5G L2/L3 Protocol Stack Development

A Project Report

submitted by

NITIN PRIYADARSHINI SHANKAR

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 project report titled **5G L2/L3 Protocol Stack Development**, submitted by **Nitin Priyadarshini Shankar**, to the Indian Institute of Technology Madras, for the award of the degree of **Bachelor of Technology**, is a bonafide record of the project work done by him under my supervision. The contents of this 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.

Place: Chennai

Date: 18th Jul 2020

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

ACKNOWLEDGEMENTS

I would like to take this opportunity to sincerely thank my guide **Dr. Krishna Jagannathan** for his encouragement and 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 test-bed project.

I would also like to thank **Jaswanthi Mandalapu, Jitesh Kumar Gupta** and the other members of the IITM testbed team for helping me learn new things in such a vast field. I would also like to acknowledge the help and knowledge provided by the CEWiT which helped me understand the implementation specific aspects of mobile communications.

Finally I would like to thank my parents for supporting me in all my endeavors and being there for me in the toughest of times and making me into who I am today.

ABSTRACT

KEYWORDS: 5G-NR ; SDAP; LTE; Makefile; ASN.1 ; RRCs.

The L2/L3 protocol stack is a main component of the Radio Access Network. This document contains a summary of my work done for the stack. The main focus of my work was to develop the SDAP sub-layer, which is the highest layer of the protocol stack. It is new to 5G-NR and was not present in LTE (the previous generation).

The development of the API required some knowledge on Makefiles and their applications, which was also studied during this project course. Makefiles are an efficient way of handling projects which contains multiple execution files and they are explained in this report.

ASN.1 was also studied in order to encode and decode the messages related to RRC sub-layer and the interface messages. Various tools, both open source and commercial were learnt in order to process the ASN.1 structures and convert them to the C programming language.

TABLE OF CONTENTS

Page

ACKNOWLEDGEMENTS i

ABSTRACT ii

LIST OF FIGURES vi

GLOSSARY vii

ABBREVIATIONS viii

CHAPTER 1: Makefile 1

1.1 Introduction 1

1.2 Basic Makefile structure 1

1.2.1 Dependency Rules 1

1.2.2 Macros 2

1.2.3 Suffix rules 3

1.3 Order of execution 3

1.4 Sample Makefile 4

CHAPTER 2: The L2/L3 Protocol Stack 6

2.1 Introduction 6

2.1.1 Protocol stack architecture 7

2.1.2 Data Flow 8

CHAPTER 3: The SDAP Sub-Layer 9

3.1 Introduction 9

3.2 QoS Handling 9

3.2.1 Example QoS Requirements 10

3.3 SDAP Entity 11

3.4 Architecture (Specified by 3GPP) 12

3.4.1 Structure 12
3.4.2 Functions 13
3.5 PDU types of the SDAP sub-layer 14

CHAPTER 4: The SDAP API 16
4.1 Introduction 16
4.2 Functionality 16
4.3 Code fragments and performance 19
4.3.1 Header Addition 19
4.3.2 Header Removal 20
4.3.3 Mapping 21

CHAPTER 5: ASN.1 23
5.1 Introduction 23
5.2 Encoding rules 23
5.3 ASN1 compilers 24
5.3.1 obj-sys compiler 25
5.3.2 OSS Noklava compiler 25
5.4 F1-Setup Encoding 27
5.4.1 Introduction 27
5.4.2 Procedure 27
5.4.3 Decoded Data 29
5.4.4 Conclusion 32

APPENDIX A: Socket Programming in C/C++. 33
A.1 Introduction 33
A.2 API's for server 33
A.2.1 Socket creation 33
A.2.2 Bind 34
A.2.3 Listen 34
A.2.4 Accept 34
A.3 API's for client 35

A.3.1 Socket creation 35

A.3.2 Connect 35

APPENDIX B: Timing Analysis 36

B.1 Introduction 36

B.2 <time.h> 36

B.3 Calculation 37

B.4 Sample Program 37

REFERENCES 38

LIST OF FIGURES

Figure	Title	Page
2.1	The 5G User-plane and Control-plane protocol stack.	6
2.2	NR downlink user-plane protocol architecture as seen from the device.	7
2.3	Example of user-plane data flow.	8
3.1	GBR type QoS Flows.	10
3.2	Non-GBR type QoS Flows.	11
3.3	Structural View.	12
3.4	Functional View.	13
3.5	An example of an SDAP data PDU when the header is not configured.	14
3.6	Format of an SDAP data PDU formed during Downlink and if the header is configured.	14
3.7	Format of an SDAP data PDU formed during Uplink and if the header is configured.	15
3.8	Format for an end marker control PDU.	15
4.1	Control of SDAP entity.	17
4.2	Sample DL data flow in an SDAP entity.	18
5.1	Encoding rules.	23
5.2	Comparison of different encoding rules.	24
5.3	ASN1C Workflow.	24
5.4	ASN1C Studio.	26
5.5	F1-Setup Request.	27
5.6	F1-Setup Request on ASN1C Studio.	28
5.7	Encoded Data in Hex.	28
A.1	Describes various states followed by the server and client.	33

