in case of dual connectivity
RLC channels
RLC
Segmentation, ARQ
Segmentation, ARQ
Logical channels
MAC
Multiplexing
Hybrid ARQ
To additional cells in case of carrier aggregation
Transport channel
PHY
Coding CBG retransmission
Antenna and resource mapping

Figure 2.3: Protocol Stack (User and Control Plane)

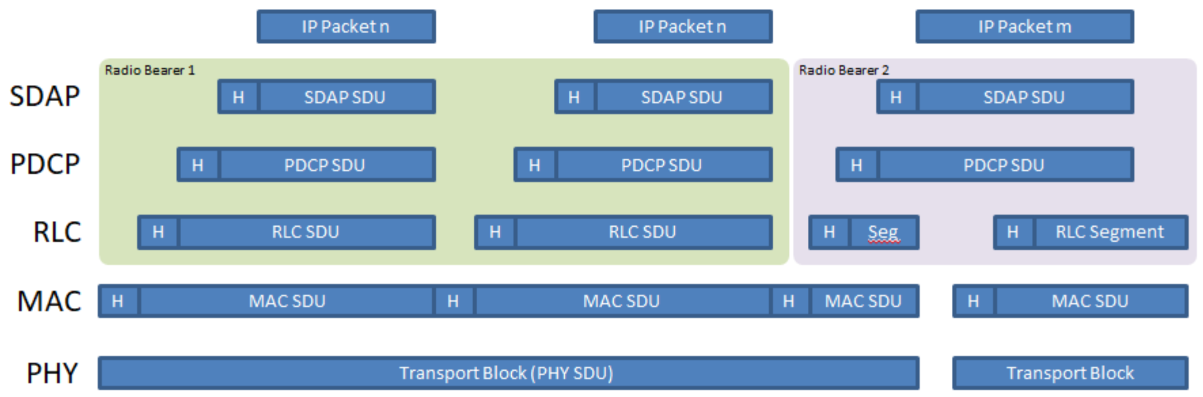


Figure 2.4: Protocol Stack Data Flow

2.2.1 Service Data Adaptation Protocol (SDAP)

The SDAP is a new addition in NR and is responsible for mapping QoS flow bearers from the 5G core network to data and signalling radio bearers (DRBs and SRBs), which are logical pathways through other RAN protocol stack sublayers. This layer is also responsible for marking the QoS Flow Identifier (QFI) for downlink and uplink packets, which in turn is used to identify the 5G QoS Indicator (5QI) that specifies the QoS requirements for the packet.

2.2.2 Packet-Data Convergence Protocol (PDCP)

The PDCP sublayer performs header compression, encryption, duplication and removal, and in-sequence delivery functions.

PDCP's header compression is mainly done to reduce the amount of data flowing through the radio interface, and its implementation is based on the RoHC (Robust Header Compression) framework, which is a set of standardized header compression algorithms also used in several other mobile technologies. PDCP is also responsible for ciphering of user-plane packets and integrity protection of control-plane packets, to ensure that control messages are originating from the correct source. At the receiver side, the PDCP performs the corresponding decompression and decryption algorithms to retrieve the original packets before delivering to the upper layers.

The PDCP also removes duplicate packets and ensures in-sequence delivery (if required), which is especially useful in cases like intra-gNB handover, where both the old and the newly connected gNBs could have some overlap. PDCP also duplicates packets sometimes to improve diversity, and these duplicates will be transmitted over multiple cells. This is particularly useful for services that demand high reliability.

Another use of PDCP is to ensure dual connectivity, where a UE is connected to two cell groups, namely the Master Cell Group (MCG) and the Secondary Cell Group (SCG). The radio bearer can split across the two cell groups and PDCP is responsible to distribute the data to the MCG and the SCG when this happens.

2.2.3 Radio Link Control (RLC)

The RLC protocol is responsible for segmenting and reassembling the PDCP PDUs into suitably sized RLC PDUs. This is done to ensure that data is delivered in appropriately-sized packets to the lower layers. It also handles the retransmission of erroneously received PDUs and the removal of duplicates. Depending on the type of service and data being transferred, RLC can be configured in three modes - Transparent Mode (TM), Unacknowledged Mode (UM), and Acknowledged Mode (AM).

Some major differences compared to LTE is that RLC does not ensure in-sequence delivery (which is done by PDCP in NR) and that concatenation has now been removed, with segmentation being the only modification that RLC does to an SDU apart from header addition. Both of these have been done to reduce the latency of the RLC sublayer.

The RLC sublayer is explained in detail in Chapter 3.

2.2.4 Medium Access Control (MAC)

The MAC layer handles logical channels and performs priority handling, multiplexing, hybrid ARQ and retransmissions, scheduling, discontinuous reception and more.

As described in Section 2.1.3, the MAC layer provides services to the RLC and processes packets coming through the logical channels. The different logical channels are as follows:

- *Broadcast Control Channel* (BCCH): Used for transmission of system information from the network to all UEs in the cell.
- *Paging Control Channel* (PCCH): Used for paging those devices whose location is unknown to the network.
- *Common Control Channel* (CCCH): Used for transmission of control information in conjunction with random access procedures. This is by default SRB 0, and is given the highest priority for transmission.
- *Dedicated Control Channel* (DCCH): Used for transmission of control information for a specific UE.
- *Dedicated Traffic Channel* (DTCH): Used for transmitting user data to/from a UE. This logical channel type is mapped to DRBs which can carry user data of various services.

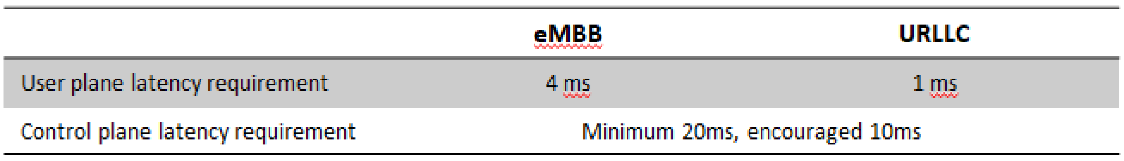
Data coming from these channels are multiplexed to MAC PDUs after Logical Channel Prioritization (Discussed in detail in Chapter 4) which are then mapped to transport channels. Error correction of these packets are done using hybrid ARQ. The main element of the MAC layer is the scheduler, which decides what data should be transmitted where and when, and it takes into account several parameters from across the system, including QoS requirements, available antennas, current traffic, available slots and frequency bands etc.

2.2.5 Radio Resource Control

While all the above are user plane protocols, RRC is a control plane protocol which is responsible for configuring and managing all the user plane protocols. It primarily handles broadcast of system information, configuration of radio bearers, channels, QoS flows etc, creating and destroying user plane protocol entities, interfacing between 5GC and NG RAN, mobility functions, measurement functions, connection management, and more.

2.3 Programming Framework and Paradigms

The 5G protocol stack is a bulky implementation but it also has stringent requirements for latencies. ITU's IMT-2020 RAN latency requirements are given below:



	eMBB	URLLC
User plane latency requirement	4 ms	1 ms
Control plane latency requirement	Minimum 20ms, encouraged 10ms	

Figure 2.5: IMT-2020 NG RAN Latency Requirements

Where:

- **User Plane Latency**: Defined as the one-way time to deliver a packet from the L2/L3 SDU ingress point to the L2/L3 egress point of the radio interface (assuming unloaded conditions, i.e. a single user, for small IP packets for both DL and UL)
- **Control Plane Latency**: The transition time from a most "battery efficient" state (i.e. idle state) to the start of continuous data transfer (i.e. active state).

To meet these requirements, several processes have to be parallelized and packets need to be handled efficiently, for which an abstracted variant of the DPDK library is used.

DPDK, or Data Plane Development Kit, is a set of data plane libraries and network interface driver controllers for fast packet handling. While deployed, it creates an Environment Abstraction Layer (EAL) for the specific software/hardware environment it is in, and this abstracts out the OS handling and parallelization which is taken care of automatically.

In the 5G testbed context, the most important uses of the DPDK library are zero-copy mechanisms and multithreading. Copying is an expensive operation since functions like `memcpy()` are $\mathcal{O}(n)$ and this is significant given that very few packet manipulations in the protocol stack exceed $\mathcal{O}(n)$ (Reassembly in RLC, for example, can be implemented as a merge sort on a linked list which is $\mathcal{O}(n \log n)$. This is explained in more detail in Chapter 3).

Thus, the DPDK elements abstracted to the Testbed Protocol Stack are not physically moved around; their pointers are manipulated in a way that they can be accessed sequentially. The important elements used in the Testbed are described in the following subsections.

2.3.1 ngPkt

An abstraction of `dpdk_rte_mbuf`, DPDK allows zero-copy manipulations along with reducing the number of misses in the CPU TLB table.

An `ngPkt` has a pivot/offset `data_off` 128 bytes to the right, and this pivot is used as the main reference point to add headers and data.

```
#define RTE_PKTMBUF_HEADROOM 128
#define MBUF_SIZE (2048 + sizeof(struct rte_mbuf)
+ RTE_PKTMBUF_HEADROOM)
```

It is divided into three sections:

- **Headroom**: This is used to store two kinds of data: protocol headers and the user's own metadata, if needed. Each layer's header is prepended starting from the pivot, i.e. the SDAP header is prepended so that its last byte is the pivot, the PDCP header is prepended to the starting point of the SDAP header, and so on. The metadata is added from byte 0 onwards from left to right, and it can be used to contain implementation-specific information within each user's L2 entity.
- **Data**: The data area is defined in the range of `data_off + buf_addr` to `data_off + data_len + buf_addr`, where `data_len` is its length. Data is appended to the right of `data_off`.
- **Tailroom**: With the default size as 0, it refers to something that `data_len` does not contain.

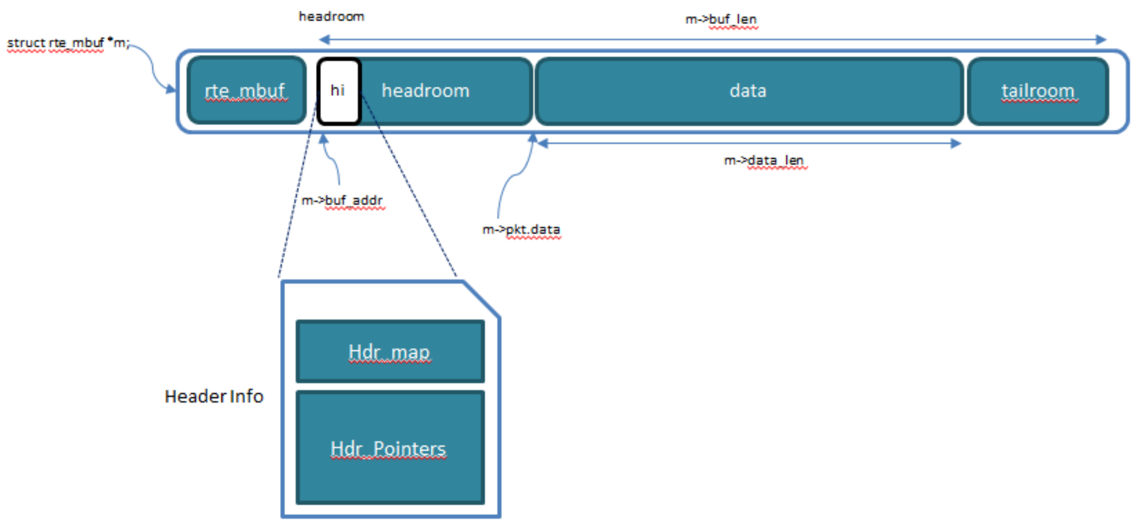


Figure 2.6: ngPkt Structure

2.3.2 ngPool

An abstraction of `rte_mempool`, it is an allocator of a fixed size object, and this is where all the `ngPkt` elements are stored. The `ngPool` automatically allocates memory to newly created `ngPkts`, enqueues and dequeues `ngPkts` and also handles a cache for speeding up frequent operations.

2.3.3 ngRing

An abstraction of `rte_ring`, this is a part of DPDK's ring library and serves as a ring buffer between two sublayers. While its size is fixed (and will have to be pre-decided based on throughput and protocol needs), it allows for efficient queue management because of the following features:

1. It is always FIFO and ensures packet order.
2. It can handle multiple consumers and multiple producers, which is especially useful when handling several entities, channels, and radio bearers.
3. It can also handle bulk (for a predefined number of packets) and burst (up to a predefined number of packets) enqueuing/dequeueing.

It is essentially an implementation of a circular buffer within DPDK's framework and the head and tail pointers rotate about the queue of `ngPkts` as follows:

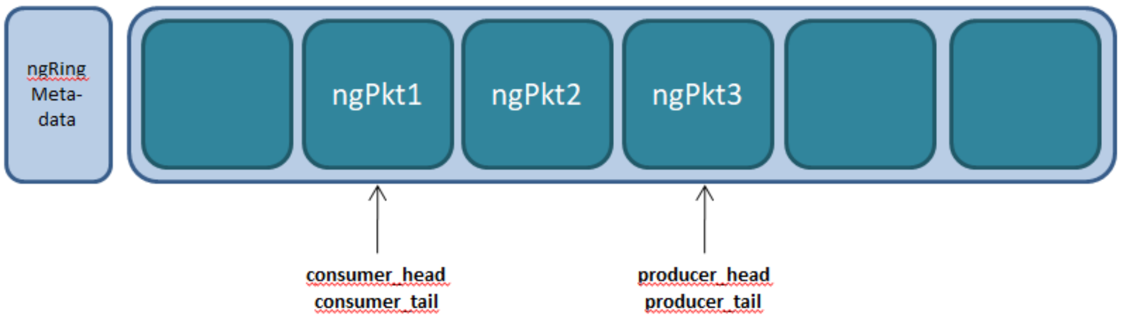


Figure 2.7: ngRing Structure

2.3.4 Other Framework Elements

In order to handle the above elements, timers, hashes, and threads are extensively used.

