

# **Building 5G L2/L3 Protocol Stack**

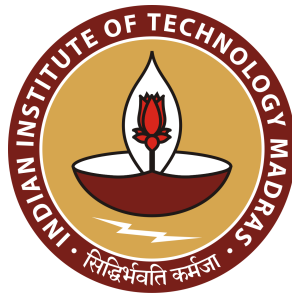
*A Project Report*

*submitted by*

**TEJARAM SANGADI**

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

**BACHELOR OF TECHNOLOGY**



**DEPARTMENT OF ELECTRICAL ENGINEERING  
INDIAN INSTITUTE OF TECHNOLOGY MADRAS**

**JUL 2020**

# **CERTIFICATE**

This is to certify that the thesis (or project report) titled **Building 5G L2/L3 Protocol Stack**, submitted by **Tejaram Sangadi**, to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology**, is a bona fide record of the project work done by him (her) under my (our) supervision. The contents of this thesis (or project report), in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

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

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

Place: Chennai

Date: 24<sup>th</sup> July 2020

## ACKNOWLEDGEMENTS

I would like to express my gratitude to **Prof. Krishna Jagannathan** and **Prof. Radha Krishna Ganti** for giving me the opportunity to contribute to this challenging project. Their invaluable guidance has helped significantly while doing this project.

I am also grateful to the 5G Protocol Stack Development team, IIT Madras for their continued support during the project period.

Finally, I would like to thank my parents who have been supportive throughout the journey.

# **ABSTRACT**

**KEYWORDS:** 4G LTE ; 5G NR; L2/L3 Protocol Stack; MAC; ASN1;

Looking at the trends of evolution of cellular technology from 1G to 4G, there have been significant improvements. These improvements have been the backbone for globalisation. 4G LTE technology has revolutionised the usage of smartphones by connecting people with food aggregators, cab-hailing services, payment platforms and many more services which have opened the potential for a new market. The problem faced now is congestion of the networks due to heavy bandwidth requirement and subscriber demand. The next cellular technology has been architected to solve the problems of the existing technology and offers many more features. 5G NR is the latest cellular technology which unlocks the key to the advent of exhaustive applications. This has been possible due to key improvements in existing 4G cellular technology (LTE). This work focuses on understanding what goes into building next-generation cellular technology, design of the MAC module and the usage of ASN1 in the 5G NR Protocol L2/L3 protocol stack.

# TABLE OF CONTENTS

	Page
<b>ACKNOWLEDGEMENTS</b> . . . . .	i
<b>ABSTRACT</b> . . . . .	ii
<b>LIST OF FIGURES</b> . . . . .	v
<b>GLOSSARY</b> . . . . .	vi
<b>ABBREVIATIONS</b> . . . . .	vii
<b>CHAPTER 1: INTRODUCTION</b> . . . . .	1
1.1 Why 5G? . . . . .	1
1.2 What is 5G? . . . . .	2
1.3 How is 5G Developed? . . . . .	4
<b>CHAPTER 2: Literature Review</b> . . . . .	5
2.1 Open Solutions . . . . .	5
2.2 ORAN Architecture . . . . .	6
<b>CHAPTER 3: MAC Module</b> . . . . .	9
3.1 MAC Functionalities . . . . .	9
3.2 Code Architecture . . . . .	10
<b>CHAPTER 4: ASN.1</b> . . . . .	12
4.1 Introduction . . . . .	12
4.2 Encoding rules . . . . .	12
4.3 ASN1 compilers . . . . .	13
4.3.1 Obj-sys compiler . . . . .	14
4.3.2 OSS Noklava compiler . . . . .	14

<b>CHAPTER 5: Discussion of Findings . . . . .</b>	<b>16</b>
5.1 MAC Execution Time . . . . .	16
5.2 F1-Setup Encoding & Decoding . . . . .	18
5.2.1 Introduction . . . . .	18
5.2.2 Procedure . . . . .	18
5.2.3 Decoded Data . . . . .	20
5.2.4 Conclusion . . . . .	23
<b>APPENDIX A: ASN1C Open Source Compiler v9.29 . . . . .</b>	<b>26</b>
A.1 Installation Guide . . . . .	26
<b>APPENDIX B: Extracting ASN1 From 3GPP Technical Specifications. . .</b>	<b>28</b>
B.1 Procedure from Specification . . . . .	28
B.2 Steps for Extraction . . . . .	28
B.3 Working of the Python Script . . . . .	29
<b>APPENDIX C: Socket Programming in C/C++. . . . .</b>	<b>32</b>
C.1 Introduction . . . . .	32
C.2 API's for server . . . . .	33
C.2.1 Socket creation . . . . .	33
C.2.2 setsockopt . . . . .	33
C.2.3 Bind . . . . .	33
C.2.4 Listen . . . . .	34
C.2.5 Accept . . . . .	34
C.3 API's for client . . . . .	34
C.3.1 Socket creation . . . . .	34
C.3.2 Connect . . . . .	34
<b>REFERENCES . . . . .</b>	<b>37</b>

# LIST OF FIGURES

Figure	Title	Page
1.1	Requirements for 5G Networks, Wikipedia (2020) . . . . .	3
1.2	5G Development Process . . . . .	4
2.1	Interfaces in 5G, Dahlman <i>et al.</i> (2018) . . . . .	6
2.2	L2/L3 O-RAN architecture for gNB, O_RAN <sup>®</sup> (2019) . . . . .	7
2.3	O-CU Block and its Internal APIs, O_RAN <sup>®</sup> (2019) . . . . .	8
2.4	O-DU Block and Internal APIs, O_RAN <sup>®</sup> (2019) . . . . .	8
3.1	MAC Functionalities . . . . .	9
3.2	MAC-Scheduler Interface, O_RAN <sup>®</sup> (2019) . . . . .	10
3.3	Threaded Implementation . . . . .	11
4.1	Encoding rules, Walkin (2010) . . . . .	12
4.2	Comparison of different encoding rules, Walkin (2010) . . . . .	13
4.3	ASN1C Workflow, objective SYSTEMS <sup>®</sup> (2020) . . . . .	13
4.4	ASN1C Studio. . . . .	15
5.1	Timing for Thread 1 and Thread 2 . . . . .	17
5.2	F1-Setup Request. . . . .	18
5.3	F1-Setup Request on ASN1C Studio. . . . .	19
5.4	Encoded Data in Hex. . . . .	19
5.5	ASN1C Open Source Compiler Support, Walkin (2010) . . . . .	24
5.6	RRC ASN1 Encoding Scheme: UPER, 3GPP (2020c) . . . . .	24
5.7	F1AP ASN1 Encoding Scheme: APER, 3GPP (2020a) . . . . .	24
5.8	NGAP ASN1 Encoding Scheme: APER, 3GPP (2020b) . . . . .	25
C.1	State diagram for server and client model. . . . .	32

## GLOSSARY

The following are some of the commonly used terms in this report:

<b>Protocol Stack</b>	Conceptual description of the layered structure of communication protocols in which layers are depicted in hierarchical form. Every layer in the protocol stack is capable of doing a specific functionality and can have a mode of operation. Examples of protocol stacks: OSI (Open System Interconnection) model and the TCP/IP protocol stack.
<b>Software Defined Networking</b>	SDN defines the process of abstracting and concentrating many of the control actions related to packet forwarding and handling processes which would typically take place within an individual network router or switch.
<b>gNB</b>	The gNB Next Generation base station which supports the 5G NR architecture.
<b>API</b>	A set of methods, protocols, and tools used while developing software.
<b>SDU</b>	The Service Data Unit identifies the message transferred between peer layer entities but not understood by the supporting lower-layer entities.
<b>PDU</b>	A Protocol Data Unit contains data and control information which is transferred between layers in a protocol stack. Mpirical (2020)



## **ABBREVIATIONS**

NR	New Radio
LTE	Long Term Evolution
MAC	Medium Access Control
UE	User Equipment
RAN	Radio Access Network
SDAP	Service Data Adaptation Protocol
PDCP	Packet Data Convergence Protocol
RLC	Radio Link Control
MAC	Media Access Control
RRC	Radio Resource Control
DL	Downlink
UL	Uplink
SDU	Service Data Unit
PDU	Protocol Data Unit
HARQ	Hybrid Automated Repeat Request
TB	Transport Block
gNB	5G Base Station
ASN1	Abstract Syntax Notation 1
BCCH	Broadcast Control Channel
PCCH	Paging Control Channel
CCCH	Common Control Channel

DCCH	Dedicated Control Channel
DTCH	Dedicated Traffic Channel
BCH	Broadcast Channel
PCH	Paging Channel
DL-SCH	Downlink-Shared Channel
UL-SCH	Uplink-Shared Channel
RACH	Random Access Channel
CBG	Code Block Group

# CHAPTER 1

## INTRODUCTION

### 1.1 Why 5G?

The immense growth in the number and type of connected devices and the abundant increment in user/network traffic volume and types, upon the performance constraints in 4G technologies, have driven industry efforts and investments towards defining, developing and deploying systems for the fifth-generation (5G) of mobile networks. The 5th generation of mobile broadband wireless networks have been designed to meet the challenging system and service requirements of the existing and emerging applications in 2020 and beyond.

The future connected societies are portrayed by the vital growth in connectivity and traffic density, network densification, and the broad range of new use cases and applications.