GLOSSARY

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

Sub-layers	A bunch of Protocols that work together in order to achieve the functionality of that particular layer it is part of.
UE measurement	Various parameters such as power, signal strength etc which are measured periodically and reported.
QoS flow	Parameters used to distinguish data based on the priority requirements of different types of data.
MAC TB	It is the output of the MAC sub-layer which is passed on to the Physical layer.
Downlink	Flow of data from the 5GC to the UE.
Network slicing	Network slicing is a specific form of virtualization that allows multiple logical networks to run on top of a shared physical network infrastructure.

ABBREVIATIONS

UE	User Equipment
RAN	Radio Access Network
RLC	Radio Link Control
IP	Internet Protocol
NAS	Non - Access Stratum
QoS	Quality of Service
DL	Downlink
PDCP	Packet Data Convergence Protocol
UL	Uplink
DRB	Data Radio Bearer
SDU	Service Data Unit
PDU	Protocol Data Unit
ARQ	Automated Repeat Request
HARQ	Hybrid ARQ
TB	Transport Block
SAP	Service Access Point
QFI	QoS Flow Indicator
LTE	Long Term Evolution
gNB	5G Base Station
MAC	Media Access Control
EPC	Evolved Packet Core
GBR	Guaranteed Bit-Rate
RRC	Radio Resource Control

SDAP	Service Data Adaptation Protocol
5GC	5G Core Network
ASN.1	Abstract Syntax Notation 1
gNB-DU	Distributed Unit of 5G Base station
gNB-CU	Central Unit of 5G Base station

CHAPTER 1

Makefile

1.1 Introduction

This chapter is referred from Stallman (2020). These are some of the important points regarding make files.

- The make package is used for managing computer programs consisting of many executable files. The make package automatically decides which parts of a large program code need to be recompiled and issues suitable command to recompile them.
- GNU make reads its instruction from a Makefile. It searches for the presence of the Makefile in the current folder by default.
- A Makefile establishes a set of rules to determine which parts of an application need to be compiled again, and issues commands to recompile them.
- A Makefile helps in simplifying software development other complex tasks with various dependencies.
- The Makefile comprises of 3 parts namely, **dependency rules**, **macros** and **suffix rules**.

1.2 Basic Makefile structure

1.2.1 Dependency Rules

A rule has three parts, at least one target, any number of dependencies, and any number of commands with the following syntax:

target: **dependencies**

<tab> **command(s)** to make target

- The <tab> character is necessary before each command.
- A **target** can be the name of a file or an action such as clean, print etc.
- **Dependencies** are files that are required by the target to execute its contents.
- Each **command** that is present in a rule is interpreted by the default shell.
- By default, make uses the /bin/sh shell.
- If the "make **target**" command is invoked in the shell, it will:
  1. Make sure all the dependencies are up to date.
  2. If make finds out that a target is older than any of its dependencies, it re-compiles the target using the specified commands.
- Typing "make" creates the first target that is described in the Makefile by default.
- A **phony target** is not a target file. It will only have a list of commands to perform certain actions such as clean. It has no dependencies.

  Example: **clean: rm -rf *.o**

1.2.2 Macros

- By using macros, we can avoid repeating text entries and thereby making the makefile easy to modify.
- Macro definitions are of the form:

  'Macro Name' = text string
  Example: **CC = g++**

- Macros are later referenced by placing the macro name in between parentheses and preceding it with the $ sign.

  Example:
  $(CC) main.o factorial.o hello.o -o prog

- 'Command line macros' are macros that can be defined on the command line.

  Example:
  make DEBUG_FLAG=-g

- Internal macros
  - The Internal macros are predefined in make and can be used without explicit definitions.

  - “make -p” is a type of internal macro that is used to display a list of all the macros, suffix rules and targets that are present in the current build.

- Special macros
  - The macro @ is used to evaluate the name of the current target.

Example:

```
prog1 : $(objs)
$(CXX) -o $@ $(objs)
```

is equivalent to

```
prog1 : $(objs)
$(CXX) -o prog1 $(objs)
```

1.2.3 Suffix rules

Is is a way of defining default rules that ’make’ can use to build a program. There are two types of suffixes namely, double-suffix and single-suffix.