Timers enable function callbacks to be executed asynchronously, and can either be single-shot (one time) or multi-shot (periodic), depending on the usage. For example, the RLC reassembly timer is single-shot but L1/L2 control signalling is multi-shot for some parameters (like the TTI).

Hashes can be used to store any mappings. Some important uses of hashes are to store the QFI to DRB mapping, LCID to Radio Bearer mapping, sublayer configuration values etc.

Threads are useful to parallelize tasks across different cores. DPDK aids in abstracting `pthreads`, or physical threads, to `lthreads`, or logical threads, which can then be used to run parallel functions.

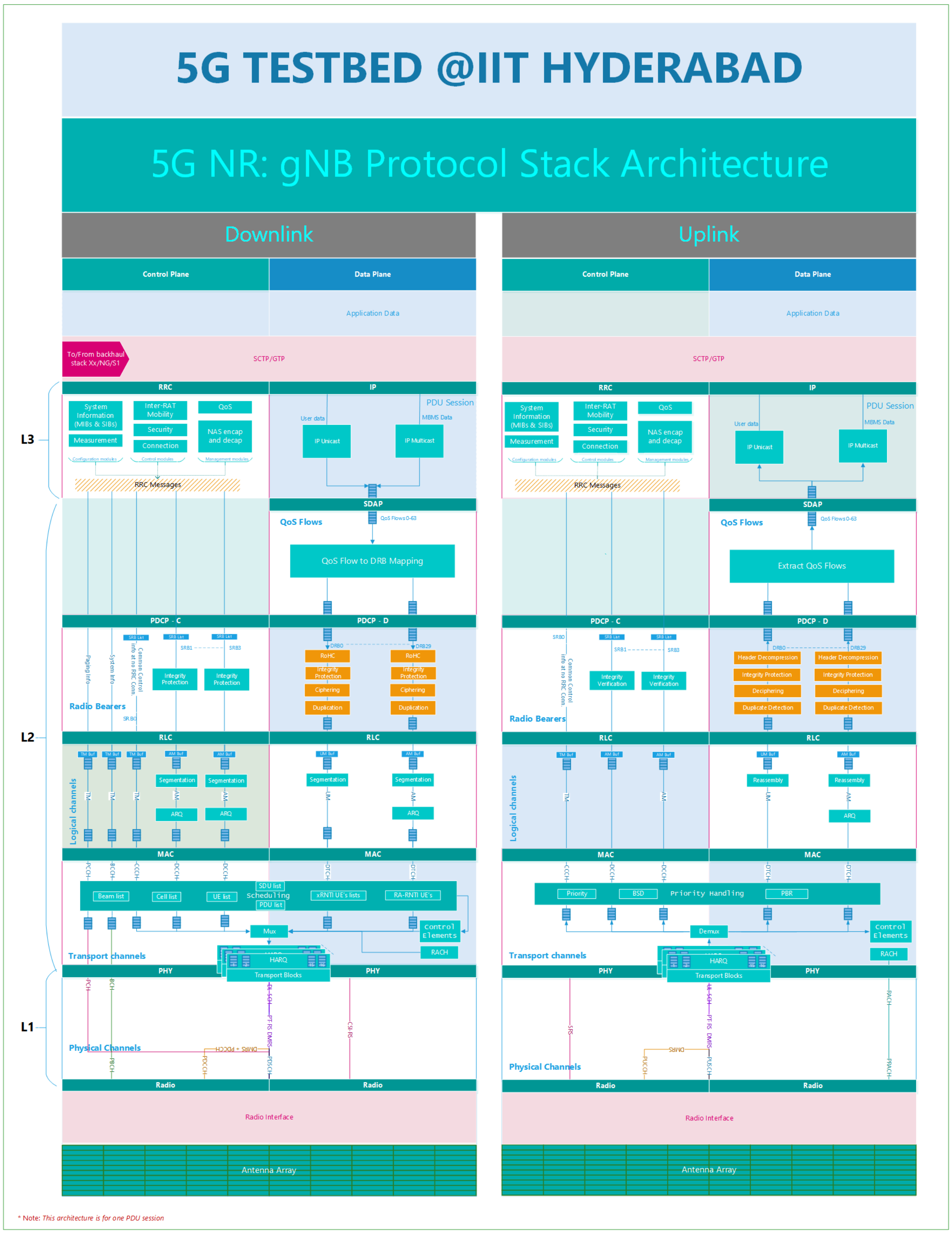


Figure 2.8: Testbed PS Architecture (Each sublayer is connected by ngRings)

CHAPTER 3

NR Radio Link Control

3.1 Introduction to RLC

As discussed earlier in Chapter 2, the RLC protocol handles segmentation and reassembly of PDCP PDUs into suitably sized RLC PDUs that are then fed into the MAC layer. It also handles retransmission of erroneous PDUs and duplicate removal.

3.2 Modifications from LTE RLC

There are some important differences between the RLC protocol used in LTE and that used in NR, which are:

- RLC does not do concatenation in NR and only performs segmentation. The motivation behind this is to reduce latency. For concatenation, the RLC PDU cannot be assembled until the uplink grant is given as the transport block size cannot be known in advance, and this will cause an extra delay.
- Since an important feature of NR is the flexibility of the entire 5G system, it offers various avenues to create network splits. One such new feature in NR is the F1 interface, which lies between PDCP and RLC. This is done to split the Central Unit (CU) and the Distributed Unit (DU). The F1 interface consists of F1-C and F1-U for the control plane and the user plane respectively (Refer to Fig 1.7 for more details). This split also allows for traffic aggregation between NR and E-UTRA transmissions, thereby supporting the non-standalone mode too. Moreover, it supports dual connectivity.
- RLC in NR does not ensure in-sequence delivery; that functionality has been moved to PDCP in the form of reordering. This is again done to reduce the latency, since the later packets do not have to wait for retransmission of an earlier missing packet before being delivered to the higher layers, but can instead be forwarded immediately. This also removes the need of a large memory buffer, which could overflow especially during any burst arrivals.
- Subsequently, the header formats for RLC packets have been changed in NR. They will be discussed in further detail in the following sections about the roles aand operations of the three RLC modes.

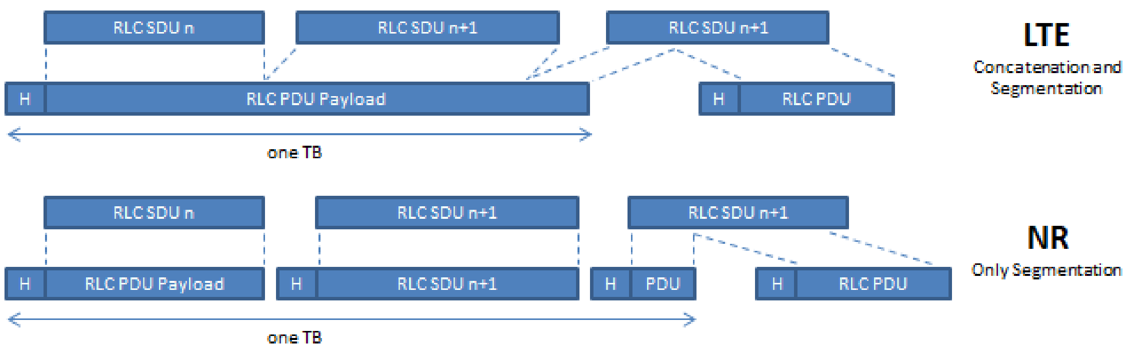


Figure 3.1: RLC Packet Handling in LTE and NR

3.3 RLC Modes of Operation

The RLC protocol has three modes of operation, namely the Transparent Mode (TM), Unacknowledged Mode (UM), and Acknowledged Mode (AM). All of these are used for forwarding different kinds of packets based on their requirements.

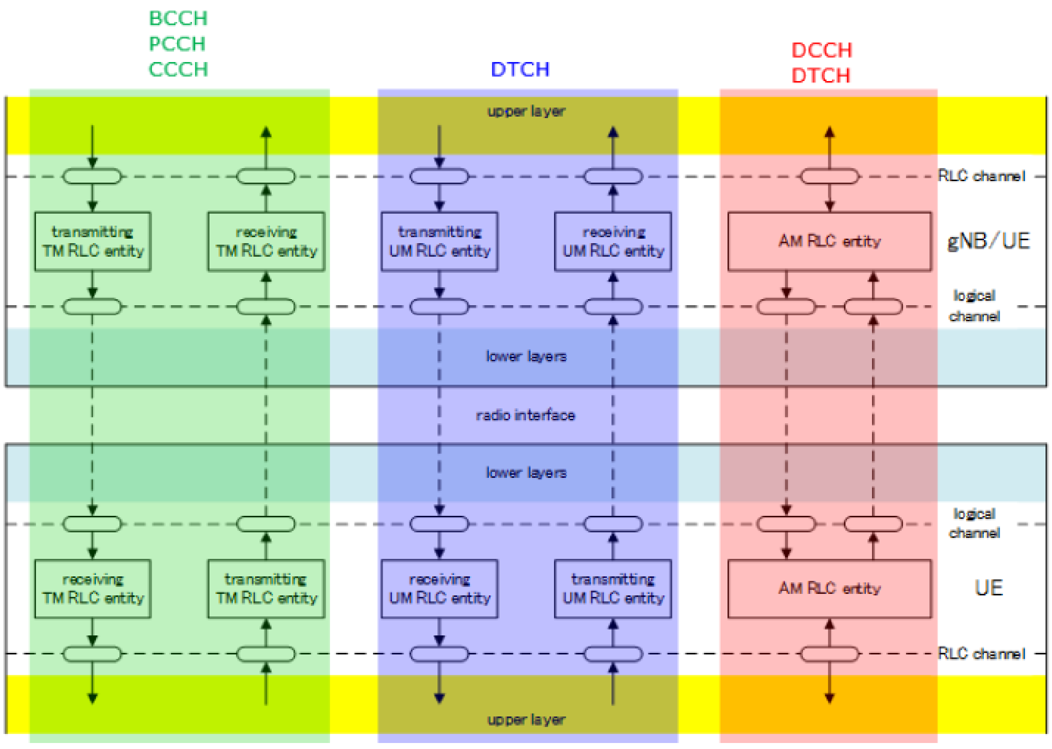


Figure 3.2: RLC Modes and Corresponding Logical Channels (as per TS 38.322)

3.3.1 Transparent Mode (TM)

The Transparent Mode (TM) is used by packets to bypass RLC without any modifications. No header addition, segmentation, or any other function is implemented on these packets. This is used for control-plane broadcast information that flows through the Broadcast Control Channel, Paging Control Channel, and the Common Control Channel, since these messages should reliably reach multiple users. The sizes of these messages are chosen such that they are reliable and don't need segmentation, and can thus bypass the RLC operations.

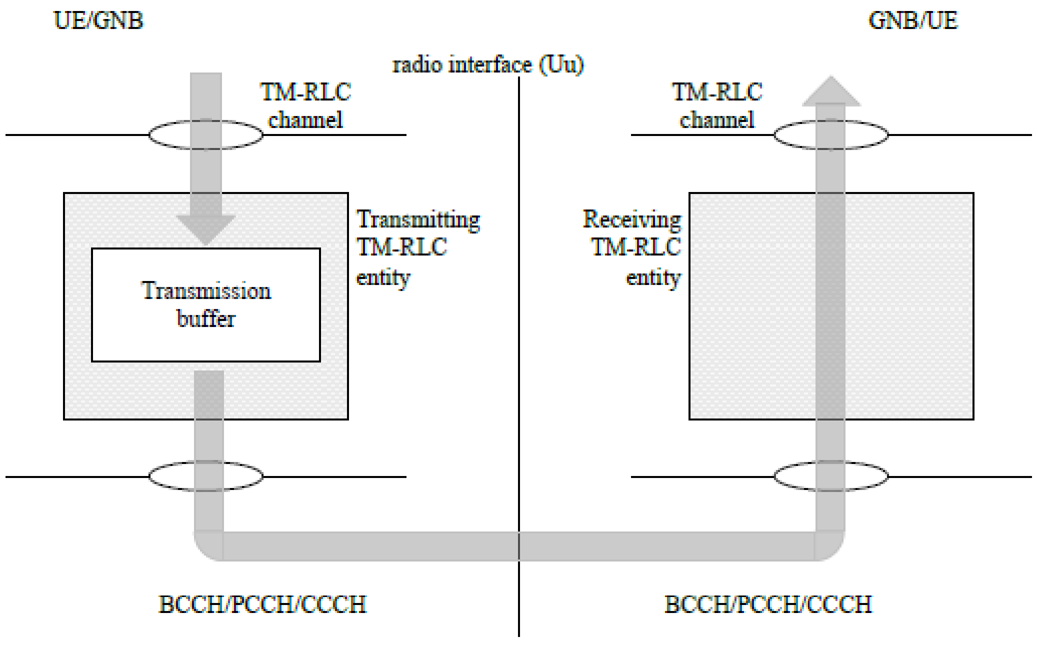


Figure 3.3: Model of the transmitter and receiver TM entities (As per TS 38.322)

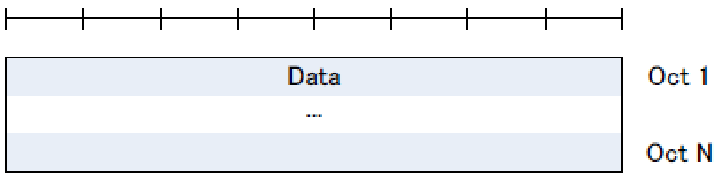


Figure 3.4: TM Data PDU (Does not consist of any headers; as per TS 38.322)

3.3.2 Unacknowledged Mode (UM)

The Unacknowledged Mode (UM) is used by packets in the Dedicated Traffic Channel (both downlink and uplink) when segmentation is required. The UM mode is especially useful for continuous data streams when error-free delivery is not required; for example, voice calling or VoIP. It is in some sense analogous to UDP in terms of the use cases that it caters to.