As a consequence, there is a constant need to push the performance envelope of the wireless systems to the new frontiers in order to satisfy the requirements for larger capabilities through virtualized and software-defined network (SDN) architectures. Ahmadi (2019)

## 1.2 What is 5G?

ITU defines 5G in IMT-2020 (vision document for 5G Networks by ITU) and has defined three use cases described below briefly:

1. Provides 100x faster user-experienced data rate than LTE: enhanced Mobile Broadband (eMBB):
2. Has an end-to-end system latency of 1ms: Ultra-Reliable Low-Latency Communications (URLLC)
3. Can handle 1 million devices/sq.km: massive Machine-Type Communications (mMTC).

The eMBB use case is fundamentally based on the progression and improvement of LTE technology. Ahmadi (2019)

The below picture depicts all the requirements for 5G networks:


Capability	Description	5G requirement	Usage scenario
Downlink peak data rate	Minimum maximum data rate technology must support	20 Gbit/s	eMBB
Uplink peak data rate		10 Gbit/s	eMBB
User experienced downlink data rate	Data rate in dense urban test environment 95% of time	100 Mbit/s	eMBB
User experienced uplink data rate		50 Mbit/s	eMBB
Latency	Radio network contribution to packet travel time	4 ms	eMBB 
		1 ms	URLLC
Mobility	Maximum speed for handoff and QoS requirements	500 km/h	eMBB/URLLC
Connection density	Total number of devices per unit area	$10^6/\text{km}^2$	mMTC
Energy efficiency	Data sent/received per unit energy consumption (by device or network)	Equal to 4G	eMBB
Area traffic capacity	Total traffic across coverage area	10 Mbps/m <sup>2</sup>	eMBB
Peak downlink spectrum efficiency	Throughput per unit wireless bandwidth and per network cell	30 bit/s/Hz	eMBB

Fig. 1.1: Requirements for 5G Networks, Wikipedia (2020)

## 1.3 How is 5G Developed?

3GPP, an industry body has many working groups which try to reach the goals set by ITU in IMT-2020. The below picture depicts the process of development. The process involves bringing all the major tech giants and research labs together to devise universal technical specifications. These specifications are approved by several standards organisations (ETSI, TSDSI, ...) and published as standards. The open-source community develops solutions in compliance with these standards. Casaccia (2017)

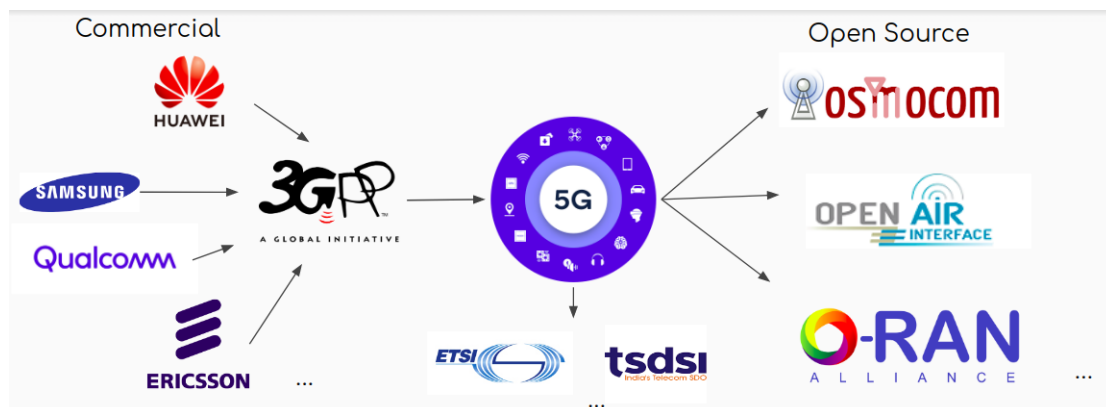


Fig. 1.2: 5G Development Process

# CHAPTER 2

## Literature Review

### 2.1 Open Solutions

3GPP technical specifications emphasise more on the aspects of the UE and implementation-specific details of Basestation are left to be managed by vendors and operators.

This is where O-RAN alliance comes into the picture offering an open solution focusing on bringing white box hardware and open interfaces

O-RAN specifications of Working Group 8 (WG8):Stack Reference Design, offer an open architecture for implementing Basestation L2/L3 Stack.

The purpose of Working Group 8 is to design the software architecture and release plans for the O-CU (O-RAN Central Unit) & O-DU (O-RAN Distributed Unit) building upon the O-RAN and 3GPP specifications for the 5G protocol. stack.O\_RAN<sup>®</sup> (2019)

The architecture of 5G NR consists of three major components:

1. UE: User Equipment (Eg. Mobile Phone)
2. Basestation (gNB)
3. 5G Core Network

These components further consist of different modules. The below picture depicts the 5G network comprising different interfaces between the modules.

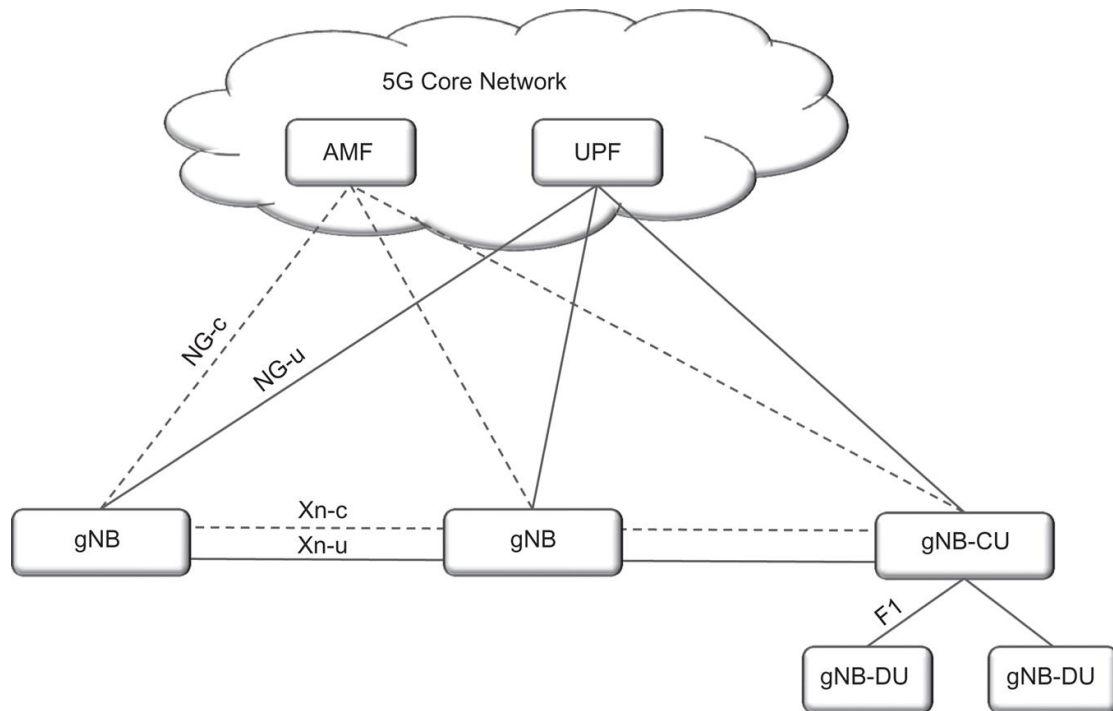


Fig. 2.1: Interfaces in 5G, Dahlman *et al.* (2018)

## 2.2 ORAN Architecture

**gNB**, The Basestation for 5G is further split into two components gNB-CU and gNB-DU.

**gNB-CU**: Central Unit in O-RAN architecture is defined as O-CU.

**gNB-DU**: Distributed Unit in O-RAN architecture is defined as O-DU.

O-RAN in WG8 Specifications has pointed out the API blocks and Call flows for L2/L3 Architecture. The below figures depict the architecture and modules for O-CU and O-DU