- Doubles-suffix is written in terms of a source suffix and a target suffix.

  Example:

  ```
  .cpp.o:
      $(CC) $(CFLAGS) -c $<
  ```

  - This rule is used to substitute the .o files and .cpp files from the command.
  - $< is a special macro which in this case stands for a .cpp file that is used to produce a .o file.

- This is same as the pattern rule “%.o : %.cpp”

  ```
  %.o : %.cpp
      $(CC) $(CFLAGS) -c $<
  ```

1.3 Order of execution

- The make utility keeps track of the modification times of the target files and the dependency files. If any dependency file that is used by a target file has been modified after the last compilation, the make utility issues commands to compile the target file again.

- The first target file is the one that is built. The rest of the targets are ignored unless the first target depends on other targets defined below.
- Make takes care of the order in which the targets have to be created.

1.4 Sample Makefile

```
IDIR1 = ../../../gNB/inc
IDIR2 = ../../../common/ringBuffer/inc
IDIR3 = ../../../common/headers/inc
SDIR1 = ../../../gNB/src
SDIR2 = ../../../common/ringBuffer/src
SDIR3 = ../../../common/headers/src

CC=gcc
CFLAGS  =-I$(IDIR1)
CFLAGS +=-I$(IDIR2)
CFLAGS +=-I$(IDIR3)

ODIR=obj
LDIR =../lib

LIBS=

_DEPS1 = gNB_map.h  make_gNB_data.h
DEPS = $(patsubst %,$(IDIR1)/%,$(_DEPS1))

_DEPS2 = ringBuffer.h
DEPS += $(patsubst %,$(IDIR2)/%,$(_DEPS2))

_DEPS3 = SDAP_headers.h
DEPS += $(patsubst %,$(IDIR3)/%,$(_DEPS3))

_OBJ =  gNB_RX.o  make_gNB_PDU.o  make_gNB_SDU.o  ringBuffer.o
    SDAP_headers.o
OBJ = $(patsubst %,$(ODIR)/%,$(_OBJ))

```

```

$(ODIR)/%.o: %.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

$(ODIR)/%.o: $(SDIR1)/%.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

$(ODIR)/%.o: $(SDIR2)/%.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

$(ODIR)/%.o: $(SDIR3)/%.c $(DEPS)
	$(CC) -c -o $@ $< $(CFLAGS)

test_gNB_RX: $(OBJ)
	$(CC) -o $@ $^ $(CFLAGS) $(LIBS)

.PHONY: clean

clean:
	rm -f $(ODIR)/*.o test_gNB_RX
```

The above makefile was used in the development of the SDAP API

CHAPTER 2

The L2/L3 Protocol Stack

2.1 Introduction

The L2/L3 Protocol Stack is part of the RAN (UE + gNB). It has the following Sub-Layers that help achieve its functionality.