The transmitting UM entity has two main operations on an incoming PDCP PDU/RLC SDU: segmentation and header addition. Segmentation is done to ensure that the incoming packets appropriately fit into transport blocks which are transmitted over radio through the lower layers. The size of the transport block is provided by the MAC layer to the RLC sublayer through L1/L2 control signalling (which is generally done every tick), and this is used to segment RLC SDUs (if required).

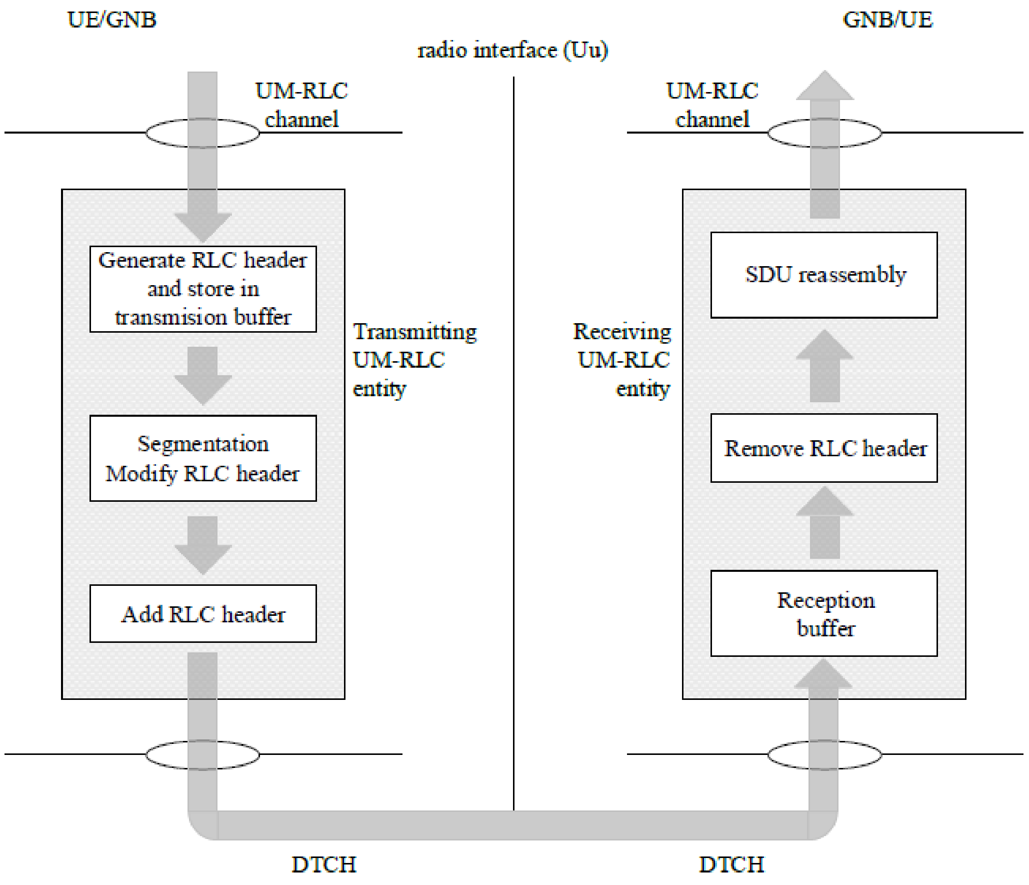


Figure 3.5: Model of the transmitter and receiver UM entities (As per TS 38.322)

A header is added to each outgoing RLC PDU based on whether segmentation is done. If the PDU is a segment, then the header indicates its position within the unsegmented RLC SDU, which the receiving RLC entity uses to reassemble the packet before removing the header and delivering to the upper layers.

UM Header Formats

The RLC PDU is structured as a bit string onto which headers are added. Depending on whether a PDU has undergone segmentation and if yes, its position, an RLC header is prepended.

RLC SDUs are byte aligned, and is included in an RLC PDU from the first bit onwards. There are multiple formats for the different cases and in all of them, the first and most significant bit is the leftmost bit, and the last and least significant bit is the rightmost bit. However, this may change in the implementation due to the Endianness of the processors used.

The most basic RLC UM PDU header structure is shown in the figure below, when the PDU contains an entire SDU.

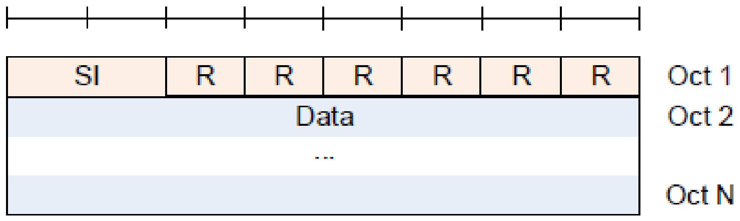
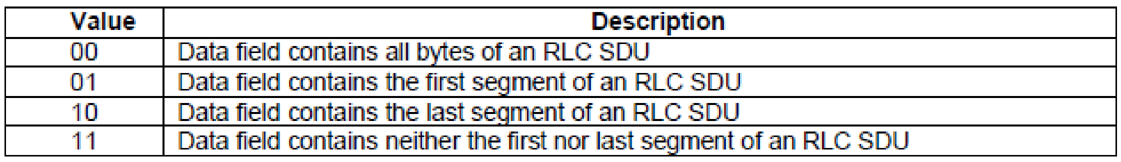


Figure 3.6: UMD PDU containing a complete RLC SDU (As per TS 38.322)

Here, the first two bits of the header store the *Segmentation Info*, or SI in short. The different SI values can be interpreted as follows:



Value	Description
00	Data field contains all bytes of an RLC SDU
01	Data field contains the first segment of an RLC SDU
10	Data field contains the last segment of an RLC SDU
11	Data field contains neither the first nor last segment of an RLC SDU

Figure 3.7: SI Values

Thus, for the UMD PDU described above, the SI field will be set to 00. The Reserved fields (indicated by R) are present to ensure byte aligning, and are set to 0 by default.

The figure below shows a more complex PDU format. Along with the SI and R fields, it introduces two new fields - the Sequence Number (SN) field and the Segment Offset (SO) field, both of which are used when an RLC SDU is segmented.

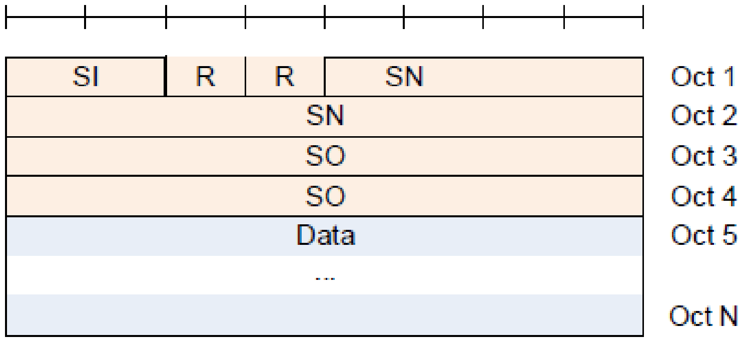


Figure 3.8: UMD PDU with 12 bit SN and with SO (As per TS 38.322)

The SN field for a UM RLC entity can be of two sizes - 6 bits or 12 bits - depending on the configuration of the RLC entity (which is done by RRC). The above PDU has a 12 bit SN which is used to indicate the sequence number of the RLC SDU to which this PDU belongs. In RLC UM, the sequence number is incremented by one for every segmented RLC SDU.

The SO field indicates the position of the RLC SDU segment in bytes within the original RLC SDU. Specifically, it indicates the position within the original SDU to which the first byte of the segment corresponds. The SO of the segment which contains the SDU's first byte is numbered from 0. A UMD PDU carrying the first segment of an RLC SDU does not contain the SO field in its header, and is just identified using the SI field (which will be set to 01).

In all PDU formats, the data field's granularity is one byte, and its maximum size is the maximum size of a PDCP PDU.

```
struct RlcUm12Hdr
{union
 {struct
  {uint8_t si :2;        /*! Segment info  */
   uint8_t sn :6;        /*! Sequence number  */
  }fields;
  uint8_t first;
 }firstbyte;
 uint16_t so;            /*! Segment offset  */
};
```

UM Transmit and Receive Operations

The UM segmentation, header addition, buffering, and reassembly algorithms use several state variables, counters, constants, and timers, all of which are used to track the state of the UM entity.

All state variables and counters are non-negative, and can take values from 0 to $2^{snFieldLength-1}$, i.e. 0 - 63 for a 6 bit SN and 0 - 4095 for a 12 bit SN. All the arithmetic operations performed by the UM entity are affected by the UM modulus, i.e. $FinalVal = (Result)mod(64)$ for a 6 bit SN and $FinalVal = (Result)mod(4096)$ for a 12 bit SN. In other words, all variables are constrained by the maximum value possible for the SN field to hold.

The modulus base is also used when any arithmetic comparisons are made between state variables or SN values. For the receiving UM entity, the modulus base is given by $RX\_Next\_Highest - UM\_Window\_Size$. This modulus base is subtracted from all the values involved after which an absolute comparison is performed. For example, the expression $(RX\_Next\_Highest - UM\_Window\_Size) \leq SN < RX\_Next\_Highest$ is evaluated as $(RX\_Next\_Highest - UM\_Window\_Size) - Mod\_Base]modulo(UM\_Modulus) \leq [SN - Mod\_Base]modulo(UM\_Modulus) < [RX\_Next\_Highest - Mod\_Base]modulo(UM\_Modulus)$, where the UM Modulus is given by $UM_Modulus = 2^{snFieldLength-1}$.

The transmitting UM RLC entity maintains the following state variables:

- TX_Next: This state variable holds the SN to be assigned for the newly formed UMD PDU. Initially set to 0, it is updated as and when the UM RLC entity submits a UMD PDU that includes the last segment of the corresponding RLC SDU to the lower layers.

Each receiving UM RLC entity maintains the following state variables:

- RX_Next_Reassembly: This is the UM receive state variable which holds the value of the earliest SN that is still considered for reassembly. It is initalized to 0.
- RX_Timer_Trigger: This state variable holds the value of the SN following the SN which triggered t-Reassembly (a reassembly timer)
- RX_Next_Highest: This holds the value of the SN following the SN of the UMD PDU with the highest SN among all received UMD PDUs thus far. This serves as the higher edge of the reassembly window and is initially set to 0.

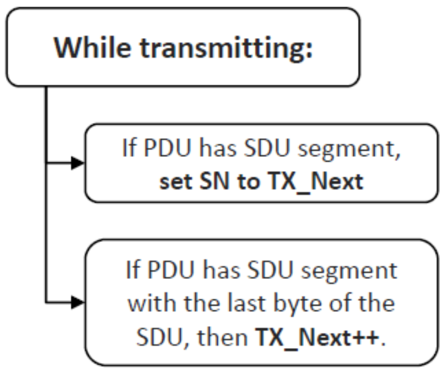

While transmitting:
If PDU has SDU segment, set SN to TX_Next
If PDU has SDU segment with the last byte of the SDU, then TX_Next++.

Figure 3.9: UM Transmit Operations

The UM entity also maintains the following constant:

- UM_Window_Size: This is used by the receiving UM RLC entity to define SNs of those UMD PDUs that can be received without advancing the reassembly window. For a 6 bit SN, the window size is 32 and for a 12 bit SN, it is 2048.

The receiving entity, as mentioned above, maintains a reassembly window with the higher edge being RX_Next_Highest and the size set to UM_Window_Size.

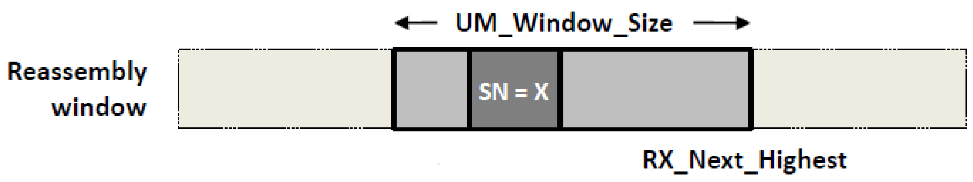


Figure 3.10: UM Reassembly Window

In the code implementation of the standalone RLC sublayer, the lower layers are abstracted by an IP socket, and the PDUs are fed over Ethernet to the receiving entity for reassembly. Similarly, the higher layers have been replaced with an ad hoc application layer that performs file handling.

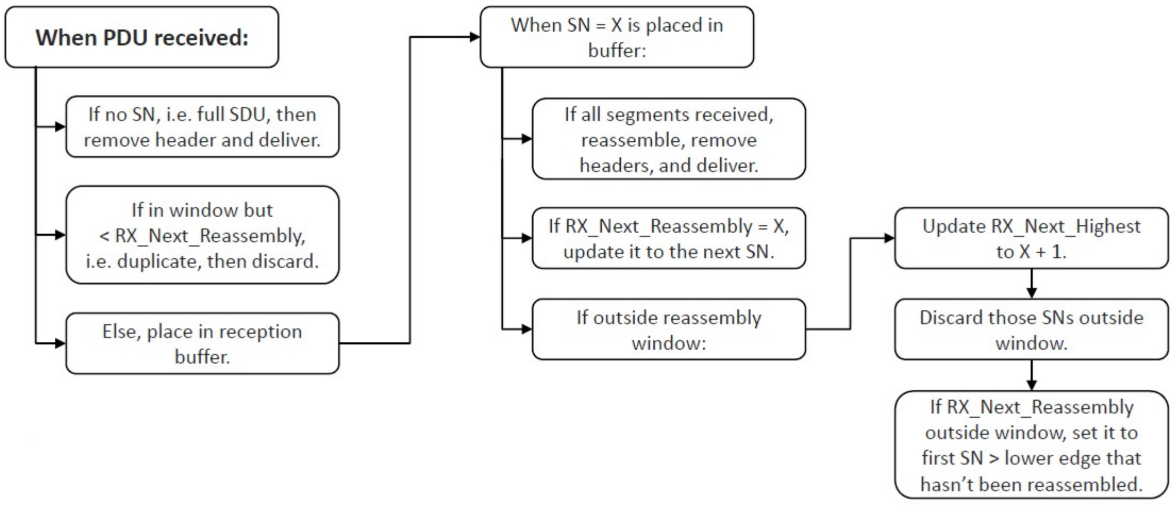

When PDU received:
If no SN, i.e. full SDU, then remove header and deliver.
If in window but < RX_Next_Reassembly, i.e. duplicate, then discard.
Else, place in reception buffer.
When SN = X is placed in buffer:
If all segments received, reassemble, remove headers, and deliver.
If RX_Next_Reassembly = X, update it to the next SN.
If outside reassembly window:
Update RX_Next_Highest to X + 1.
Discard those SNs outside window.
If RX_Next_Reassembly outside window, set it to first SN > lower edge that hasn't been reassembled.