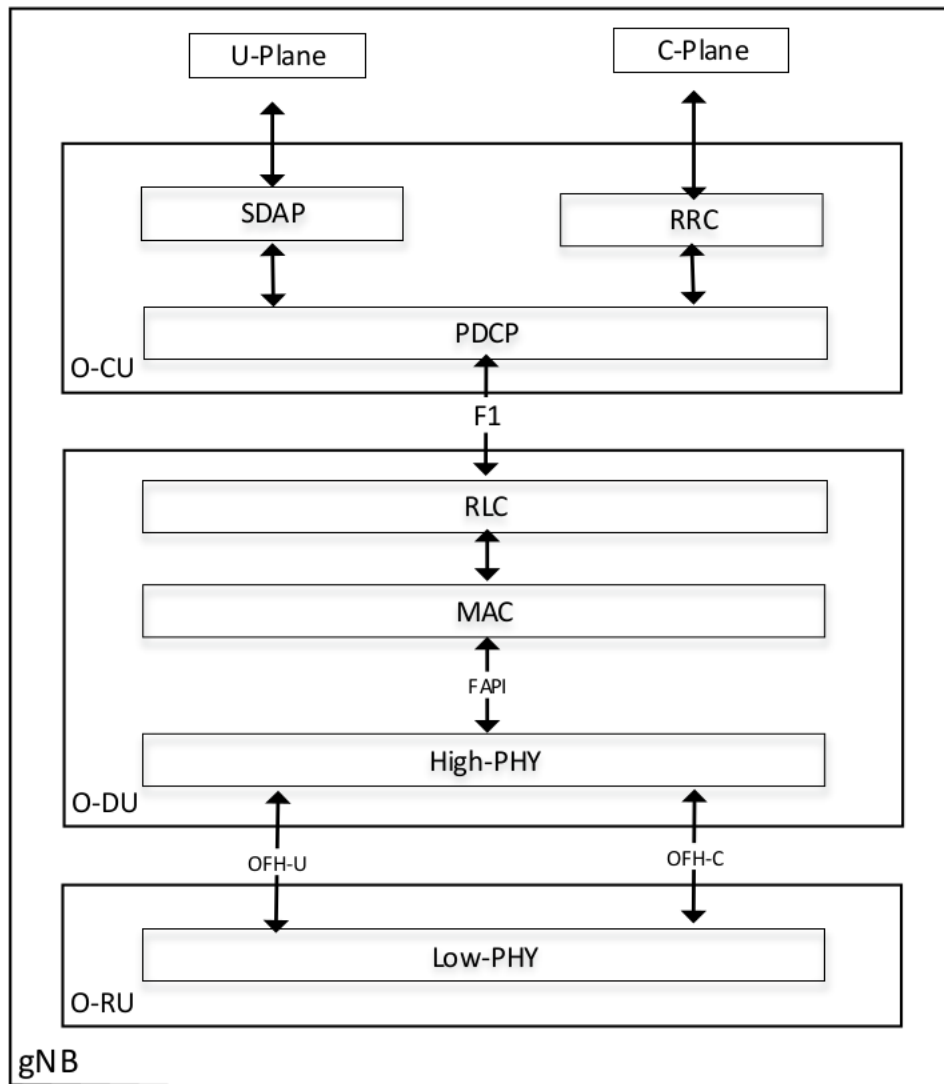


Fig. 2.2: L2/L3 O-RAN architecture for gNB, O\_RAN<sup>®</sup> (2019)

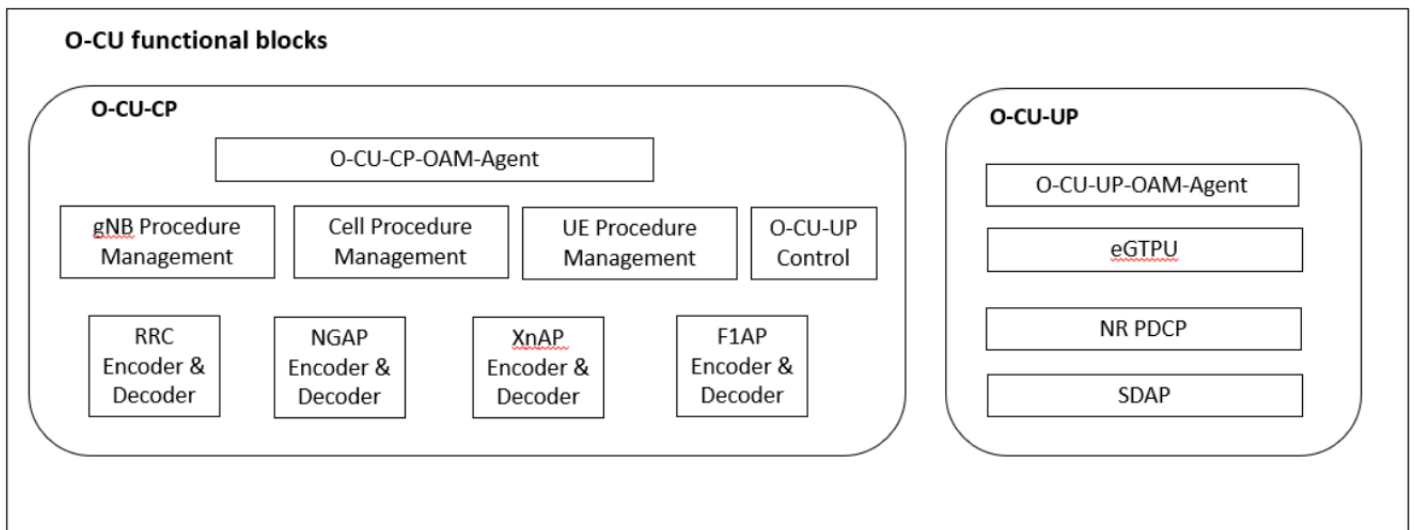


Fig. 2.3: O-CU Block and its Internal APIs, O\_RAN<sup>®</sup> (2019)

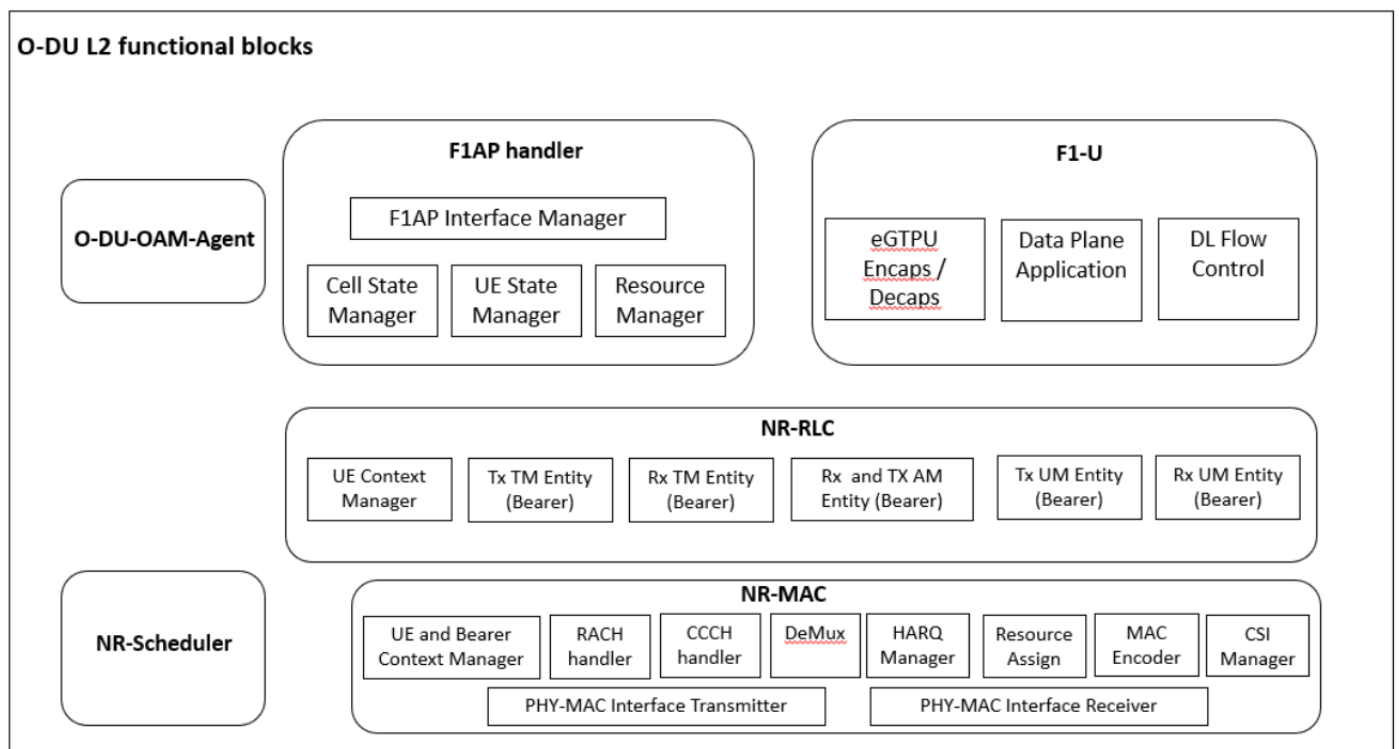


Fig. 2.4: O-DU Block and Internal APIs, O\_RAN<sup>®</sup> (2019)

# CHAPTER 3

## MAC Module

### 3.1 MAC Functionalities

MAC unit is the final module in the L2 stack which consists of many functionalities. Below picture depicts the functionalities present in the MAC unit.

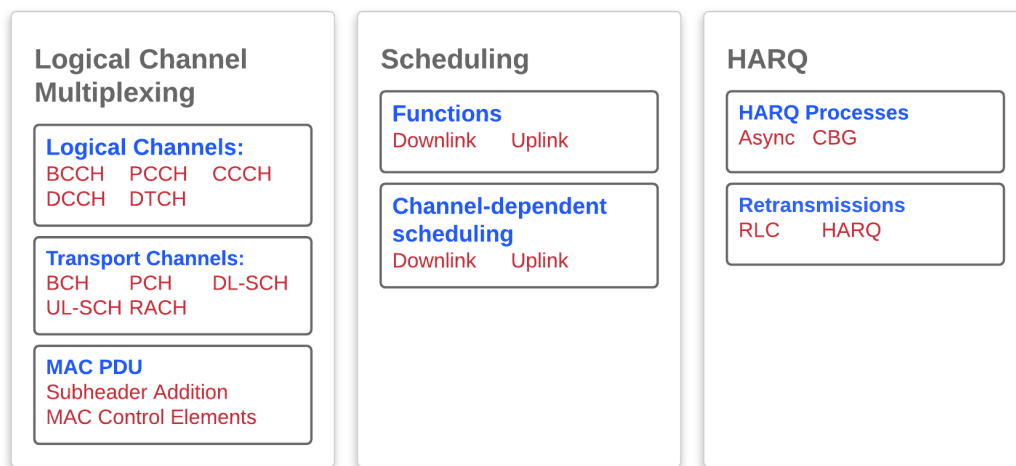


Fig. 3.1: MAC Functionalities

The below architecture from O-RAN depicts the functionalities of MAC module, ie Mux/DeMux and HARQ. The remaining functionalities are taken care by other APIs.