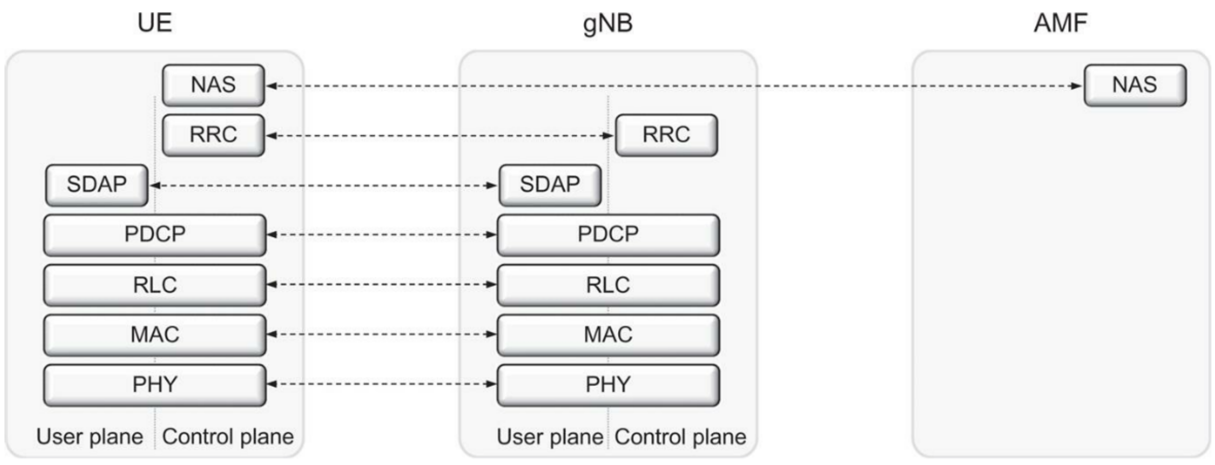


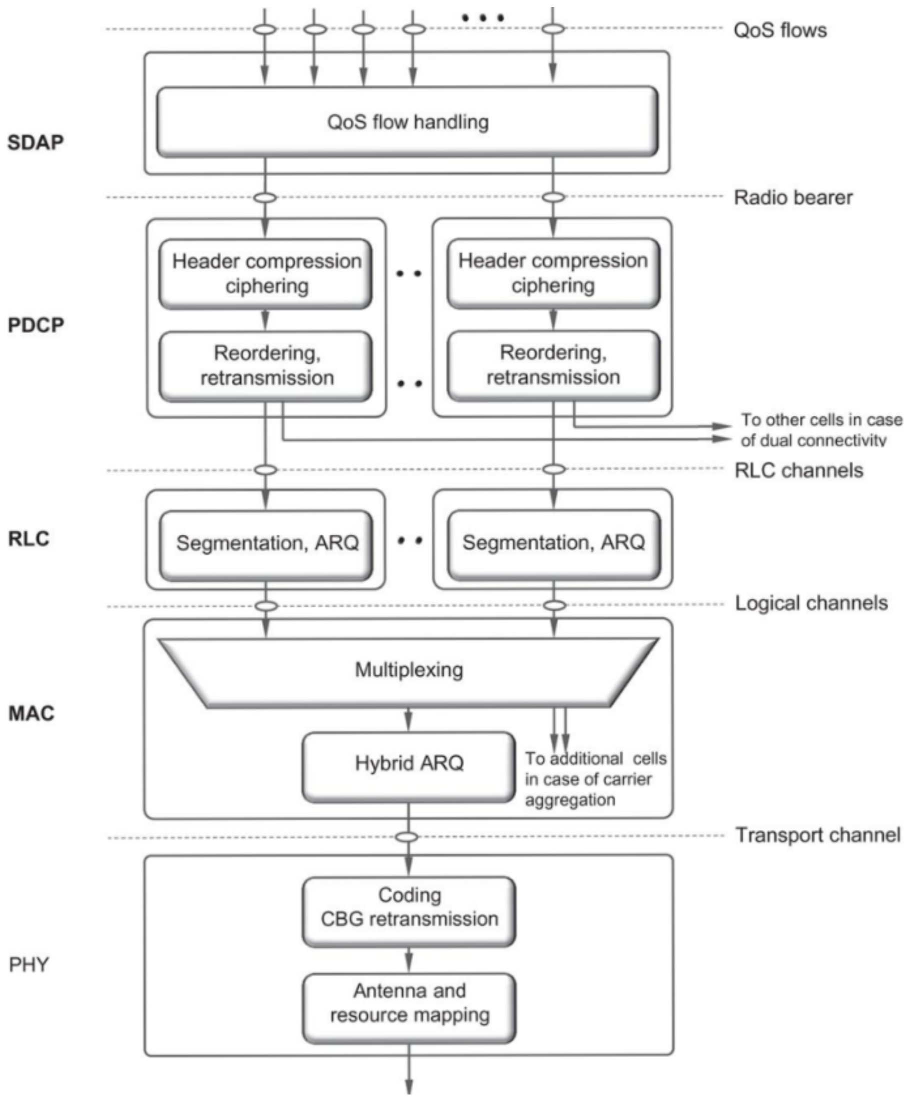
Figure 2.1: The 5G User-plane and Control-plane protocol stack.

The Sub-layers that are part of the L2/L3 protocol stack are as follows:

- User - Plane
  - SDAP
  - PDCP
  - RLC
  - MAC
- Control - Plane
  - RRC
  - PDCP
  - RLC
  - MAC

The SDAP sublayer will be studied in detail.

2.1.1 Protocol stack architecture



QoS flows
SDAP
QoS flow handling
Radio bearer
PDCP
Header compression ciphering
Reordering, retransmission
To other cells in case of dual connectivity
RLC channels
RLC
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 2.2: NR downlink user-plane protocol architecture as seen from the device.

This is the architecture as specified by 3GPP which we have followed in order to build the protocol stack. We are able to see the various connections between different sub-layers also known as service access points(SAPs). There are various types of connections between two subsequent layers which is also shown in the diagram.