Figure 3.11: UM Receive Operations

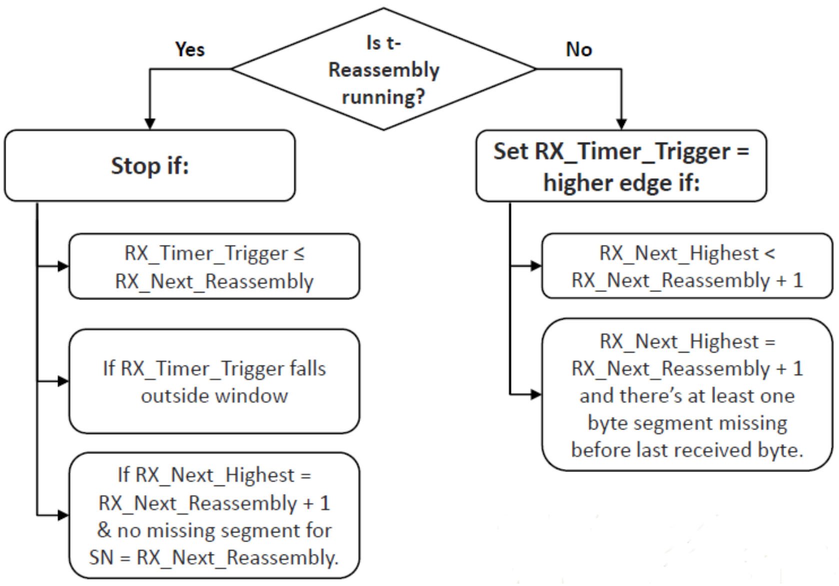

Is t-Reassembly running?
Yes
No
Stop if:
RX_Timer_Trigger ≤ RX_Next_Reassembly
If RX_Timer_Trigger falls outside window
If RX_Next_Highest = RX_Next_Reassembly + 1 & no missing segment for SN = RX_Next_Reassembly.
Set RX_Timer_Trigger = higher edge if:
RX_Next_Highest < RX_Next_Reassembly + 1
RX_Next_Highest = RX_Next_Reassembly + 1 and there's at least one byte segment missing before last received byte.

Figure 3.12: UM Reception Timer Operations

The algorithms above highlight how the receiving UM entity functions. Along with a reassembly window, a reassembly timer also runs that is used to detect any packet losses in the lower layers. Only one instance of it can run at one time, and is controlled by the values of RX_Timer_Trigger, RX_Next_Reassembly, and RX_Next_Highest. Packets of the same SN are considered for reassembly if all segments have arrived within the reassembly timer and window.

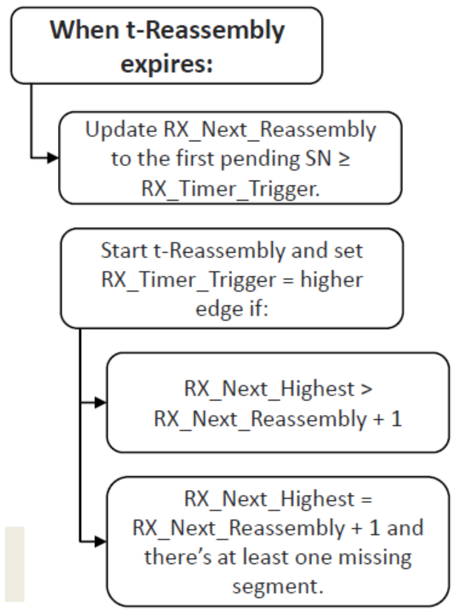

When t-Reassembly expires:
Update RX_Next_Reassembly to the first pending SN ≥ RX_Timer_Trigger.
Start t-Reassembly and set RX_Timer_Trigger = higher edge if:
RX_Next_Highest > RX_Next_Reassembly + 1
RX_Next_Highest = RX_Next_Reassembly + 1 and there's at least one missing segment.

Figure 3.13: UM Reception Timer Expiry Handling

UM Segmentation

After receiving the transport block size from the MAC layer, the RLC layer decides whether segmentation is required. There are four cases that a packet can fall under:

1. **Case 1**: PDU is the same size as transport block. Here, add the PDU to the transport block and send to lower layers.

2. **Case 2**: PDU is smaller than the transport block with sufficient space for at least one extra segment (i.e. the space is greater than 4 bytes as the RLC header itself is 4 bytes at most). Then, if the allowed TB size is X, take the first X bytes of the next SDU and add to the TB. If there is more space left, then segment the next SDU as well, and so on. In each case, add the appropriate headers.

3. **Case 3**: PDU is smaller than the transport block but the remaining space is not enough to hold any payload (i.e only a part or the complete header of another segment will fit in). In this case, send the TB to lower layers (with some space wasted).

4. **Case 4**: PDU is larger than the transport block. In this case again, segment the PDU, add the appropriate header, and then update the remaining TB size.

In each of these cases, different decisions are taken, and the recommended UM segmentation algorithm is given as follows:

```
Input: Remaining transport block size RemTBSize, head of PDCP-RLC Ring
       Buffer RLCSDUBufHead
Output: Outgoing transport block (to lower layers)
SDUSize = RLCSDUBufHead->size;
/* RLC header size is at most 4 bytes                          */
while RemTBSize $\geq 0$ do
    if SDUSize $\leq RemTBSize - 4$ then
        SDU = Pop(RLCSDUBufHead);
        AddHdr(SDU, "FULLSDU");        // No SN as complete SDU
        TB.append(SDU);
        RemTBSize = RemTBSize-sizeof(SDU);           // =0 in Case 1
        RLCSDUBufHead++;
        SDUSize = RLCSDUBufHead->size;
    end
    if SDUSize > RemTBSize - 4 & RemTBSize > 4 then
        PDU = FirstXBytes(X, RLCSDUBufHead);         // Take first X
         bytes, X = RemTBSize-4
        rmv(X, RLCSDUBufHead); // Remove first X bytes in buf
        AddHdr(PDU, "SEGMENT", "Y");      // Check if segment is
         first or last or mid of SDU. Y = SI Flag
        RemTBSize = RemTBSize-sizeof(SDU);
    end
    if 0 $\leq RemTBSize \leq 4$ then
        Add to RLC-MAC buffer;
        break;
    end
    end
```

Algorithm 1: RLC UM Segmentation

In the above algorithm, the 1 and the 4 are subtracted from the remaining TB sizes as they are the sizes of the RLC headers for no SN and for 12 bit SN with SO respectively.

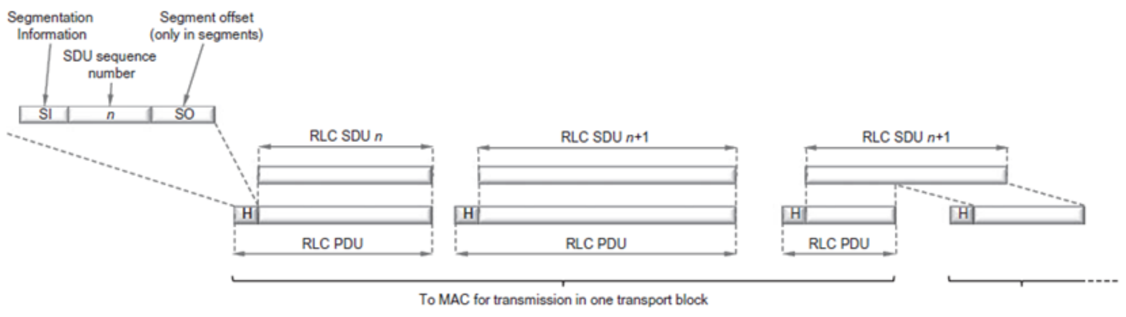


Figure 3.14: Generation of RLC PDUs from SDUs

The above can be implemented using the Testbed DPDK framework by using `ngPkt`-related functions. Header addition can be done by using `rte_pktmbuf_prepend`, and data can be added using `rte_pktmbuf_prepend`. If an `ngPkt` does not have enough space for a full SDU, segmentation can be done by using `rte_pktmbuf_trim`. Then, the different segments can be chained together where their metadata will contain the pointers to the next segments. All of these will ensure adherence to the zero-copy requirement.

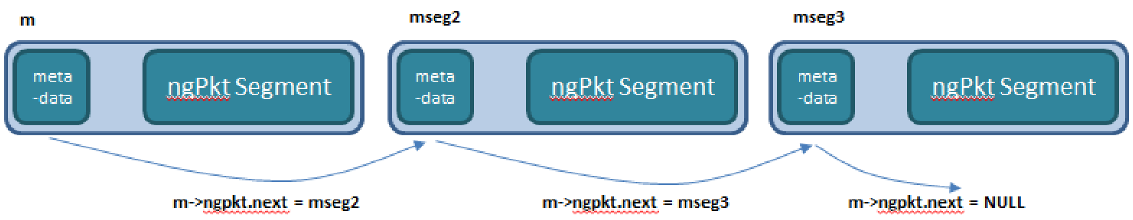


Figure 3.15: Segmentation using ngPkts

UM Reassembly

On the receiving side, as mentioned earlier, packets are placed in the buffer if they are within the reassembly window and are not duplicates. To ensure that zero copies happen, a linked list-like structure can be used to receive SNs.

Each ngPkt can be linked to a metadata header corresponding to its SN. The metadata will contain the SN, the number of received bytes (RxBytes), and the number of remaining bytes (RemBytes). The ngPkts are chained to the sublist as and when they arrive (which is not necessarily in order), and RxBytes is incremented.

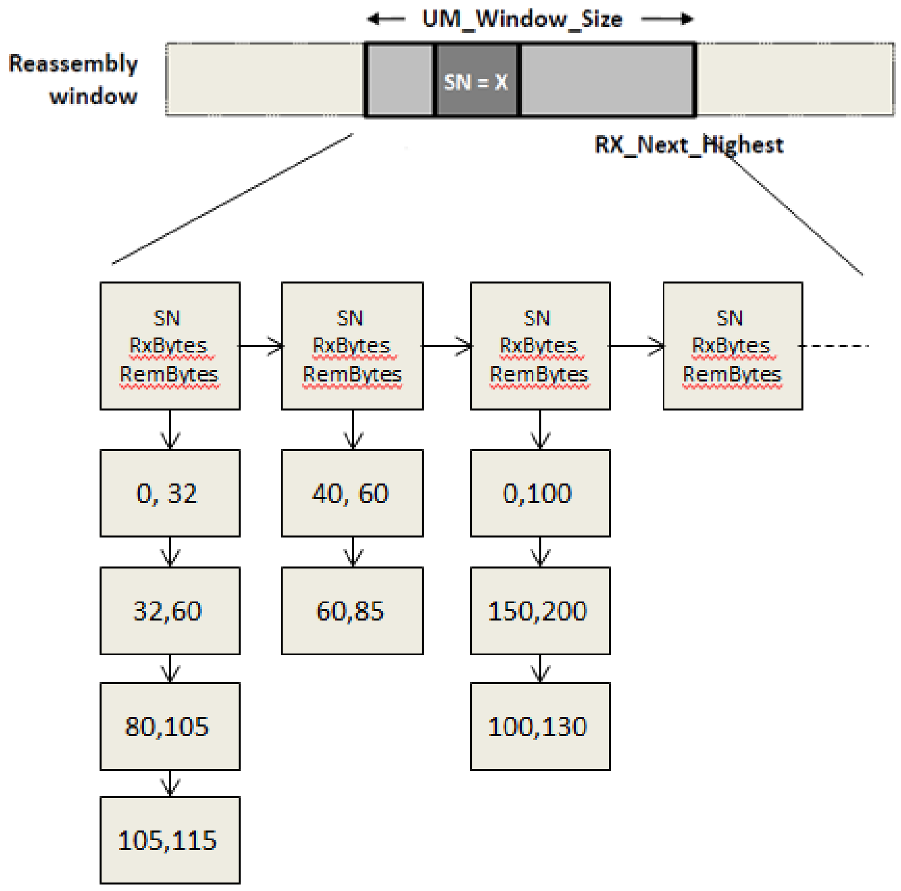


Figure 3.16: UM Reassembly Structure

RemBytes cannot be estimated until the last segment for that SN has arrived (which can be detected by its SI). Once it arrives, the number of bytes in the SDU is given by SO_LastPkt + sizeof(LastPkt), and the metadata can be updated accordingly.

Once all bytes have been received (i.e. RxBytes = RemBytes), reassembly can be done using SO as a reference. Merge sort is the recommended algorithm for linked lists due to its relatively low time complexity $\mathcal{O}(n \log n)$. Since a linked list is used, only pointer manipulations suffice for sorting thereby ensuring that no copies are made.

Finally, when reassembly for a particular SN is done, the ngPkt RLC headers can be removed using `rte_pktmbuf_adj` and the segments can be merged using the function `rte_pktmbuf_linearize` before being sent to the higher layers.