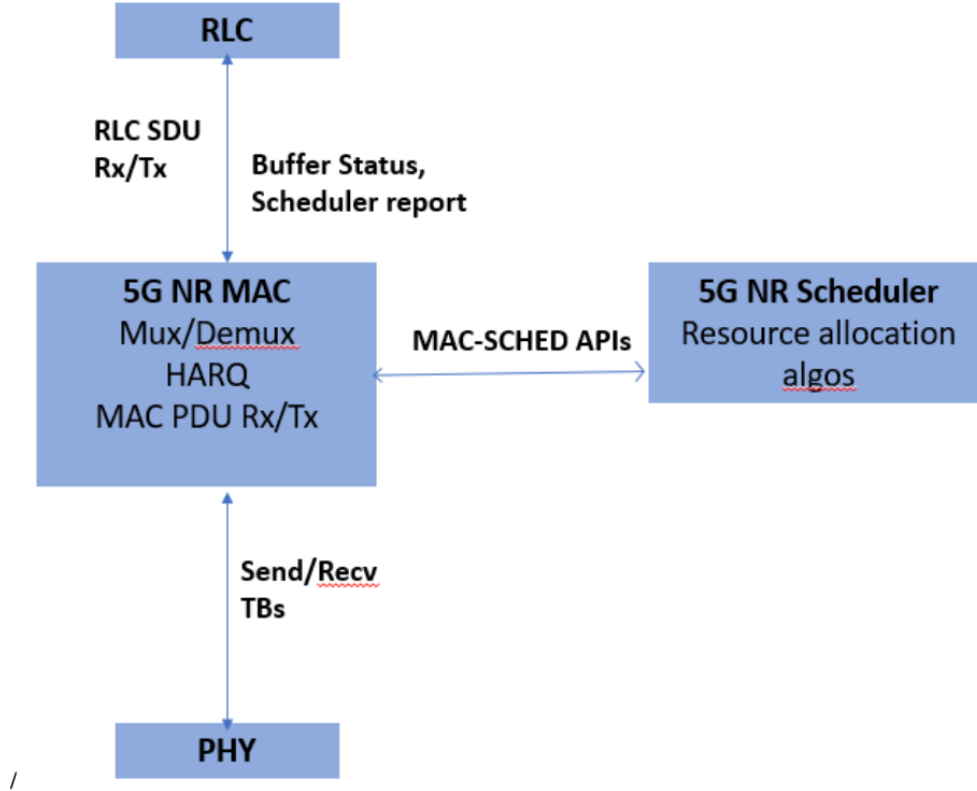


Fig. 3.2: MAC-Scheduler Interface, O\_RAN<sup>®</sup> (2019)

## 3.2 Code Architecture

The test source code follows the model of using threads to implement MAC's functionalities of Multiplexing and Demultiplexing.

A thread is a singular sequence stream in a process. Since threads have some of the features of processes, they are also called lightweight processes. Threads are not independent of each other like processes. Threads share their code section, data section and OS resources (open files and signals) with other threads. Similar to a process, a thread has its own program counter (PC), a register set, and a stack space. Jain (2020)

### Functions of Each Thread at gNB side:

**Thread 1:** Generate data which is to be received from RLC and perform MAC functionalities (Multiplexing and Adding 8/16 bit header)

**Thread 2:** Send the data to PHY if the TB size reaches max\_TB\_size.

### Functions of Each Thread at UE side:

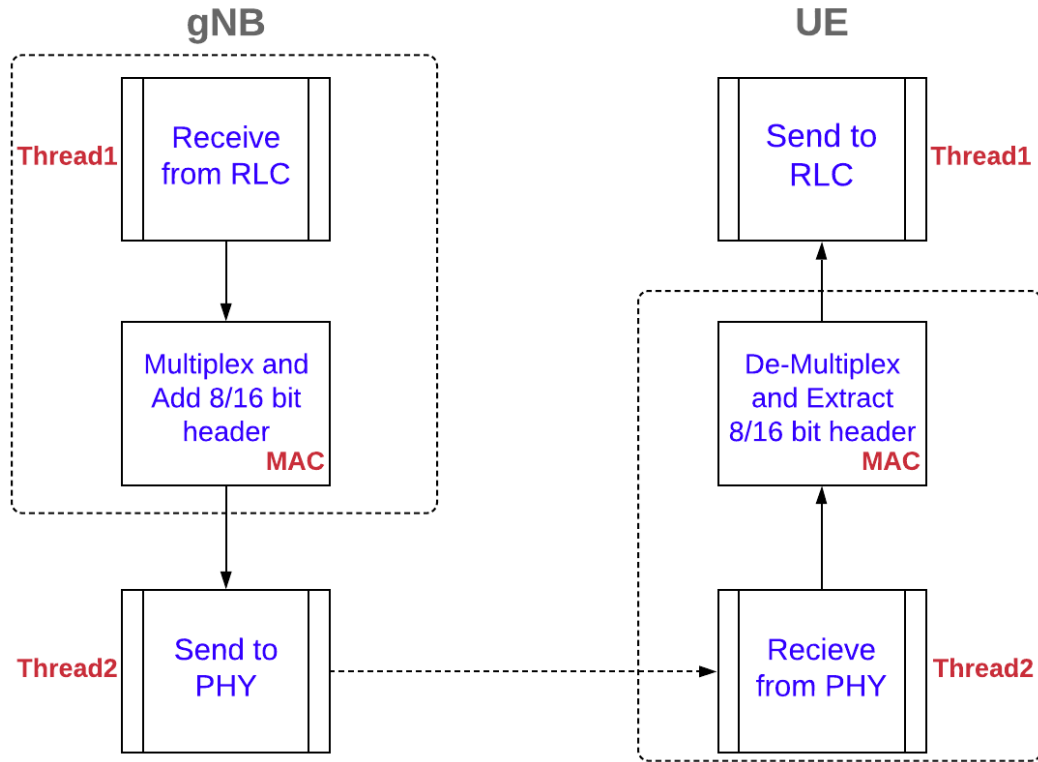


Fig. 3.3: Threaded Implementation

**Thread 1:** Received data after MAC functionalities (De-Multiplexing and Extracting 8/16 bit header) and store it in the RLC Buffer.

**Thread 2:** Send the data to PHY if the TB size reaches max\_TB\_size.

Sending and Receiving from PHY is emulated using socket programming, see C The use of a link loopback server helped to send and to receive data packets in the same system. The source code of MAC module is present in L2 Protocol Stack hosted by the server of 5G Testbed Lab, IITM.

# CHAPTER 4

## ASN.1

### 4.1 Introduction

Abstract Syntax Notation 1 is a representation used for defining data transmitted by telecommunications protocols, unmindful of language implementation and physical description of these data, whatever the application, regardless of the complexity. ITU (2014)

### 4.2 Encoding rules

ASN.1 has a collection of rules exactly specifying how messages need to be "encoded" for communication with another machine. Every set of "encoding rules" has distinct characteristics, like compactness or decoding speed, which make it a good fit for particular environments. All the encoding rules are able to represent any messages we would like to exchange. Walkin (2010)

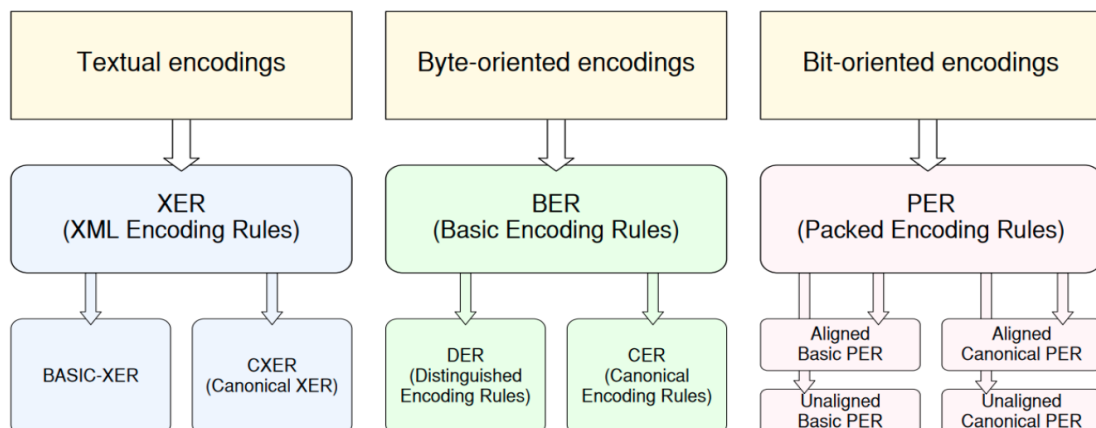


Fig. 4.1: Encoding rules, Walkin (2010)

The ASN.1 standard describes many ways to encode data. A protocol developer can select textual, binary-based or compact bit-packed encoding rules depending on space, interoperability and efficiency requirements.