2.1.2 Data Flow

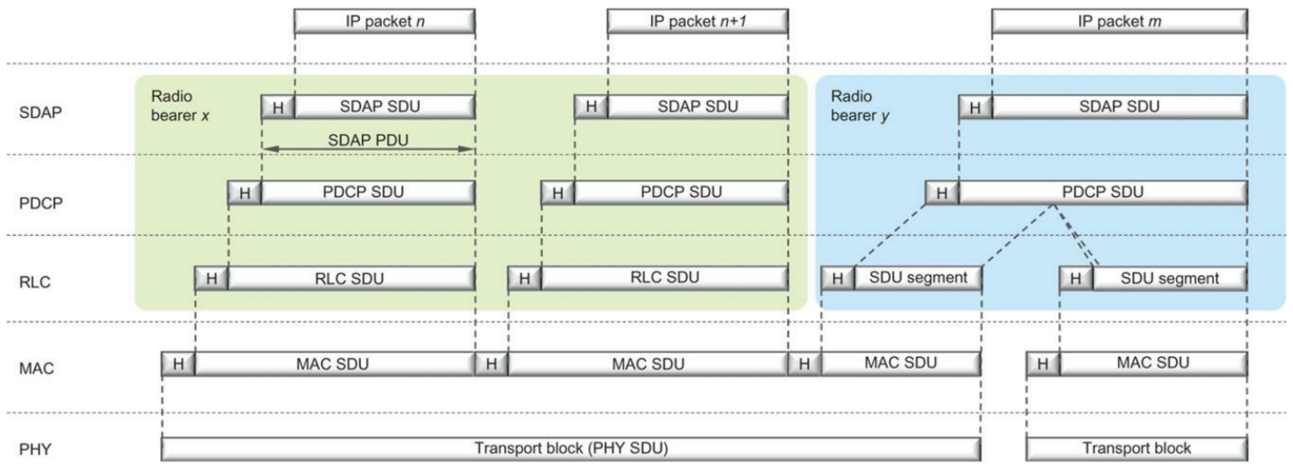


Figure 2.3: Example of user-plane data flow.

This picture shows an example data flow between the sub-layers according to the architecture specified by 3GPP. We can see that the SDAP and PDCP Sub-layers add their headers in the respective radio bearers.

In the RLC sub-layer, there is an option of segmentation in which the packets are split on one side and combined back together on the other side.

The MAC sub-layer takes care of appending various packets together and forming the MAC TB. Then the TB is passed on to the physical layer.

CHAPTER 3

The SDAP Sub-Layer

3.1 Introduction

The SDAP sub-layer is responsible for mapping between a QoS flow from the 5GC and a DRB in the gNB, as well as marking the quality-of- service flow identifier (QFI) in UL and DL packets. The reason for the introduction of SDAP in NR is the new quality-of-service handling compared to LTE when connected to the 5G core. The SDAP sub-layer is also responsible for the mapping between QoS flows and DRBs. This chapter has been cited from Dahlman *et al.* (2018)