3.3.3 Acknowledged Mode (AM)

The Acknowledged Mode (AM) is used by packets in the Dedicated Control Channel (DCCH) and the Dedicated Traffic Channel (DTCH) (both uplink and downlink). Apart from segmentation and reassembly, AM also performs retransmission of erroneous or unreceived packets, and is hence more reliable (it is analogous to TCP in some sense).

In AM, the RLC entity is bidirectional - i.e unlike UM where there is a transmitting and receiving entity, the same AM entity handles both since PDU receptions here have to be acknowledged back to the transmitting side.

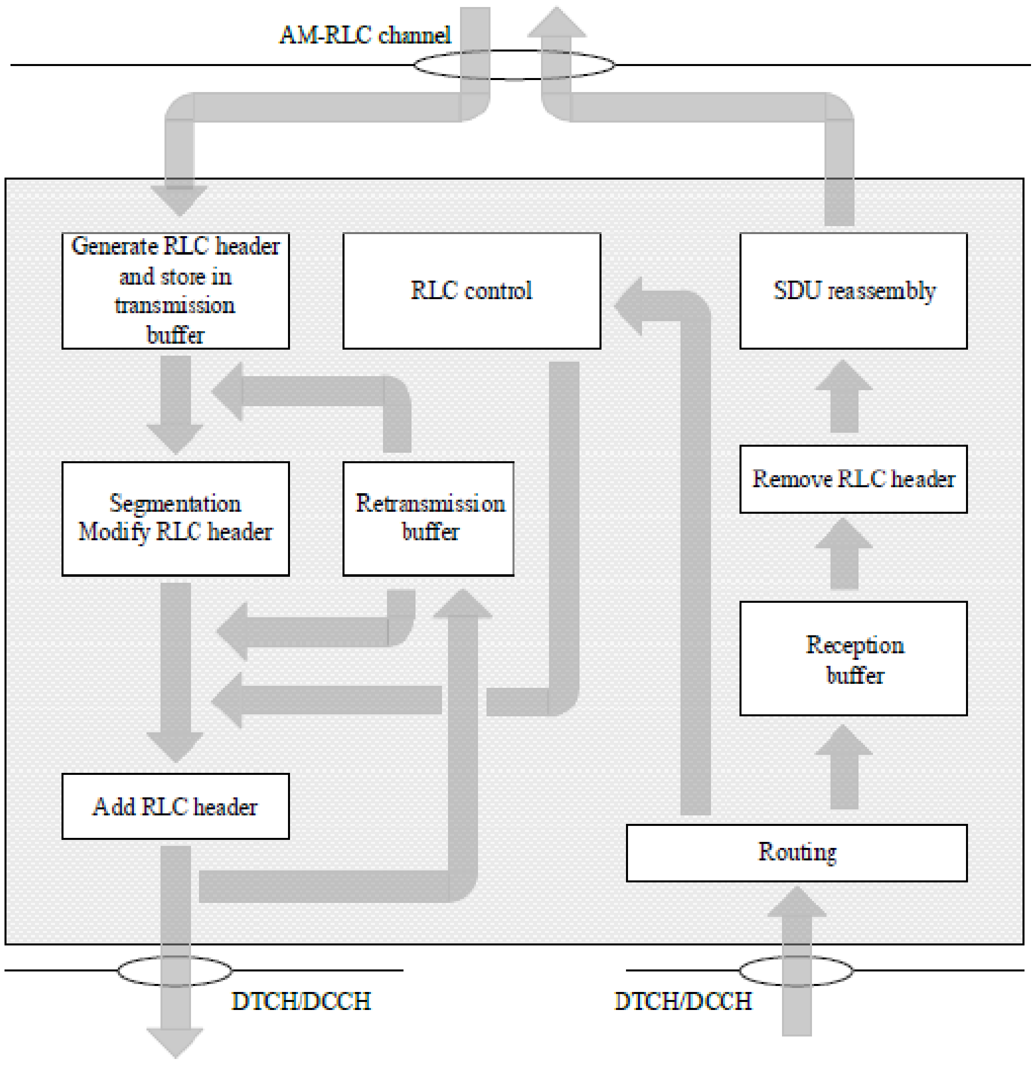


Figure 3.17: Model of the Ackowledged Mode entity (As per TS 38.322)

A header is added to each outgoing RLC PDU based on whether segmentation is done. If the PDU is a segment, then the header indicates its position within the un-segmented RLC SDU, which the receiving RLC entity uses to reassemble the packet before removing the header and delivering to the upper layers.

AM Packet Formats

In AM, there are two kinds of PDUs - the AM Data PDU (AMD PDU) and the RLC control PDU. The control PDU is prioritized over AMD PDUs for transmission.

The AMD PDU format is mostly similar to the UMD PDU format with the addition of two new bits; a D/C bit which indicates whether the PDU is a data PDU or a control PDU, and a poll bit P which is set to 1 whenever a status report is requested. The AM SN is either 12 bits long or 18 bits long, which will be configured by RRC. Also, an SN is given to every sent PDU, even if it is a full SDU.

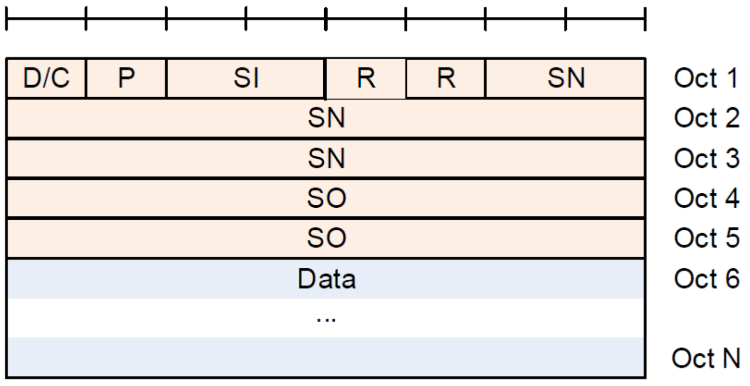


Figure 3.18: AMD PDU with 18 bit SN and SO (As per TS 38.322)

In addition, a status report is sent by the receiving side of the AM entity to the transmitting side whenever requested using the poll bit.

The status report consists of several variables that indicate the SNs/SOs of the packets that have not been received. These variables are as follows:

- **CPT**: The Control PDU Type (CPT) field is set to 000 for a STATUS PDU and the remaining values are reserved for later usage.
- **ACK_SN**: This field states the highest SN that has been received so far. ACK_SN is set to the SN after the highest SN received.
- **E1**: The E1 (Extension 1) field indicates if a set of NACK_SN, E1, E2, and E3 fields follow.
- **NACK_SN**: This field indicates the SN of a lost RLC SDU or segment.
- **E2**: This indicates whether a set of SOStart and SOEnd fields follow.

- **SOStart**: This indicates the start point of a range of SOs that have been reported as missing. There can be multiple SOStarts for the same NACK_SN.
- **SOEnd**: This indicates the end point of a range of SOs that have been reported as missing. There can be multiple SOEnds for the same NACK_SN.
- **E3**: This indicates whether a range of SNs have not been received.
- **NACK range**: This indicates the number of consecutive SNs that have not been received, starting from and including NACK_SN.

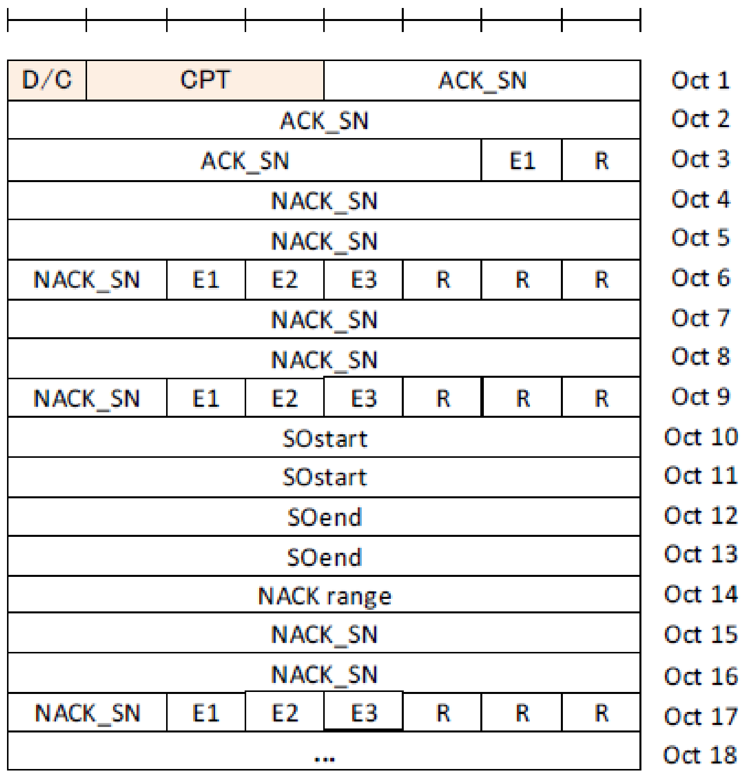


Figure 3.19: AM STATUS PDU with 18 bit SN (As per TS 38.322)

AM Transmit and Receive Operations

The AM segmentation, header addition, buffering, reassembly, polling, and status reporting algorithms use several variables to track the entity state. All state variables follow rules similar to that in UM.

In AM, there is a transmitting window too, and the following variables are maintained by the transmitting side:

- TX_Next: Holds the SN for the newly generated PDU on the transmission side

- TX_Next_Ack: Holds the SN for the next SDU for which ACK is to be received in-sequence. It serves as the lower edge of the transmitting window.

It also maintains the following constant:

- AM_Window_Size: Its purpose is similar to that in UM, but is set to 2048 for a 12 bit SN and 131072 for an 18 bit SN.

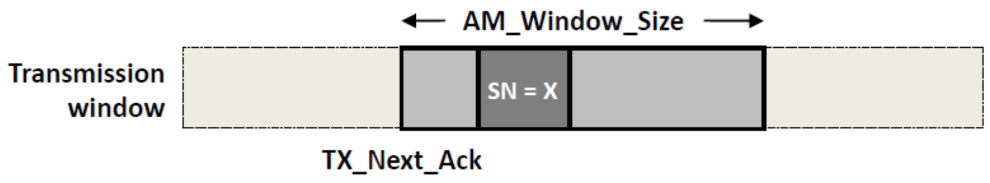


Figure 3.20: AM transmission window

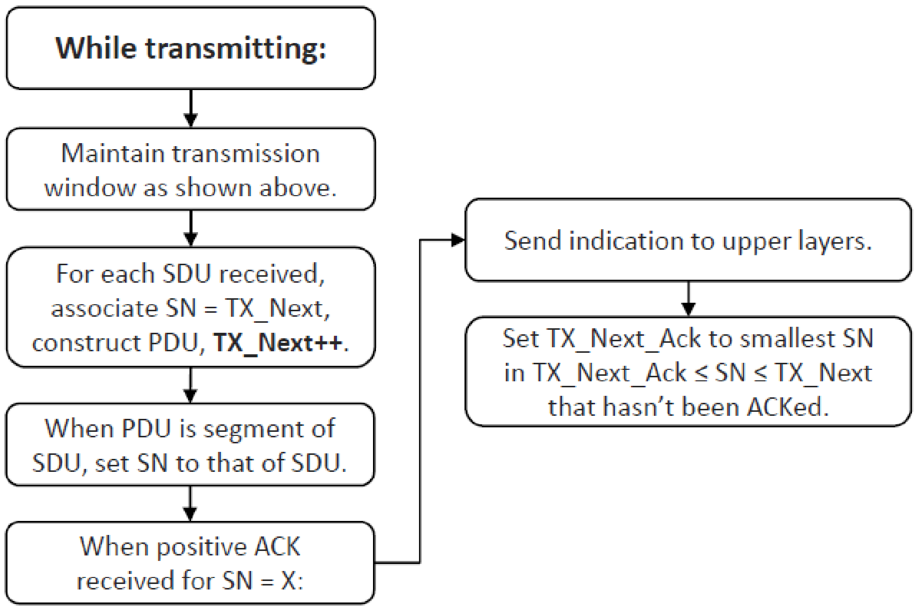

While transmitting:
Maintain transmission window as shown above.
For each SDU received, associate SN = TX_Next, construct PDU, TX_Next++.
When PDU is segment of SDU, set SN to that of SDU.
When positive ACK received for SN = X:
Send indication to upper layers.
Set TX_Next_Ack to smallest SN in TX_Next_Ack ≤ SN ≤ TX_Next that hasn't been ACKed.

Figure 3.21: AM transmit operations

There is a reassembly window on the receiving side too. It discards any SN received outside the window or any duplicate SN. It also keeps track of the following variables which are used for reassembly:

- RX_Next: SN following the last in-sequence fully received SDU that serves as the lower edge of the reassembly window. It is updated whenever SN = RX_Next is received.
- RX_Next_Highest: The SN following the SDU with the highest SN among received SDUs.

- RX_Highest_Status: Highest possible SN which can be indicated by ACK_SN in the STATUS PDU.
- RX_Next_Status_Trigger: The SN following that which triggered t-Reassembly.