Encoding variant	Compactness	Interoperability
XER (BASIC or CXER)	Not compact	Human readable UTF-8 XML subset
BER (DER or CER)	Very good	BER decoder can read DER and CER encoded data. Universal debuggers and decoders exist (unber and enber are part of asn1c distribution).
Aligned PER	Nearly best	PER stream decoding can only be done using a corresponding ASN.1 syntax. Unaligned/Aligned variants are incompatible. Basic PER decoder can read Canonical PER encoded data.
Unaligned PER	Best	

Fig. 4.2: Comparison of different encoding rules, Walkin (2010)

## 4.3 ASN1 compilers

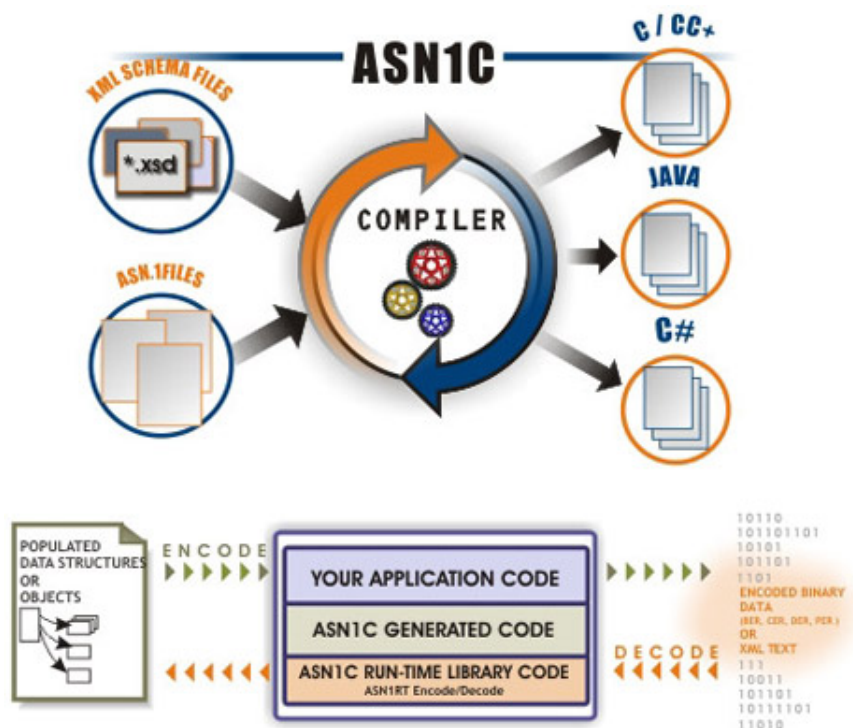


Fig. 4.3: ASN1C Workflow, objective SYSTEMS<sup>©</sup> (2020)

There are many ASN1 compilers available online and can be broadly divided into two categories:

- Commercial: Examples of some notable ones are OSS Nokalva and Object Systems
- Open Source: asn1c compiler on Github written by Lev Walkin.

### **4.3.1 Obj-sys compiler**

The Obj-Sys ASN1C compiler translates ASN.1 and/or XML schema (XSD) source specs into C, C++, C#, Java, or Python source code. Developers can use this code to translate structures/objects to and from finished ASN.1 messages using ITU-T/ISO encoding rules BER, CER, DER, OER, PER, UPER, JER(JSON), or XER(XML). [objective SYSTEMS<sup>©</sup> (2020)]

### **4.3.2 OSS Noklava compiler**

The OSS ASN.1 Tools for C is a comprehensive development toolkit for swiftly developing applications using ASN.1. This product highlights a powerful ASN.1:2015 capable compiler, a runtime library with ASN.1 BER, CER, DER, PER/UPER, CPER/CUPER, OER, COER, XER, CXER, E-XER, and JSON encoder/decoder engines, and a vibrant set of services to simplify and boost the development.[OSS\_Nokalva<sup>©</sup> (2020)]



**ASN1Studio** is a handy tool from OSS Nokalva for building Applications involving usage of ASN1.

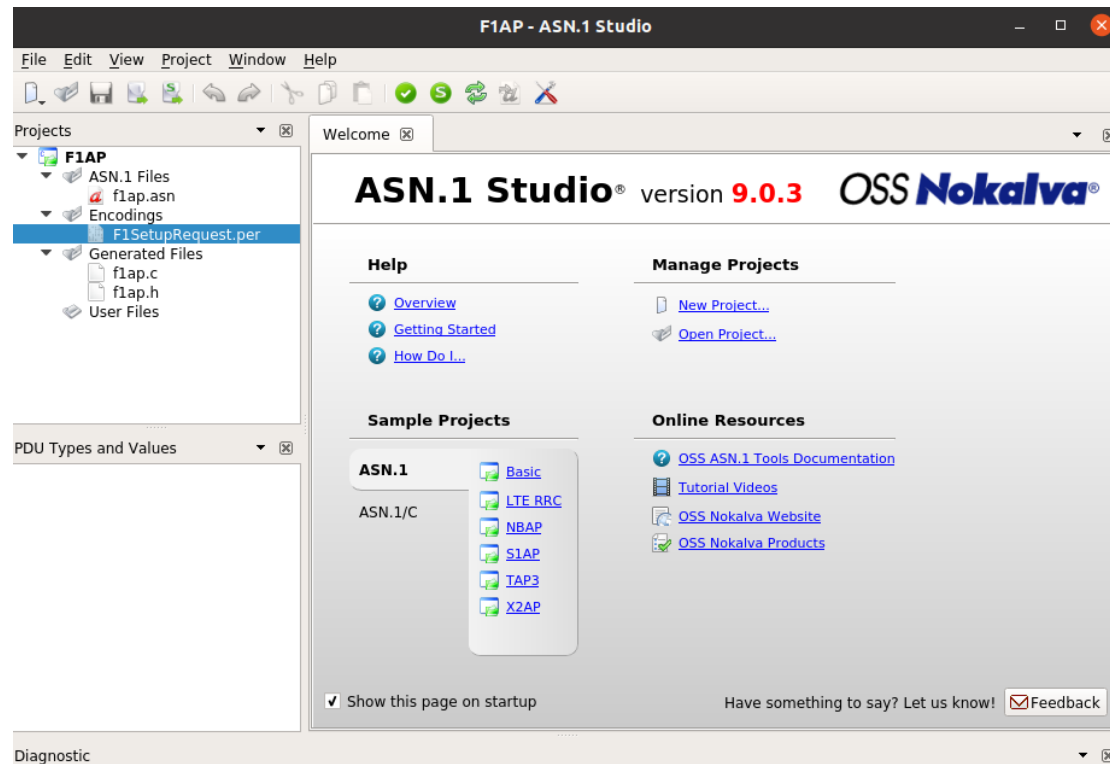


Fig. 4.4: ASN1C Studio.

It makes checking constraints for IEs easier which in turn helps in faster debugging. Fig 5.3 depicts the F1 Setup Request IE with filled IEs (first message transferred between O-DU and O-CU).

# CHAPTER 5

## Discussion of Findings

### 5.1 MAC Execution Time

The timing analysis was performed using `clock()` function defined in `<time.h>` library. `clock` function gives the time used by the program so far (user time + system time). The result is `CLOCKS_PER_SECOND` is program time in seconds.

Code was run on a computer with an Intel(R) Core(TM) i5-7200U CPU running @ 2.50GHz using 8GB of RAM, running Ubuntu version 16.04 with following specs:

- CPU MHz: 600.014
- CPU max MHz: 3100.0000
- CPU min MHz: 400.0000
- Thread(s) per core:2
- Core(s) per socket:2
- Socket(s):1

(Generate Data and Perform MAC functionalities)

Order of execution time for thread1:	0.01ms
--------------------------------------	--------

(Sending Transport Block Over the Socket)