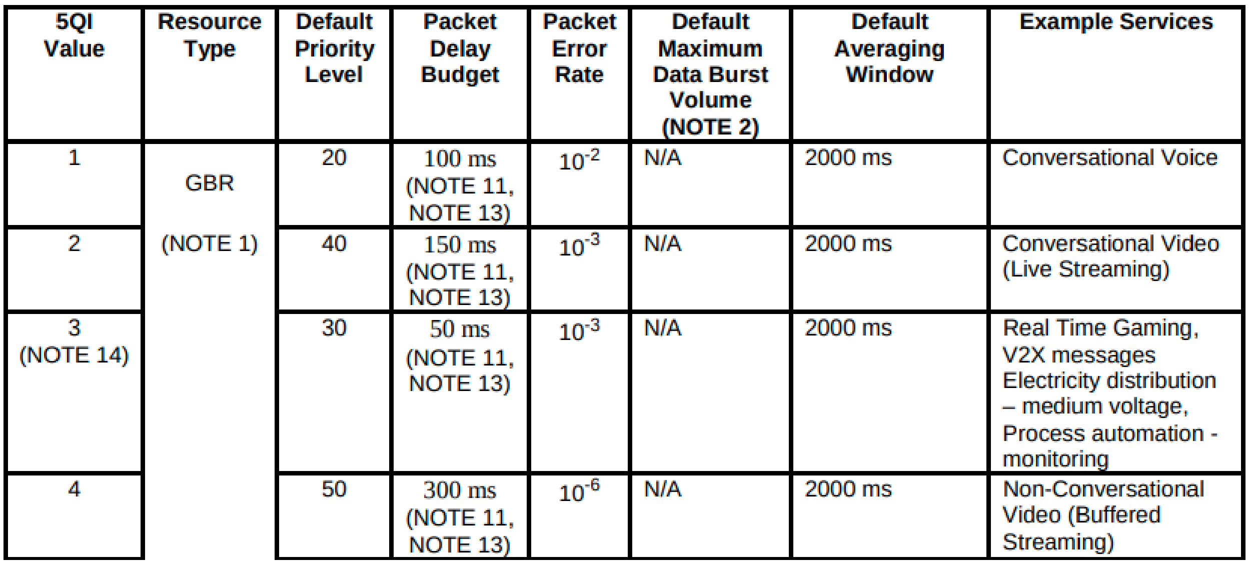
3.2 QoS Handling

The 5GC is in charge of the QoS control, not the radio-access network. QoS handling is essential for the realization of network slicing. For each UE, there are multiple PDU sessions, each having multiple QoS flows which are mapped to different DRBs. The IP packets are mapped to the QoS flows according to the QoS requirements, for example in terms of delay or required data rate. Each packet can be marked with a QoS Flow Identifier (QFI) to assist UL QoS handling.

The second step, mapping of QoS flows to DRBs, is done in the 5G RAN. Thus, the core network is aware of the service requirements, while the radio-access network only maps the QoS flows to DRBs. The QoS flow to DRB mapping is not necessarily a 1-to-1 mapping. It means that, multiple QoS flows can be mapped to the same DRB

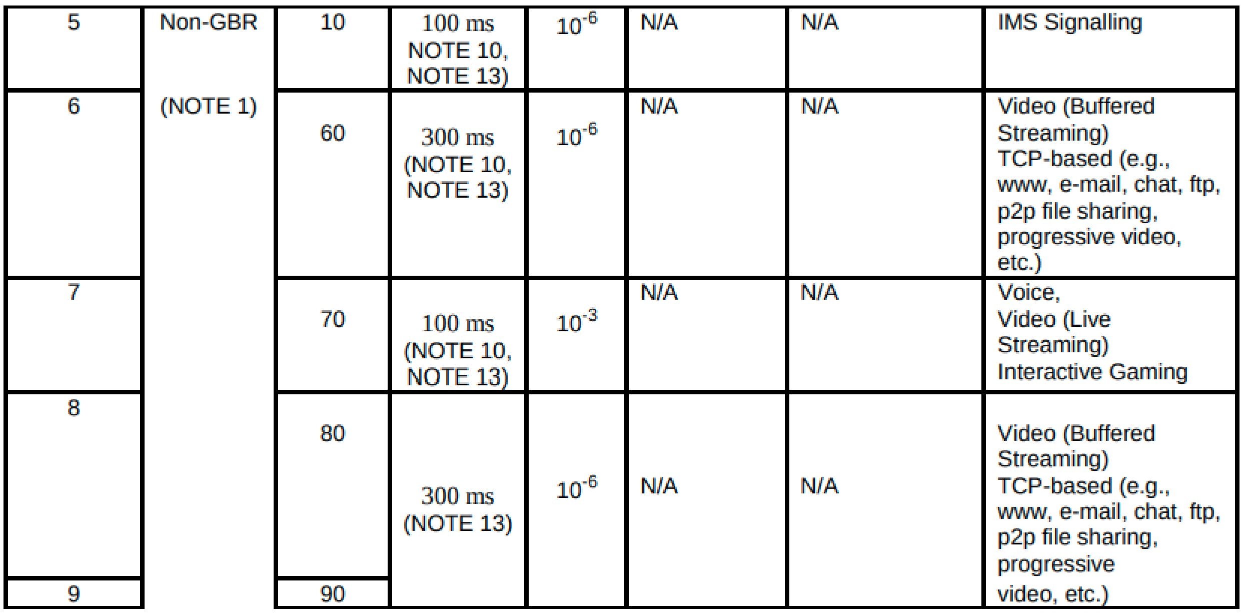
There are two ways of controlling the mapping from QoS flows to DRBs in the UL: reflective mapping and RRC configurations. In the case of reflective mapping, when connected to the 5G core network, the device observes the QFI in the DL packets for the PDU session. This provides the device with knowledge about which IP flows are mapped to which QoS flow and radio bearer. The device then uses the same mapping for the UL traffic. In the case of explicit mapping, the QoS flow to DRB mapping is configured in the device using RRC signaling.

3.2.1 Example QoS Requirements



5QI Value	Resource Type	Default Priority Level	Packet Delay Budget	Packet Error Rate	Default Maximum Data Burst Volume (NOTE 2)	Default Averaging Window	Example Services
1	GBR (NOTE 1)	20	100 ms (NOTE 11, NOTE 13)	10^{-2}	N/A	2000 ms	Conversational Voice
2		40	150 ms (NOTE 11, NOTE 13)	10^{-3}	N/A	2000 ms	Conversational Video (Live Streaming)
3 (NOTE 14)		30	50 ms (NOTE 11, NOTE 13)	10^{-3}	N/A	2000 ms	Real Time Gaming, V2X messages Electricity distribution – medium voltage, Process automation - monitoring
4		50	300 ms (NOTE 11, NOTE 13)	10^{-6}	N/A	2000 ms	Non-Conversational Video (Buffered Streaming)

Figure 3.1: GBR type QoS Flows.