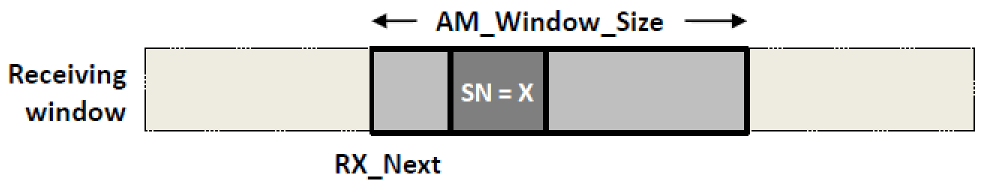


Figure 3.22: AM reassembly window

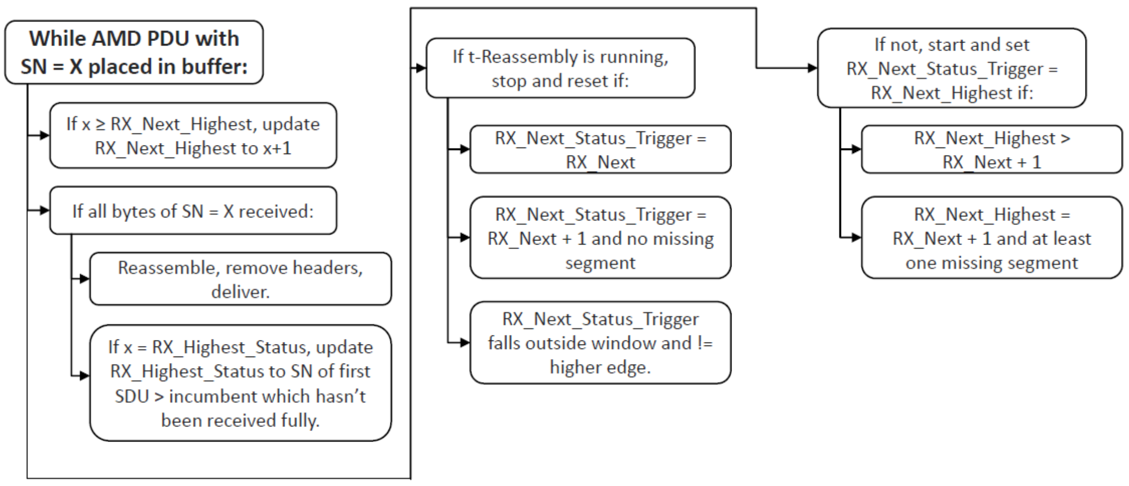

While AMD PDU with SN = X placed in buffer:
If x ≥ RX_Next_Highest, update RX_Next_Highest to x+1
If all bytes of SN = X received:
Reassemble, remove headers, deliver.
If x = RX_Highest_Status, update RX_Highest_Status to SN of first SDU > incumbent which hasn't been received fully.
If t-Reassembly is running, stop and reset if:
RX_Next_Status_Trigger = RX_Next
RX_Next_Status_Trigger = RX_Next + 1 and no missing segment
RX_Next_Status_Trigger falls outside window and != higher edge.
If not, start and set RX_Next_Status_Trigger = RX_Next_Highest if:
RX_Next_Highest > RX_Next + 1
RX_Next_Highest = RX_Next + 1 and at least one missing segment

Figure 3.23: AM receive operations

Segmentation and reassembly in AM are fairly similar to that in UM, with the addition of status reporting. Every packet in AM has to be acknowledged, for which the receiving side of the AM entity sends a status report to the transmitting side using ARQ (Automatic Repeat reQuest).

After every poll bit transmission by the transmitting side, it keeps a counter and a timer of how many packets were sent, before it polls again. When this reaches the receiving side, it puts together a STATUS PDU and sends it back. Instead of explicitly acknowledging every packet, it acknowledges that all packets have been received till a particular SN *except* those that have not, for which it lists out the SNs and generates a NACK.

The AM transmitting side also maintains a retransmission window and the packet is considered if TX_Next_Ack $\leq$ SN < TX_Next. Every time a packet is retransmitted, a retransmission counter is maintained and if the packet cannot be sent after a maximum amount of allowed retransmissions (that is configured by RRC), RLC will indicate to the upper layers.

A special case that could arise is that the packet considered for retransmission may not necessarily fit in the transport block allocated for that particular TTI. When this happens, resegmentation is required, and this is an optional feature. However, it is recommended as the TBs are used more effectively then rather than just being padded.

3.4 Standalone RLC Implementation

In the standalone implementation of the RLC sublayer, the upper layers have been abstracted by an application layer that performs file handling to load and write the file from the transmitting and receiving entity respectively. The lower layers have also been abstracted by IP sockets that can send across data after RLC processing.

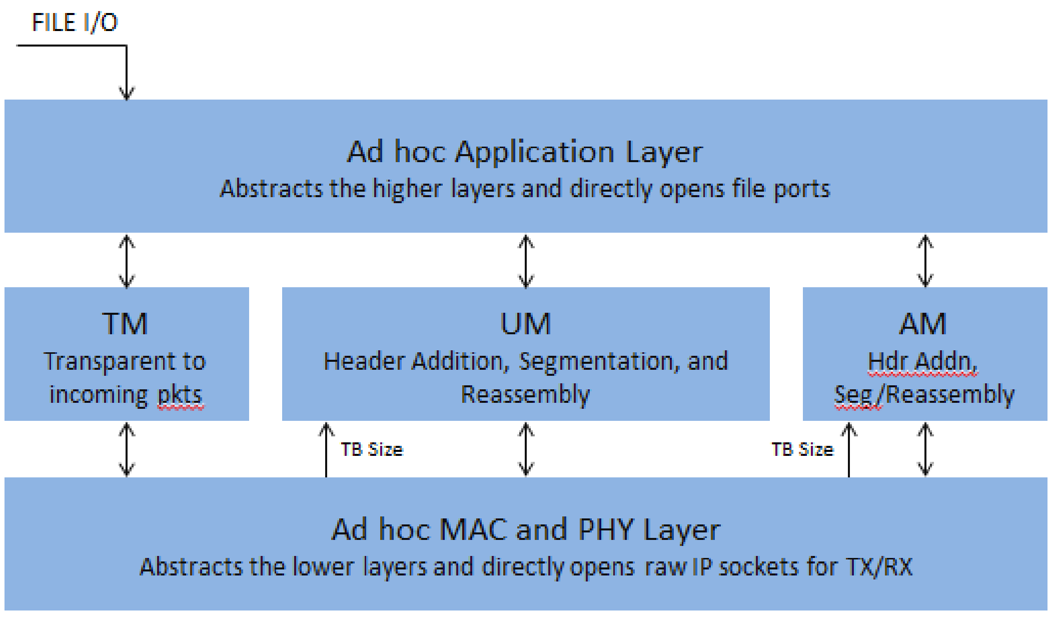

FILE I/O
Ad hoc Application Layer
Abstracts the higher layers and directly opens file ports
TM
Transparent to incoming pkts
UM
Header Addition, Segmentation, and Reassembly
AM
Hdr Addn, Seg/Reassembly
TB Size
TB Size
Ad hoc MAC and PHY Layer
Abstracts the lower layers and directly opens raw IP sockets for TX/RX

Figure 3.24: Standalone RLC Implementation Structure

This implementation performs header addition and removal for all three modes along with segmentation and reassembly. The size of the segmented PDU can be configured and this variable can be given to the L1/L2 control signalling interface later while integrating with the rest of the stack.

CHAPTER 4

NR Uplink Priority Handling

4.1 Introduction

Uplink priority handling in NR is a part of the uplink scheduler. While the uplink scheduler is similar to the downlink scheduler, it is distributed across several UEs as opposed to being centralized in the gNB. Moreover, the UEs have stricter power and frequency constraints and are allowed only to transmit when they receive a grant from the gNB.

Unlike the downlink scheduler which is left to the developers' implementation, the uplink priority handler has clearly been defined in the standard so that it will be the same across UEs and can be finely controlled by the gNB downlink scheduler. This is because a given scheduling grant applies to a specific uplink carrier of the device and not a particular logical channel. However, the configurations for priority handling are done by RRC in the gNB (that uses several parameters from the core network) which implies that UEs can be configured for several different use cases. This configuration algorithm is up to the implementation depending on the gNB manufacturer/service provider.

Uplink priority handling is done for different channels within a single UE and these logical channels are multiplexed onto the same transport block based on their priorities and configured rates. This is not required when the uplink scheduling grant provides sufficient resources but that rarely happens since radio networks today are highly loaded.

The greedy approach would be to serve logical channels in the order of decreasing priority but this will lead to channel starvation of the lower-priority channels. For example, if every transport block can carry 10 bytes but the logical channels with the two highest priorities can supply 6 bytes and 5 bytes per transport block respectively, then the remaining channels will not get access to the radio and will overflow in the buffer. This is especially critical when a control channel gets starved. Hence, the algorithm used (in both NR and LTE) accounts for channel starvation issues and ensures that the prioritization is not greedy.

Uplink priority handling is done by the Logical Channel Prioritization block in the MAC Layer. As the block diagram below highlights, LCP is only done only on the uplink, and is connected to three logical channels, namely the Common Control Channel (CCCH), Dedicated Control Channel (DCCH), and the Dedicated Traffic Channel (DTCH). Apart from these, LCP also handles control plane data generated within the MAC layer.

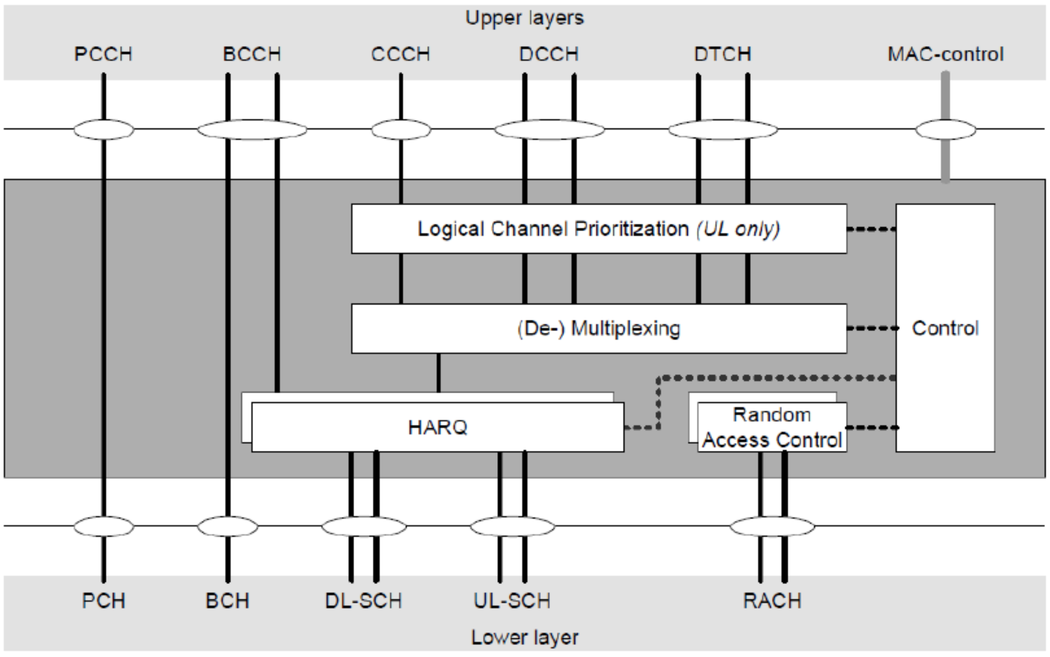


Figure 4.1: Block Diagram of the NR MAC Layer (As per TS 38.321)

4.2 The LCP Algorithm

The priority order for logical channel prioritization is defined by TS 38.321 as follows:

1. C-RNTI MAC CE or any data from UL-CCCH
2. Configured Grant Confirmation MAC CE
3. MAC CE for BSR, with the exception of BSR used for padding
4. Single entry PHR MAC CE or multiple entry PHR MAC CE
5. Data from any logical channel, except data from UL-CCCH
6. MAC CE for recommended bitrate query

7. MAC CE for BSR included for padding

These user and control plane packets are fed to logical channels from DRBs and SRBs. The DRBs will carry data plane packets but they can have different priority values based on the use case. For example, mission critical applications would be given preference over web browsing. This is of special importance to NR since 5G is designed to handle a wide variety of services.

The core LCP algorithm is the same in both LTE and NR, and has two stages. In the first stage, channels are served in the order of priority *upto* a predefined upper limit. This is done to ensure that the lower-priority channels do not get starved. In the second stage, the channels are served in strict priority order until the grant is fully exploited or the buffer is empty.

Each logical channel is assigned the following parameters by RRC in the gNB, which LCP uses to run:

- *priority*: A higher value indicates a lower priority.
- *prioritizedBitRate*: This sets the PBR for each logical channel. The PBR is similar to a guaranteed bit rate in the sense that it specifies a minimum bit rate per channel so that no channel gets starved.
- *bucketSizeDuration*: This sets the BSD for each logical channel, which specifies the duration up to which the channel has to be served at its PBR.

From the above, it can be inferred that:

$$BucketSize_{LCID=j} = PBR_j * BSD_j$$

Where $BucketSize_{LCID=j}$ specifies the maximum size of the bucket that is served at PBR for a logical channel with a Logical Channel ID j.

The UE maintains a counter B_j which is initialized to 0 when the channel is established, and this is incremented by $PBR_j * TTI$ for each TTI till it exceeds the bucket size, after which it gets set to $PBR_j * BSD_j$.

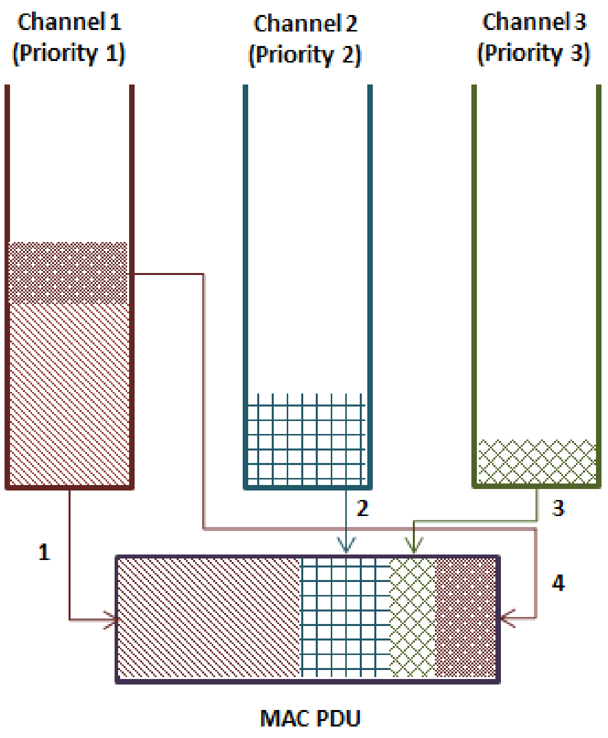


Figure 4.2: Pictorial Representation of the LCP Algorithm

4.3 NR Enhancements

The main LCP algorithm in NR is quite similar in implementation to that in LTE. However, NR is designed to cater to a wide range of services, which brings in a flexibility that was not there in LTE. The transmission durations are not fixed in NR and there are more use cases (URLLC, for example). Thus, a more advanced scheme is needed.