Order of execution time for thread2:	0.1ms
--------------------------------------	-------

```

-----inside thread1-----
-----generated data-----
00 03 01 03 00 00 02 00 01 04 02 04 04 02 04 04 01 02 02 03 04 02
00 00 03 03 04 01 02 03 04 03 02 02 03 04 04 00 04 00 00 03 02 04
00 03 00 03 02 00
-----After 8b header addition-----
05 32 00 03 01 03 00 00 02 00 01 04 02 04 04 02 04 04 01 02 02 03
04 02 00 00 03 03 04 01 02 03 04 03 02 02 03 04 04 00 04 00 00 03
02 04 00 03 00 03 02 00

-----inside thread2 -----
No. of clicks: 52 clicks (0.000052 seconds). For generating 5079ith packet
-----inside thread1-----
-----generated data-----
02 02 02 04 02 00 03 03 01 00 03 02 00 00 04 01 01 03 01 00 00 03
01 04 00 01 04 00 02 02 00 01 04 04 00 03 04 00 01 02 01 04 01 03
02 02 04 03 02 03
No more SDUs, TRANSPORT BLOCK FULL

-----TB_SIZE : 468 -----
----- MUX OUTPUT-----
05 32 00 04 04 00 02 01 03 03 01 01 04 00 02 03 03 02 03 02 00 04
03 02 01 01 04 00 01 00 04 01 00 04 02 04 01 04 03 00 02 04 01 04
02 01 02 00 00 03 04 00 05 32 02 03 02 00 04 01 00 03 03 04 01 03
01 03 00 04 00 03 01 04 04 00 03 01 01 03 04 03 01 00 03 03 03 03
00 00 04 00 00 00 04 03 03 02 03 00 02 03 00 03 05 32 00 00 00 03
01 03 01 02 01 04 03 00 04 03 03 04 00 04 04 02 04 01 00 00 03 04
00 02 04 03 03 04 03 03 00 01 02 03 04 03 03 04 00 02 04 00 04 00
00 03 05 32 02 01 04 03 01 00 04 02 03 02 00 00 02 04 00 03 01
03 04 01 03 00 02 03 00 02 02 00 04 02 04 01 04 04 02 04 03 03 01
04 00 04 04 04 03 01 03 01 04 05 32 04 04 00 00 01 03 02 01 02 02
00 01 03 01 00 03 01 04 03 04 03 02 02 02 03 01 02 04 01 00 01 03
04 01 03 01 01 00 04 00 04 04 01 02 03 04 02 01 03 01 05 32 02 03
00 04 00 04 03 04 03 01 04 01 04 04 02 04 02 00 01 01 02 02 02 04
02 02 00 04 03 03 02 01 02 03 02 04 02 02 01 02 01 00 04 00 01 03
02 03 01 00 05 32 01 03 03 04 04 00 03 04 01 02 00 04 00 04 02 04
00 01 02 01 00 00 02 04 00 03 00 04 04 01 00 00 01 03 01 03 00 00
02 01 04 04 02 01 03 01 00 04 02 04 05 32 00 00 04 04 04 02 00 01
03 01 04 03 01 01 03 00 04 03 02 03 02 01 03 01 04 01 03 01 02 02
03 03 02 02 04 04 01 01 00 00 02 00 00 01 03 04 01 02 04 03 05 32
00 03 01 03 00 00 02 00 01 04 02 04 04 02 04 04 01 02 02 03 04 02
00 00 03 03 04 01 02 03 04 03 02 02 03 04 04 00 04 00 00 03 02 04
00 03 00 03 02 00 3f 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

-----PACKET SENT -----
No. of clicks: 182 clicks (0.000182 seconds).For sending 508 thTB in thread2

```

Fig. 5.1: Timing for Thread 1 and Thread 2

## 5.2 F1-Setup Encoding & Decoding

### 5.2.1 Introduction

F1-Setup is one of the initial messages sent from the gNB-DU to the gNB-CU in order to initiate the setup procedure between both the units. It is the initiating message for a potential handshake consisting of 4 different messages for the setup of the F1-interface.

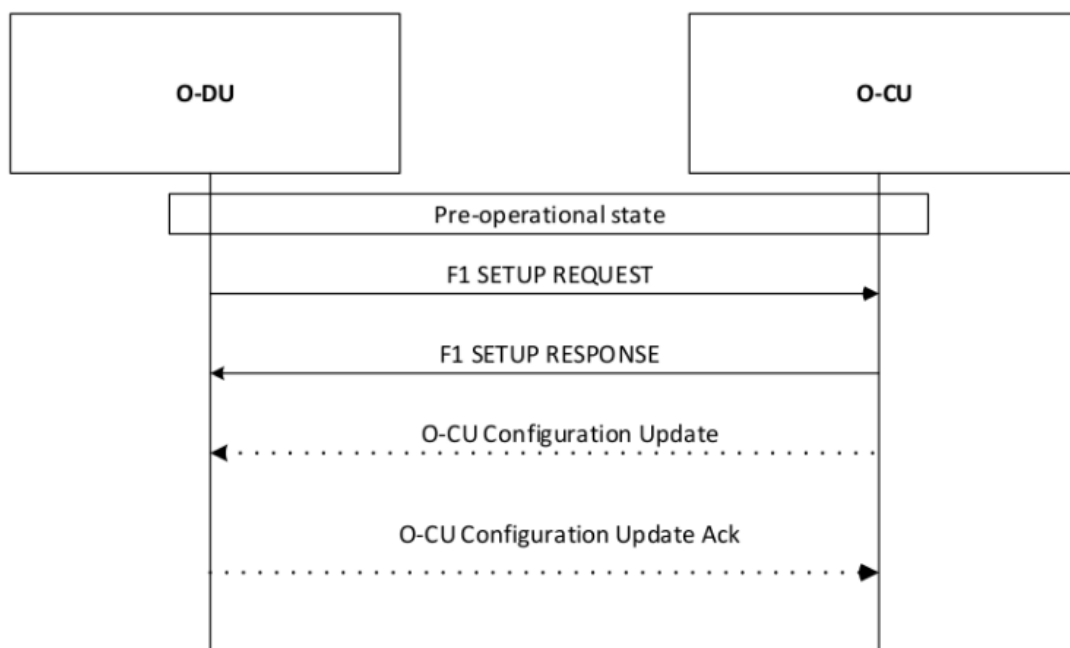


Fig. 5.2: F1-Setup Request.

### 5.2.2 Procedure

The ASN1STUDIO software was used to compile the F1 interface.asn file in order to generate the C files. Later the C structures were filled using the APER encoding rule in order to generate the desired output.

▼ F1SetupRequest		F1SetupRequest	SEQUENCE		0000 0500 4E00 0200 0C0...
▼ protocols	5		SEQUENCE OF	(SIZE(0..65535))	<.000 0000>00 0500 4E00...
▼ 1			SEQUENCE		00 4E00 0200 0C
id	78	ProtocolIE-ID	INTEGER <F1AP-PROTOCOL-IES.6id>	(0..65535)	00 4E
criticality	reject	Criticality	ENUMERATED <F1AP-PROTOCOL-IES.&criticality>		<00...>
value	TransactionID		F1AP-PROTOCOL-IES.&Value		<.00 0000> 0200 0C
TransactionID	12	TransactionID	INTEGER	(0..255, ...)	00 0C
▼ 2			SEQUENCE		00 2A00 0200 02
id	42	ProtocolIE-ID	INTEGER <F1AP-PROTOCOL-IES.6id>	(0..65535)	00 2A
criticality	reject	Criticality	ENUMERATED <F1AP-PROTOCOL-IES.&criticality>		<00...>
value	GNB-DU-ID		F1AP-PROTOCOL-IES.&Value		<.00 0000> 0200 02
GNB-DU-ID	2	GNB-DU-ID	INTEGER	(0..68719476735)	00 02
▼ 3			SEQUENCE		00 2D40 0702 0063 656C 6...
id	45	ProtocolIE-ID	INTEGER <F1AP-PROTOCOL-IES.6id>	(0..65535)	00 2D
criticality	ignore	Criticality	ENUMERATED <F1AP-PROTOCOL-IES.&criticality>		<01...>
value	GNB-DU-Name		F1AP-PROTOCOL-IES.&Value		<.00 0000> 0702 0063 65...
GNB-DU-Name	"cell1"	GNB-DU-Name	PrintableString	(SIZE(1..150, ...))	02 0063 656C 6C31
▼ 4			SEQUENCE		002C 0033 0000 002B 002...
id	44	ProtocolIE-ID	INTEGER <F1AP-PROTOCOL-IES.6id>	(0..65535)	002C
criticality	reject	Criticality	ENUMERATED <F1AP-PROTOCOL-IES.&criticality>		<00...>
value	GNB-DU-Served-Cells-List		F1AP-PROTOCOL-IES.&Value		<.00 0000> 33 0000 002B ...
GNB-DU-Served-Cells-List	1	GNB-DU-Served-Cells-List	SEQUENCE OF	(SIZE(1..512))	0000 002B 002D 4000 000...
▼ 1			SEQUENCE		002B 002D 4000 0000 000...
id	43	ProtocolIE-ID	INTEGER <F1AP-PROTOCOL-IES.6id>	(0..65535)	002B
criticality	reject	Criticality	ENUMERATED <F1AP-PROTOCOL-IES.&criticality>		<00...>
value	GNB-DU-Served-Cells-Item		F1AP-PROTOCOL-IES.&Value		<.00 0000> 2D 4000 0000 ...
GNB-DU-Served-Cells-Item		GNB-DU-Served-Cells-Item	SEQUENCE		4000 0000 0000 0000 0000...
▼ served-Cell-Information		Served-Cell-Information	SEQUENCE		<...0 0000> 00 0000 0000 ...
GNB-DU-System-Information		GNB-DU-System-Information	SEQUENCE		0001 2501 AB
mib-message	'25'H	MIB-message	OCTET STRING		<.00 0000> 01 25
sib1-message	'AB'H	SIB1-message	OCTET STRING		01 AB
IE-Extensions		IE-Extensions	SEQUENCE OF	(SIZE(1..65535))	
IE-Extensions		IE-Extensions	SEQUENCE OF	(SIZE(1..65535))	

Fig. 5.3: F1-Setup Request on ASN1C Studio.

Encoding Viewer

Aligned PER ▾

☐ Details

00000000	00	00	05	00	4E	00	02	00	0C	00	2A	00	02	00	02	00	....N.....*
00000010	2D	40	07	02	00	63	65	6C	6C	31	00	2C	00	33	00	00	-@...cell1.,.3..
00000020	00	2B	00	2D	40	00	00	00	00	00	00	00	00	00	00	00	..+.-@.....
00000030	00	00	00	00	00	00	00	00	00	00	04	00	00	00	00	00	.....
00000040	00	00	00	04	00	00	00	00	00	00	01	00	00	01	25	01	.....%.
00000050	AB	00	AB	00	0A	80	00	00	00	C7	40	03	0F	09	00		.....@....

Fig. 5.4: Encoded Data in Hex.