5	Non-GBR (NOTE 1)	10	100 ms NOTE 10, NOTE 13)	10^{-6}	N/A	N/A	IMS Signalling
6		60	300 ms (NOTE 10, NOTE 13)	10^{-6}	N/A	N/A	Video (Buffered Streaming) TCP-based (e.g., www, e-mail, chat, ftp, p2p file sharing, progressive video, etc.)
7		70	100 ms (NOTE 10, NOTE 13)	10^{-3}	N/A	N/A	Voice, Video (Live Streaming) Interactive Gaming
8		80	300 ms (NOTE 13)	10^{-6}	N/A	N/A	Video (Buffered Streaming) TCP-based (e.g., www, e-mail, chat, ftp, p2p file sharing, progressive video, etc.)
9		90					

Figure 3.2: Non-GBR type QoS Flows.

3.3 SDAP Entity

The SDAP sublayer is made up of several SDAP entities. There is an SDAP entity established for every PDU session. An SDAP entity receives SDAP SDUs from the upper layers and submits the SDAP PDUs to the lower layer. It also receives SDAP PDUs from the upper layers and submits the SDAP SDUs to the upper layer.

A PDU session is a logical connection between a UE and the UPF (present in the core network). It is of various types mainly IPv4, IPv6, Ethernet etc. They enable transfer of data from the core network to the UE and vice-versa

3.4 Architecture (Specified by 3GPP)

Images cited from Bi (2018)