LCP identifies the profiles of different logical channels implicitly from other information available in the grant rather than depending on explicit signaling. It essentially reads the *logicalChannelConfig*, *radioBearerConfig*, QFI (QoS Flow Index), and 5QI (5G QoS Indicator) related structures specified by RRC and RRM.

Once an uplink grant is received, prior to running the main LCP algorithm, the LCP module determines which logical channels can transmit during the given grant. Thus, in NR, logical channel restrictions are introduced in TS 38.321 as follows:

- *allowedSCS-List*: Specifies the allowed subcarrier spacing indices for the given uplink grant (if configured)
- *maxPUSCH-Duration*: Sets a value greater than or equal to the maximum PUSCH duration of the UL grant (if configured)
- *configuredGrantType1Allowed*: Sets whether the UL grant is a Configured Grant

Type 1 (if configured)

- *allowedServingCells*: Includes the cell information associated to the UL grant (if configured). It does not apply to logical channels associated with a DRB with PDCP duplication but is deactivated.

All of these lists are configured using uplink transmission information received from the lower layers (using L1/L2 control signalling) for the scheduled UL transmission.

The motivation behind introducing the PUSCH restriction is to control whether latency-critical data should be allowed to exploit a grant intended for less time-critical data. This is especially important when both URLLC and eMBB/MMTC channels are operating through the same LCP module. In this case, LTE would by default set the maximum PUSCH duration to that corresponding to the lower time-critical data before multiplexing both data packets to the same transport block. Thus, the URLLC packets undergo an unnecessary delay.

The subcarrier spacing restriction has also been introduced for similar reasons. A lower subcarrier spacing implies a longer slot duration which could cause larger delays.

Uplink carrier restrictions will help take channel conditions and diversity requirements into account, which can help improve reliability and provide better coverage (where lower frequencies are used).

The priorities of different channels are up to implementation, but the overall priorities of the services based on their QoS requirements has been recommended by 3GPP in 23.501 (Refer to Table 2.1). These priority values can also be used to configure the LCP module, and can be set by RRC in `LogicalChannelConfig`, a configuration structure that is used to create logical channels. The GBRs specified can also be used to derive the PBR needed for the service.

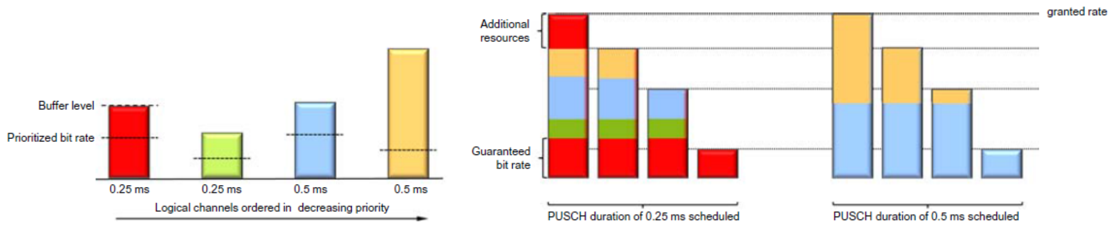


Figure 4.3: An example of LCP for different data rates and PUSCH durations (Dahlman et al)

4.4 Simulator Implementation

This section serves as a documentation for the LCP simulator. Built in the Python 2 (v2.7.12) environment, it predominantly uses `simpy` (v3.0.11), a process-based discrete event simulation framework. In addition, the simulator also uses `numpy` (v1.16.1), `pylab`, and `matplotlib`.

A configuration file is defined in the following format:

```
Number_of_Channels
PBR_List
BSD_List
TB_Size
Input_Data_Rate (WIP)
```

Once the LCP module is configured, it calculates $PBR_j * BSD_j$ for each channel j and then starts the simulation.

The following simpy elements are used in running the simulation:

- `Environment`: This manages the time and scheduling of every process in the simulation. The LCP simulator has two environments - one for the first round with PBR, and the other for the second round with strict priority order.
- `Container`: Helps in modelling the production and consumption of homogeneous, undifferentiated bulk, which in this case are the incoming data packets (to which MAC subheaders have been added).
- `PriorityResource`: Part of the `Resource` class, the `PriorityResource` is a container with a fixed capacity that assigns a priority to each token request. In the LCP case, each logical channel is assigned a priority.

```
class Packet:
    def __init__(self, env, packet_size):
        self.lchrequest = simpy.PriorityResource(env, capacity =1)
        self.rem_size = simpy.Container(env, init = packet_size,
        capacity = packet_size)
```

The PDU structure is defined as a `PriorityResource`, and the remaining PDU size is defined as a `Container`. Both the runs are simulated using two environments.

Each channel in the first round is served at the PBR, either till BSD, till the channel is empty, or till the transport block is full. In the second round, each channel is served at the incoming data rate in strict priority order either till the channel is empty or till the transport block is full.

```
Input: Logical channels $j$, priority $p_j$, prioritized bit rate $PBR_j$, bucket size
        duration $BSD_j$, transport block size $TBSize$, incoming data rate
        $DataRate_j$
Output: Outgoing transport block (to lower layers)
ReadConfigFile();
for j $\in$ $LogicalChannelList$ do
    $BucketSize_j = PBR_j * BSD_j$;
end
Initialize LCH.PriorityResource();
Initialize LCH.Container();
/* In order of decreasing priority                                    */
for j in LogicalChannelList do
    AddToPDU(min($DataRate_j * BSD_j$, $BucketSize_j$));
    if RemTBSize == 0 then
        break;
    end
end
if RemTBSize > 0 then
    for j $\in$ $LogicalChannelList$ do
        AddToPDU(min($DataRate_j * RemTimeinTTI$, RemTBSize));
    end
end
```

Algorithm 2: Logical Channel Prioritization Simulator in SimPy

4.5 Simulation Results

The simulator was run for several cases which have real world analogues. The following cases were simulated.

4.5.1 Case 1: Normal Case

No. of Channels = 8, *TB Size* = 100

LCH in Priority Order	Configured PBR	Configured BSD (s)	Input Bit Rate
1	100	0.05	80
2	50	0.05	80
3	50	0.05	80
4	50	0.05	80
5	50	0.05	80
6	50	0.05	80
7	50	0.05	80
8	50	0.05	80

Table 4.1: LCP Simulation Case 1 Configuration

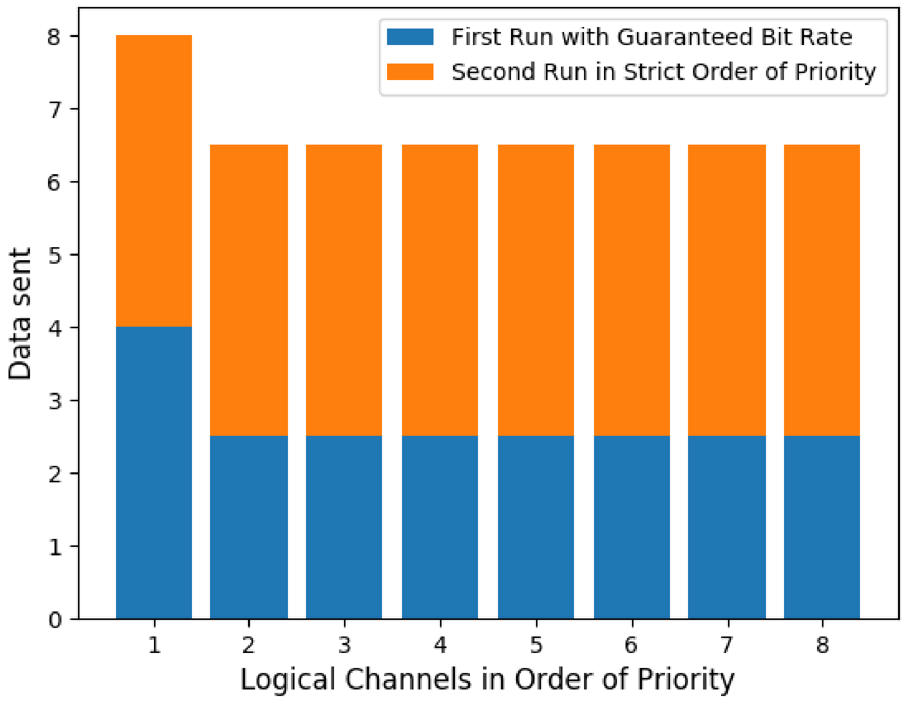


Figure 4.4: LCP Simulation Case 1: Data sent from each logical channel

This is a straightforward LCP iteration where the first channel has a higher priority as well as a higher configured PBR. Hence, it sends more data in the first round. The PDU is also large enough to accommodate the data from all channels in the second round so all packets pass through in this case.

4.5.2 Case 2: TB Full

No. of Channels = 8, *TB Size* = 100

LCH in Priority Order	Configured PBR	Configured BSD (s)	Input Bit Rate
1	100	0.05	500
2	50	0.05	500
3	50	0.05	500
4	50	0.05	500
5	50	0.05	500
6	50	0.05	500
7	50	0.05	500
8	50	0.05	500

Table 4.2: LCP Simulation Case 2 Configuration

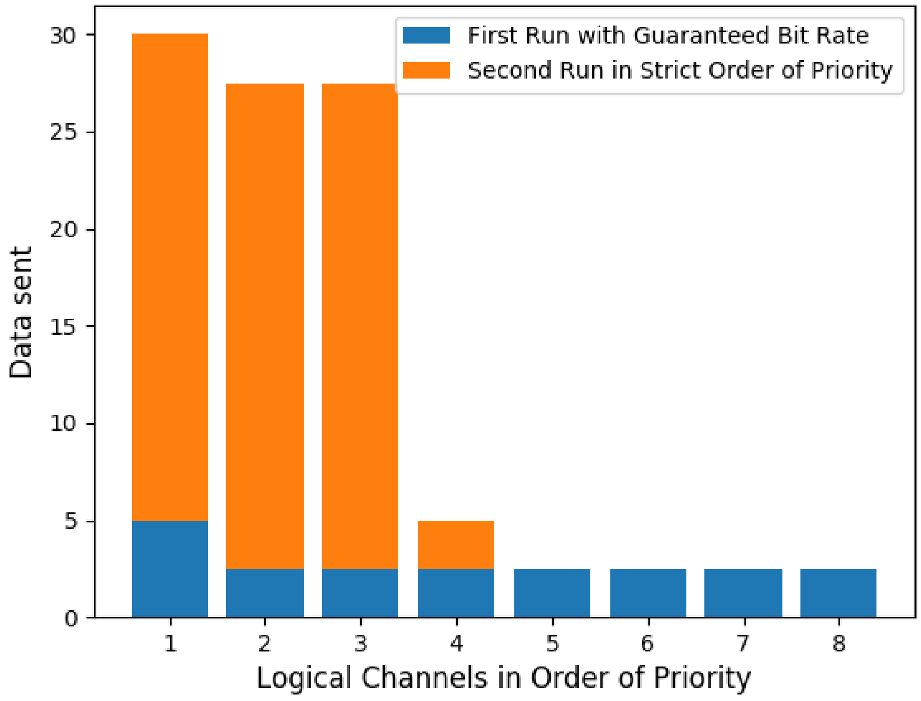


Figure 4.5: LCP Simulation Case 2: Data sent from each logical channel

Here, the input data rate is significantly higher than the PBR, and the cumulative data rate to the LCP block is higher than the TB size. The channel with the highest priority also has a higher PBR so it sends more data in the first round. In the second round when the strict priority order is followed, the first three channels are served fully but the fourth channel is only served partially since the transport block is full. Though the remaining channels get starved in the second round, the PBR in the first round ensures a guaranteed bit rate to ensure that they do not suffer from complete starvation.

4.5.3 Case 3: Infinite PBR

No. of Channels = 8, *TB Size* = 100

LCH in Priority Order	Configured PBR	Configured BSD (s)	Input Bit Rate
1	50	0.05	500
2	50	0.05	500
3	10000	0.05	500
4	50	0.05	500
5	50	0.05	500
6	50	0.05	500
7	50	0.05	500
8	50	0.05	500

Table 4.3: LCP Simulation Case 3 Configuration

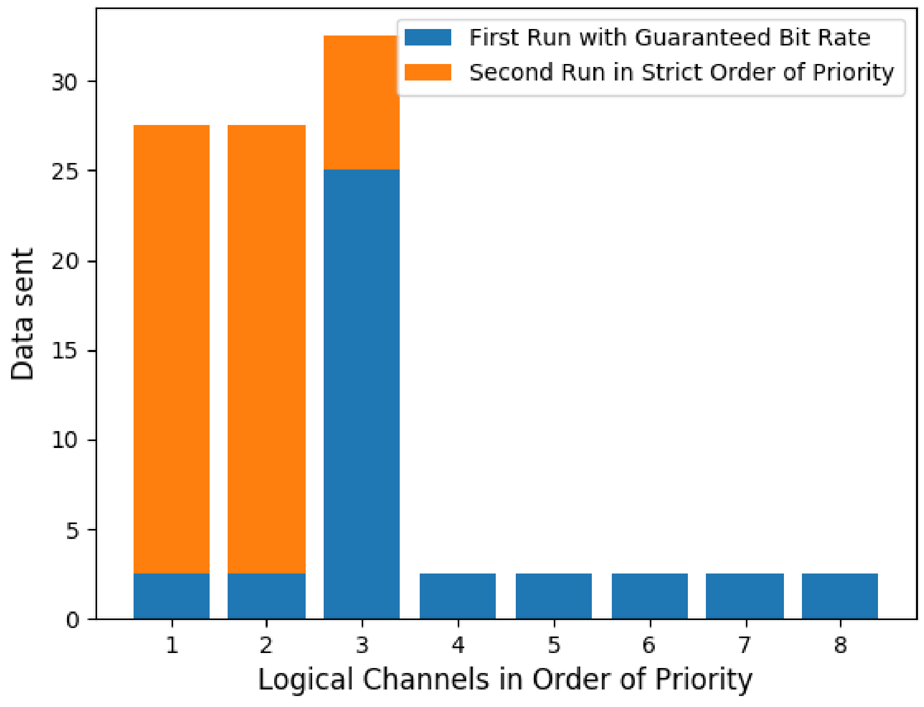


Figure 4.6: LCP Simulation Case 3: Data sent from each logical channel

Here, the channel with the third priority is assigned an infinite PBR (10000»50 so it can be assumed as infinite), so it gets the highest access to the transport block. However, it is cut short in the second round as the strict order of priority causes starvation. Channels with infinite PBR are used commonly, primarily for control information through SRBs and for URLLC packets. Both are discussed in the following subsections.