### 5.2.3 Decoded Data

The encoded hexadecimal data was then decoded to get back the original contents at the other side of the interface

```
1 value FlSetupRequest ::=
2 {
3   protocolIEs
4   {
5     {
6       id 78,
7       criticality reject,
8       value TransactionID : 12
9     },
10    {
11      id 42,
12      criticality reject,
13      value GNB-DU-ID : 2
14    },
15    {
16      id 45,
17      criticality ignore,
18      value GNB-DU-Name : "cell1"
19    },
20    {
21      id 44,
22      criticality reject,
23      value GNB-DU-Served-Cells-List :
24      {
25        {
26          id 43,
27          criticality reject,
28          value GNB-DU-Served-Cells-Item :
29          {
30            served-Cell-Information
31            {
32              nRCGI
33              {
```

```

34         pLMN-Identity '000000'H,
35         nRCellIdentity '00000000 00000000 00000000 00000000
000 ...'B
36     },
37     nRPCI 0,
38     servedPLMNs
39     {
40     {
41         pLMN-Identity '000000'H
42     }
43     },
44     nR-Mode-Info fDD :
45     {
46         uL-NRFreqInfo
47         {
48             nRARFCN 0,
49             freqBandListNr
50             {
51             {
52                 freqBandIndicatorNr 1,
53                 supportedSULBandList
54                 {
55                 {
56                     freqBandIndicatorNr 1
57                 }
58                 }
59             }
60         }
61     },
62     dL-NRFreqInfo
63     {
64         nRARFCN 0,
65         freqBandListNr
66         {
67         {
68             freqBandIndicatorNr 1,
69             supportedSULBandList

```

```

70         {
71         {
72             freqBandIndicatorNr 1
73         }
74     }
75 }
76 }
77 },
78 uL-Transmission-Bandwidth
79 {
80     nRSCS scs15,
81     nRNRB nrb11
82 },
83 dL-Transmission-Bandwidth
84 {
85     nRSCS scs15,
86     nRNRB nrb11
87 }
88 },
89 measurementTimingConfiguration '00'H
90 },
91 gNB-DU-System-Information
92 {
93     mIB-message '25'H,
94     sIB1-message 'AB'H
95 }
96 }
97 }
98 }
99 },
100 {
101     id 171,
102     criticality reject,
103     value RRC-Version :
104     {
105         latest-RRC-Version '000'B,
106         iE-Extensions

```



```
107     {
108     {
109         id 199,
110         criticality ignore,
111         extensionValue OCTET STRING : '0F0900'H
112     }
113     }
114 }
115 }
116 }
117 }
```

#### 5.2.4 Conclusion

The message was successfully encoded and decoded which allows it to pass through the F1 interface present between gNB-CU and gNB-DU.

Encoding	API function	Understood by	API function
BER	der_encode()	BER	ber_decode()
DER	der_encode()	DER, BER	ber_decode()
CER	<u>not supported</u>	CER, BER	ber_decode()
BASIC-OER	oer_encode()	*-OER	oer_decode()
CANONICAL-OER	oer_encode()	*-OER	oer_decode()
BASIC-UPER	uper_encode()	*-UPER	uper_decode()
CANONICAL-UPER	uper_encode()	*-UPER	uper_decode()
*-APER	<u>not supported</u>	*-APER	<u>not supported</u>
BASIC-XER	xer_encode(...)	*-XER	xer_decode()
CANONICAL-XER	xer_encode (XER_F_CANONICAL)	*-XER	xer_decode()

Fig. 5.5: ASN1C Open Source Compiler Support, Walkin (2010)

The RRC PDU contents in clause 6 and clause 10 are described using abstract syntax notation one (ASN.1) as specified in ITU-T Rec. X.680 [6] and X.681 [7]. Transfer syntax for RRC PDUs is derived from their ASN.1 definitions by use of Packed Encoding Rules, unaligned as specified in ITU-T Rec. X.691 [8].

Fig. 5.6: RRC ASN1 Encoding Scheme: UPER, 3GPP (2020c)

## 9.5 Message Transfer Syntax

F1AP shall use the ASN.1 Basic Packed Encoding Rules (BASIC-PER) Aligned Variant as transfer syntax, as specified in ITU-T Recommendation X.691 [5].

Fig. 5.7: F1AP ASN1 Encoding Scheme: APER, 3GPP (2020a)

## 9.5 Message Transfer Syntax

NGAP shall use the ASN.1 Basic Packed Encoding Rules (BASIC-**PER**) Aligned Variant as transfer syntax as specified in ITU-T Rec. X.691 [4].

Fig. 5.8: NGAP ASN1 Encoding Scheme: APER, 3GPP (2020*b*)

The reason behind going towards commercial compilers is APER support which is to be followed according to the specifications, see Pg:24. The present version of open-source compiler doesn't have APER support.

# APPENDIX A

## ASN1C Open Source Compiler v9.29

### A.1 Installation Guide

1. Clone this branch of repository from github as this branch contains some bug fixes of the original repository.

```
int sockfd = socket(domain, type, protocol);
```

2. Before installing make sure the below packages are present in your system following command can be used:

```
dpkg -s pkg-name
```

Replace pkg-name with following names:

- automake
- libtool
- bison
- flex

3. If these packages are not present then install using following command:

```
sudo apt-get install pkg-name
```

Where pkg-name can be **automake,libtool,flex**.

4. For installing bison 2.x we need to follow different commands as above command will install bison 3.x version

```
1 wget http://launchpadlibrarian.net/140087283/libbison-dev_2
  .7.1.dfsg-1_amd64.deb
2 wget http://launchpadlibrarian.net/140087282/bison_2.7.1.
  dfsg-1_amd64.deb
3 dpkg -i libbison-dev_2.7.1.dfsg-1_amd64.deb
4 dpkg -i bison_2.7.1.dfsg-1_amd64.deb
5
```

To prevent the update manager from overwriting this package:

```
apt-mark hold libbison-dev
apt-mark hold bison
```

[[source](#)]

5. Now that we have all the requirements for installing we can follow this installation guide:  
Go inside the cloned repository and run the below commands:

```
test -f configure || autoreconf -iv
./configure
make
```

Before running the above commands check if **autoconf** is installed using below command:

```
dpkg -s autoconf
```

For installing:

```
sudo apt-get install autoconf
```

6. Ensure `asn1c` is still behaving well after compiling on your platform To be run inside `asn1c` repository

```
make check
```

The above command will run test cases and it will take a while to complete  
Possible Test-case Failure : **XFAIL:1 (check-158.-fcompound-names.c)**  
From this issue of repository, this is an expected failure and we need not worry about it too much.

7. From inside the repository, install using the below command:

```
make install
```

After installation to check if it is installed properly:

```
man asn1c
```

[[source](#)]

8. After successful install, an example to try out:

```
1  asn1c -gen-PER -fcompound-names -findirect-choice -fno-include-deps rrc.asn
```

where `rrc.asn` is ASN1 extracted from TS 38.331 document. see B.

# APPENDIX B

## Extracting ASN1 From 3GPP Technical Specifications

### B.1 Procedure from Specification

The procedure to extract ASN1 is followed according to A.3.1.1 of 38.331. The procedure is as follows:

A typical procedure for extraction of the complete ASN.1 code consists of a first step where the entire RRC PDU contents description (ultimately the entire specification) is saved into a plain text ASCII file format, followed by a second step where the actual extraction takes place, based on the occurrence of the ASN.1 start and stop tag.

### B.2 Steps for Extraction

1. Download the `.docx` version of specification from 3gpp website.  
For RRC, click [here](#)
2. Extract the zip file and Open the `.docx`
3. File-> Save As..-> Choose `.txt` format
4. Use ExtractASN1.py and pass the .txt file as argument (along with path if in a different directory).

```
python ExtractASN1.py rrc.txt
```

5. Check for `rrc.asn` created in the directory from which python script was run.

## B.3 Working of the Python Script

There can be many ways for searching for all occurrences of substrings. One easy way is to use regular expression.

This python script uses functions of regular expression library `re` to search for "--ASN1START" and "--ASN1STOP" substrings.

The function `re.finditer(pattern,string,flags=0)`:

Return an iterator yielding match objects over all non-overlapping matches for the RE pattern in string. The string is scanned left-to-right, and matches are returned in the order found. Empty matches are included in the result.

ExtractASN1.py : Working of script is explained below in snippets.

```
1 import sys
2 import re
3 # count the arguments
4 arguments = len(sys.argv)
5 if (arguments!=2):
6     print("Expected No. of Arg = 1\t Given No. of Arg = "+ str(
7         arguments-1))
8     print("usage:" + "\t" + sys.argv[0] + "\t" "<inputfile.txt>")
9     exit()
```

The above code imports libraries and checks for the input `.txt` file.

```
9 with open (sys.argv[1], 'rt') as myfile:
10 contents = myfile.read()
11 alpha = re.finditer("-- ASN1START|-- /example/ ASN1START|-- /bad
12     example/ ASN1START", contents)
13 beta = re.finditer("-- ASN1STOP|-- /example/ ASN1STOP", contents)
14 alpha_start = []
15 alpha_end = []
```

1. Opens the input file Spec) and reads it into string variable `contents`
2. `re.finditer()` function finds for all occurrences of substrings "`— ASN1START`" or "`— /example/ ASN1START`" or "`— /bad example/ ASN1START`" and "`— ASN1STOP`" or "`— /example/ ASN1STOP`" in the specification.
3. This function returns an iterator which yields match objects which can be used to find where the expression occurs in the document.
4. The reason for searching `— /example/.*` is for matching no. of occurrences of "`— ASN1START`" with "`— ASN1STOP`". As some examples in the spec end with "`- - ASN1STOP`" rather than "`— /example/ ASN1STOP`".
5. If we try to search only for "`— ASN1START`" and "`— ASN1STOP`" we would end up with more number of "`— ASN1STOP`" than "`— ASN1START`". Eventually we have to ignore examples in the actual output.
6. We create empty lists which will hold start and end positions of match objects in `alpha` for ("`— ASN1START/ /example/ ASN1START/ /bad example/ ASN1START`") and `beta` ("`— ASN1STOP/ /example/ ASN1STOP`").

```

15     for match in alpha:
16         start,end = match.span()
17         alpha_start.append(start)
18         alpha_end.append(end)
19
20     for match in beta:
21         start,end = match.span()
22         beta_start.append(start)
23         beta_end.append(end)
24

```

7. The `match.span()` method gives out start and end position of each occurrence in `alpha`.
8. The variables `start` and `end` are appended to list variables and `alpha_end` for every match respectively.
9. Steps 7 and 8 are repeated for `beta`

```

25     name = sys.argv[1]
26     lastpos = name.rfind('.')
27     filename = name[0:lastpos]
28
29     asnfile = open(filename+".asn", "w")
30     for x in range(len(beta_start)):
31         if alpha_end[x]-alpha_start[x] == 12 and beta_end[x]-
beta_start[x] == 11:
32             asnfile.write(contents[alpha_start[x]:beta_end[x]])
33             # write contents
34             asnfile.write('\n\n')
35     asnfile.close()

```



10. `open()` function is used to create a new file with same name as the argument in write mode.
11. We run a for loop no. of times substrings "`--ASN1START`" or "`--/example/ASN1START`" or "`--/bad example/ASN1START`" occur. In the above code `len(beta_start)` could be used as well. As both `len(beta_start)` and `len(alpha_start)` are equal.
12. Now we need to check for only ("`--ASN1START`") and ("`--ASN1STOP`") which translates to checking if `alpha_end[x]-alpha_start[x]==12` and `beta_end[x]-beta_start[x]==11`.
13. `if` statement helps in excluding matches "`--/example/ASN1START`" or "`--/bad example/ASN1START`" and "`--/example/ASN1STOP`".
14. Finally we write `contents[alpha_start[x]:beta_end[x]] { -`  
`-ASN1START....--ASN1STOP}` into the file and write a newline to separate the next match. The process continues for all the occurrences.
15. Close the file after all the writes finish.

# APPENDIX C

## Socket Programming in C/C++

### C.1 Introduction

Socket programming is a means of uniting two nodes on a network to interact with each other. One of the sockets listens on a specific port at an IP address, whilst another socket reaches out to the other to create a connection. The server creates the listener socket while the client tries to reach out to the server. Sinha (2020)

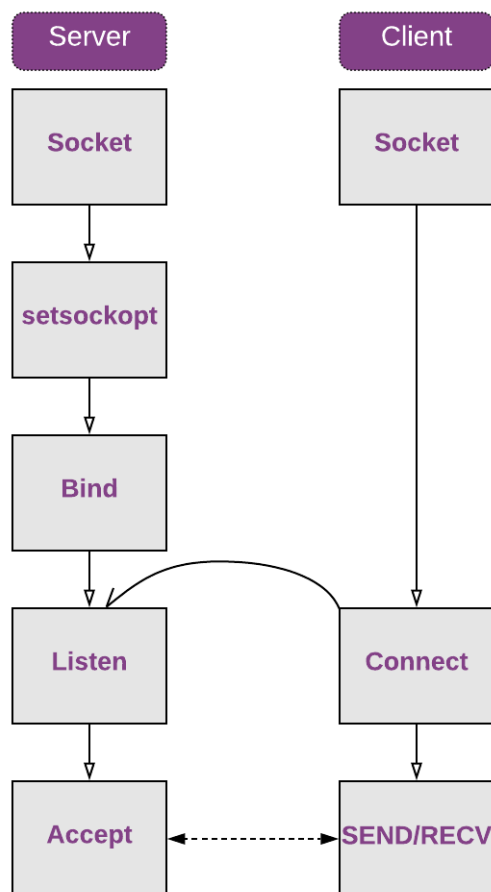


Fig. C.1: State diagram for server and client model.

## C.2 API's for server

### C.2.1 Socket creation

```
int sockfd = socket(domain, type, protocol) ;
```

**sockfd:** socket descriptor is an integer similar to a file-handle

**domain:** Type: integer;communication domain Example: IPv4 protocol: AF\_INET, IPv6 protocol: AF\_INET6 ()

**type:** communication type

- SOCK\_STREAM: TCP(reliable, connection oriented)
- SOCK\_DGRAM: UDP(unreliable, connectionless)

**protocol:** Protocol value for Internet Protocol(IP) is zero. The same number appears on the protocol field in the IP header of a packet. ( [man protocols](#) can give out more information)

### C.2.2 setsockopt

```
int setsockopt(int sockfd, int level, int optname,  
               const void *optval, socklen_t optlen);
```

The [setsockopt](#) function assists in managing options for the socket indicated by the file descriptor sockfd. This functionality is fully optional. However, it can help in reuse of the address and the port. Sinha (2020)

### C.2.3 Bind

```
int bind(int sockfd, const struct sockaddr *addr,  
         socklen_t addrlen);
```

Following the formation of the socket, the bind function is responsible for binding the socket to the address and port number specified in addr(custom data structure).

## C.2.4 Listen

```
int listen(int sockfd, int backlog);
```

It sets the server socket in an idle mode. In this mode, it waits for the client to reach the server to create a connection. The backlog specifies the maximum length till which the queue of pending connections for sockfd may grow. If a connection request reaches when the queue is full, the client can receive an error with a sign of **ECONNREFUSED**.

## C.2.5 Accept

```
int new_socket= accept(int sockfd, struct sockaddr *addr,  
                      socklen_t *addrlen);
```

It selects the first connection request on the queue of pending connections for the listening socket, sockfd. Creates a new connected socket, and then returns a new file descriptor referring to that socket. At this moment, the connection is established between client and server, and are ready for transferring data.

# C.3 API's for client

## C.3.1 Socket creation

Precisely the same as that of socket creation at the server-side.

## C.3.2 Connect

```
int connect(int sockfd, const struct sockaddr *addr,  
           socklen_t addrlen);
```

The `connect()` system call is responsible for connecting the socket indicated by the file descriptor `sockfd` to the address given by `addr`. The Server's address and port are defined in `addr` structure. Sinha (2020)

## REFERENCES

1. **3GPP** (2020a). *NG-RAN; F1 Application Protocol (F1AP); Protocol specification*. Technical Specification (TS) 38.473, 3rd Generation Partnership Project (3GPP). URL <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3260>. Version 15.9.0.
2. **3GPP** (2020b). *NG-RAN; NG Application Protocol (NGAP)*. Technical Specification (TS) 38.413, 3rd Generation Partnership Project (3GPP). URL <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3223>. Version 15.8.0.
3. **3GPP** (2020c). *NR; Radio Resource Control (RRC); Protocol specification*. Technical Specification (TS) 38.331, 3rd Generation Partnership Project (3GPP). URL <https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3197>. Version 15.9.0.
4. **Ahmadi, S.** (2019). *5G NR: Architecture, Technology, Implementation, and Operation of 3GPP New Radio Standards*. Elsevier Science. ISBN 9780128134023. URL <https://books.google.co.in/books?id=RZadDwAAQBAJ>.
5. **Casaccia, L.** (2017). Understanding 3gpp – starting with the basics. URL <https://www.qualcomm.com/news/onq/2017/08/02/understanding-3gpp-starting-basics>.
6. **Dahlman, E., S. Parkvall, and J. Skold** (2018). *5G NR: The Next Generation Wireless Access Technology*. Elsevier Science. ISBN 9780128143247. URL <https://books.google.co.in/books?id=C5poDwAAQBAJ>.
7. **ITU** (2014). Introduction to asn.1. URL <https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspxf>.
8. **Jain, R.** (2020). Multithreading in c. URL <https://www.geeksforgeeks.org/multithreading-c-2/>.
9. **Mpirical** (2020). Glossary. URL <https://www.mpirical.com/glossary>.
10. **objective SYSTEMS®** (2020). Asn1c: Asn.1 compiler. URL <https://www.obj-sys.com/products/asn1c/index.php>.
11. **O\_RAN®** (2019). Base Station O-DU and O-CU Software Architecture and APIs. URL <https://www.o-ran.org/>. V1.0.0.
12. **OSS\_Nokalva®** (2020). Asn.1 made simple — what is asn.1? URL <https://www.oss.com/asn1/resources/asn1-made-simple/introduction.html>.

13. **Sinha, A.** (2020). Socket Programming in C/C++. URL <https://www.geeksforgeeks.org/socket-programming-cc/>.
14. **Walkin, L.** (2010). Open source asn.1 compiler asn1c quick start sheet. URL <http://lionet.info/asn1c/asn1c-quick.pdf>.
15. **Wikipedia** (2020). International mobile telecommunications-2020. URL <https://en.wikipedia.org/wiki/IMT-2020>.