4.5.4 Case 4: Typical DRB/SRB Configuration

No. of Channels = 8, *TB Size* = 100

LCH in Priority Order	Configured PBR	Configured BSD (s)	Input Bit Rate
1	10000	0.05	100
2	10000	0.05	100
3	10000	0.05	100
4	50	0.05	500
5	50	0.05	500
6	50	0.05	500
7	50	0.05	500
8	50	0.05	500

Table 4.4: LCP Simulation Case 4 Configuration

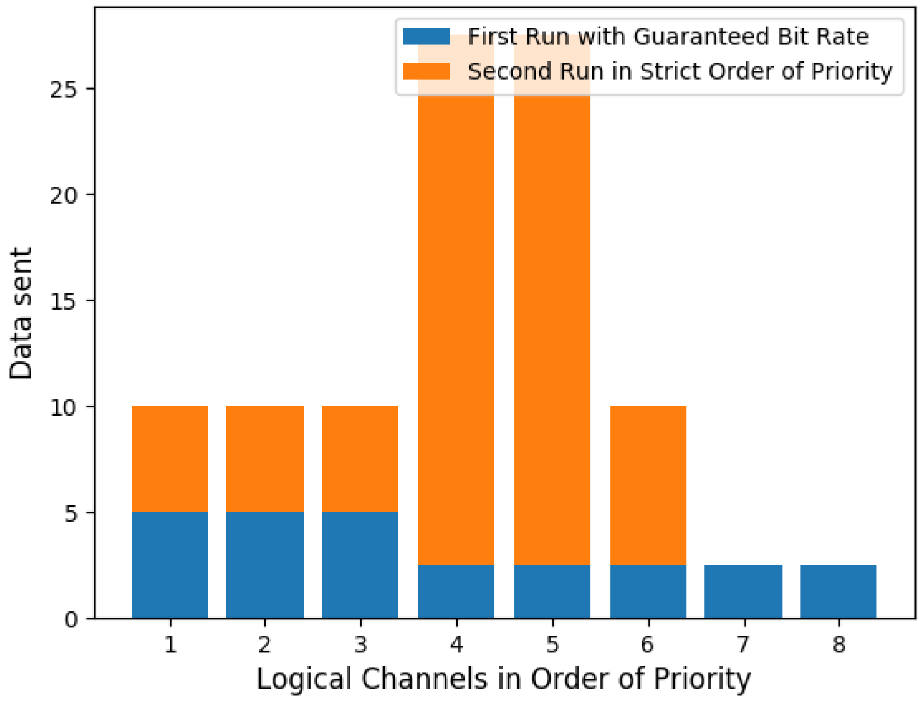
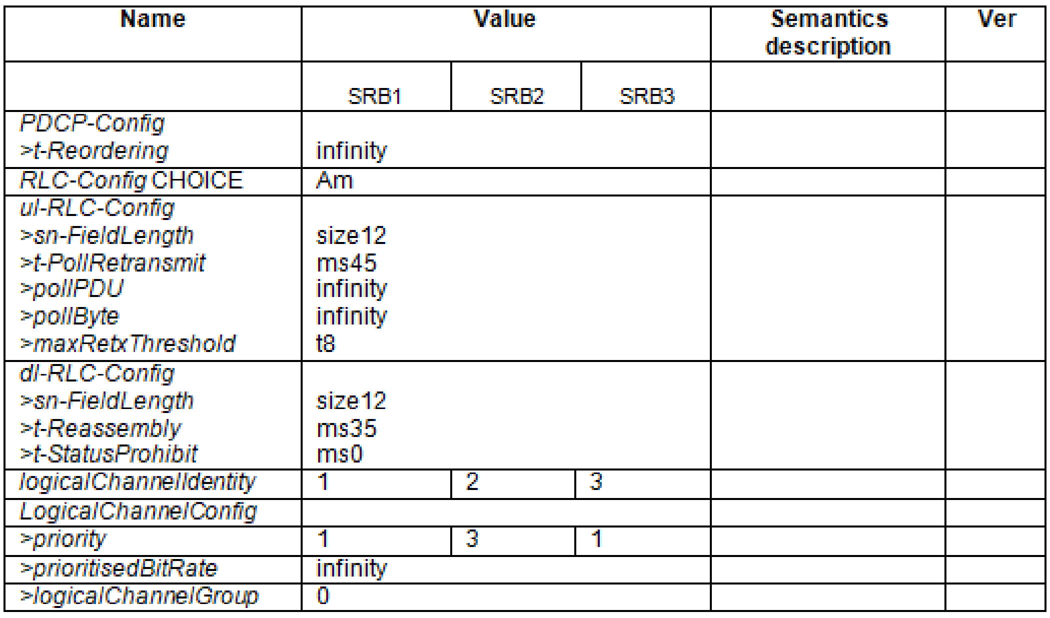


Figure 4.7: LCP Simulation Case 4: Data sent from each logical channel

This simulation reflects a typical DRB/SRB case. The SRBs, namely SRB1, SRB2, and SRB3, carry control information so it is imperative that the logical channels which correspond to them do not get starved. Hence, they are all configured to have an infinite PBR by default, as specified in the RRC standard (3GPP TS 38.331 V15.4.0 Section 9.2.1).

As the table below indicates, SRB1 and SRB3 have the joint highest priorities and all three SRBs have an infinite PBR by default. The BSD is up to implementation but generally will not affect anything since the PBR overshadows it. However, the data rates of control information are not as high as data plane channels which is why channels 4,5, and 6 send more data to the transport block in the second round.



Name	Value			Semantics description	Ver
	SRB1	SRB2	SRB3		
PDCP-Config *>t-Reordering*	 infinity				
RLC-Config CHOICE	Am				
ul-RLC-Config *>sn-FieldLength* *>t-PollRetransmit* *>pollPDU* *>pollByte* *>maxRetxThreshold*	 size12 ms45 infinity infinity t8				
dl-RLC-Config *>sn-FieldLength* *>t-Reassembly* *>t-StatusProhibit*	 size12 ms35 ms0				
logicalChannelIdentity	1	2	3		
LogicalChannelConfig					
>priority	1	3	1		
>prioritisedBitRate	infinity				
>logicalChannelGroup	0				

Figure 4.8: Default SRB Configurations by RRC

4.5.5 Case 5: DRB/SRB Configuration with Mission-Critial Service

No. of Channels = 8, *TB Size* = 100

LCH in Priority Order	Configured PBR	Configured BSD (s)	Input Bit Rate
1	10000	0.05	100
2	100000	0.05	200
3	10000	0.05	100
4	10000	0.05	100
5	50	0.05	500
6	50	0.05	500
7	50	0.05	500
8	50	0.05	500

Table 4.5: LCP Simulation Case 5 Configuration

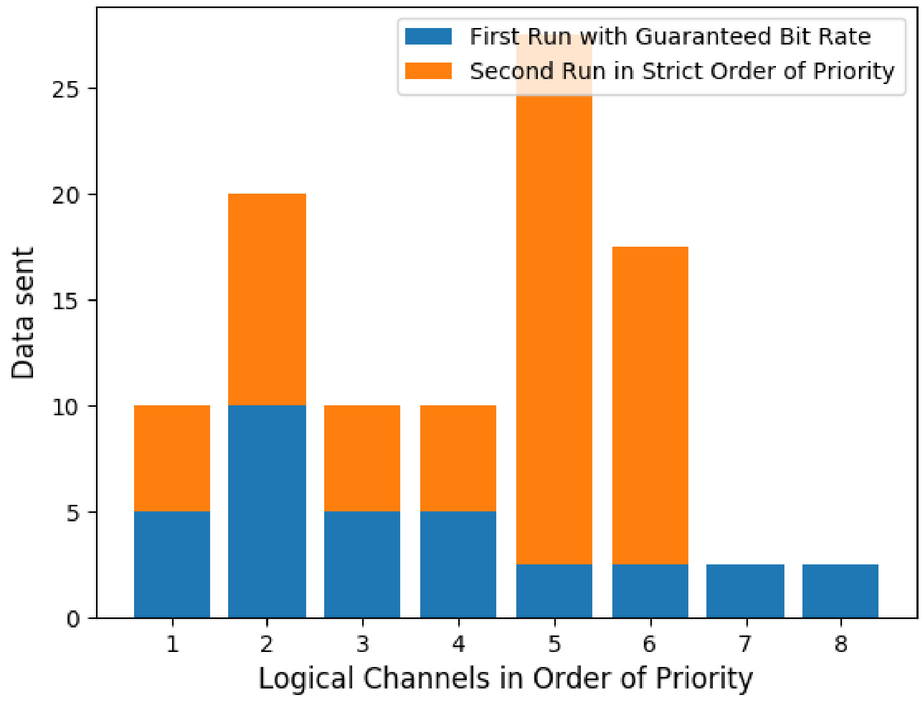


Figure 4.9: LCP Simulation Case 5: Data sent from each logical channel

This case shows introduces a DRB catering to a mission-critical service. It is still under discussions in the 3GPP RAN2 Working Group, but one recommendation is to prioritize some mission-critical cases over some SRBs (details are in the following sections). For this case, the second channel is chosen as a URLLC DRB with a "higher infinity" as its PBR as well as a higher data rate than an SRB. More about this will be discussed in the Further Work section.

4.6 Suggested Enhancements and Further Work

The above algorithm can be ported to the Testbed DPDK framework once multiple channels have been set up, along with RRC to configure them. Each channel can be modeled as a ring buffer and a hash can be used to store configuration details, and ngPkt manipulation functions discussed in prior sections can be used to join and truncate packets to fit them inside a transport block.

From a standardization perspective, several enhancements can be made to Logical Channel Prioritization and this is being discussed in the 3GPP working groups for Release 16, specifically for URLLC and IoT use cases. Some of the suggested enhancements are follow-ups of the simulations and restrictions described above, and are explained below.

TDoc R2-1800520 provides recommendations on handling infinite PBRs especially in the case of URLLC channels. Simulation Case 5 shows how an infinite PBR would affect the algorithm, but a major issue arises when there are multiple URLLC channels, particularly when all are configured with an infinite PBR. A repercussion of this approach is that all URLLC channels split the grant resources equally irrespective of their PDBs or arrival rates, which could lead to more latencies for some channels. The TDoc recommends that the "time to expiry" (SDU Arrival Time + Delay Budget) could be used as a parameter while prioritizing, but other approaches (such as those used by traditional TSNs) could be taken too.

TDoc R2-1817077 discusses the LCP channel priority order when URLLC channels are also taken into account. Since some URLLC use cases could be more mission-critical than sending some SRBs, there should be a new prioritization scheme for URLLC that allows some channels to be prioritized over some SRBs. This has also been shown in LCP Simulation Case 5.

TDoc R2-1904055 (which is in discussion currently) brings up the issue of prioritization between multiple configured grants and dynamic grants. Configured grants (Type 1) are given so that multiple devices can use periodic resources instead of them being wasted. This could cause an issue in the LCP module when different services are being sent through different configured grants. Moreover, even if prioritization is done using the priority values, preemption is an option that should be considered.

3GPP TR 38.825: Study on NR Industrial Internet of Things (IoT) (V1.0.0, March 2019) discusses the above issue as well. Release 15 states that dynamic grants always override configured grants in case of a resource conflict but this is not desirable when the configured grant is given to a URLLC service. The study also discusses control information collisions and the applications of Time Sensitive Networking (TSN) in 5G IoT, all of which are areas being looked into currently.

REFERENCES

[1] NR and NG-RAN Overall Description; Stage 2 (TS 38.300, V15.5.0), April 2019

[2] Radio Link Control (RLC) Protocol Specification (TS 38.322 V15.5.0), April 2019

[3] Medium Access Control (MAC) Protocol Specification (TS 38.321 V15.5.0), April 2019

[4] Radio Resource Control (RRC) Protocol Specification (TS 38.331 V15.5.0), April 2019

[5] System Architecture for the 5G System (TS 23.501 V16.0.2), April 2019

[6] Study on NR Industrial Internet of Things (IoT) (TR 38.825 V16.0.0), April 2019

[7] Erik Dahlman, Stefan Parkvall, Johan Sköld. 5G NR: The Next Generation Wireless Access Technology. 1st Edition, August 2018

[8] Data Plane Development Kit V19.02 Documentation,
`https://doc.dpdk.org/guides-19.02/`

[9] SimPy V3.0.11 Documentation,
`https://simpy.readthedocs.io/en/latest/`

[10] TDoc R2-1800520: LCP Enhancements for URLLC, January 2018

[11] TDoc R2-1817077: UL Logical Channel Prioritization and Multiplexing, November 2018

[12] TDoc R2-1904055: On Grant Prioritization Involving Configured Grants, April 2019