3.4.1 Structure

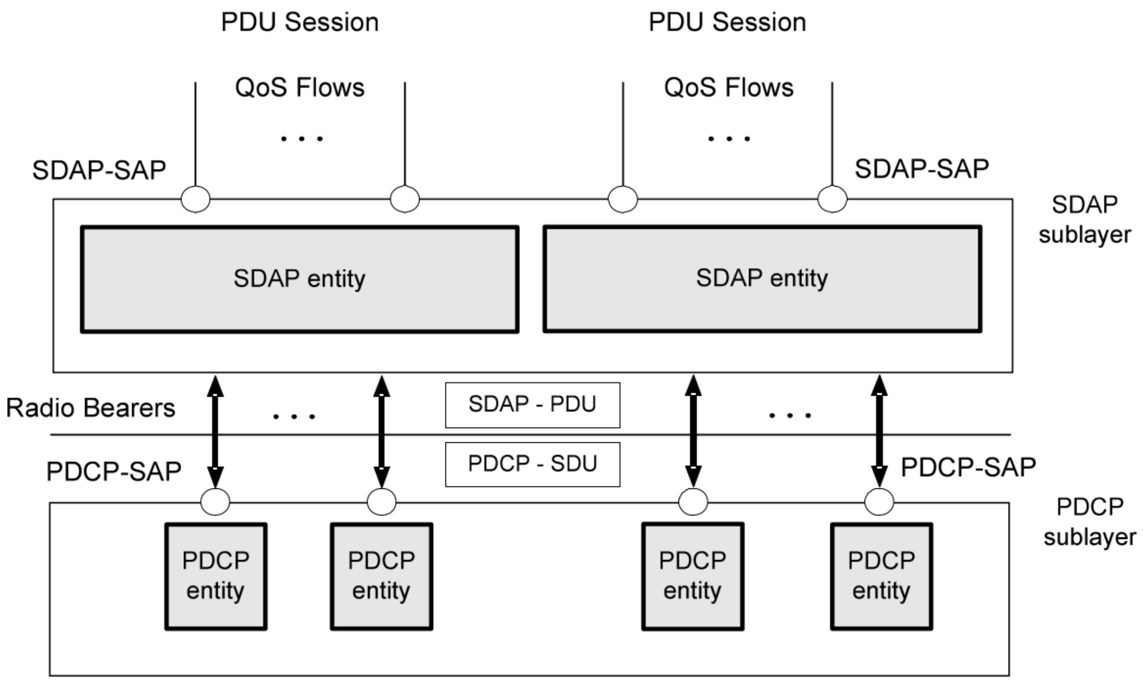
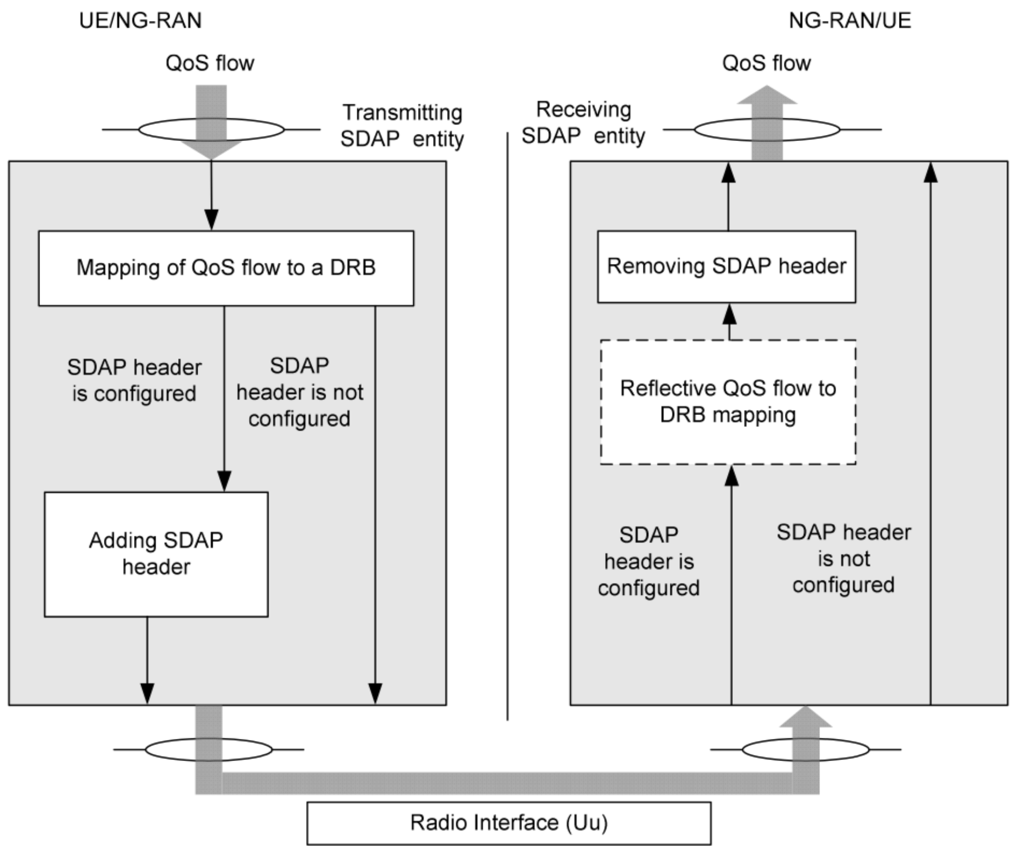


Figure 3.3: Structural View.

The PDU sessions are created in the 5GC. For each PDU session, a corresponding SDAP entity has to be initialised in order to cater to the QoS flows of that particular PDU session. The QoS flows are mapped to DRBs in the SDAP entity and then the data is passed onto the PDCP entity. There is a PDCP entity established for each DRB that has been configured. We are able to see the connection between the PDCP and SDAP as the PDCP SAP.

3.4.2 Functions


UE/NG-RAN
QoS flow
Transmitting SDAP entity
Mapping of QoS flow to a DRB
SDAP header is configured
SDAP header is not configured
Adding SDAP header
NG-RAN/UE
QoS flow
Receiving SDAP entity
Removing SDAP header
Reflective QoS flow to DRB mapping
SDAP header is configured
SDAP header is not configured
Radio Interface (Uu)
Figure 3.4: Functional View.

- Transfer of user plane data.
- Mapping between a QoS flow and a DRB for both DL and UL.
- Marking QoS flow ID in both DL and UL packets according to the formats given.
- Reflective QoS flow to DRB mapping for the UL SDAP data PDUs. It is done only on the UE side.

3.5 PDU types of the SDAP sub-layer

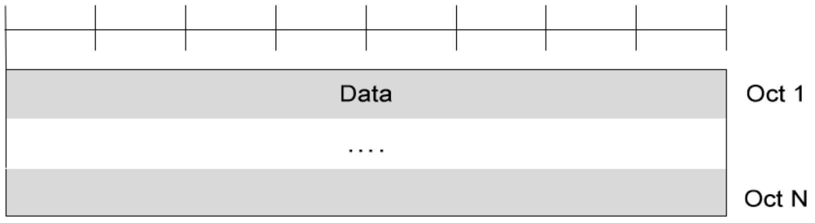


Figure 3.5: An example of an SDAP data PDU when the header is not configured.

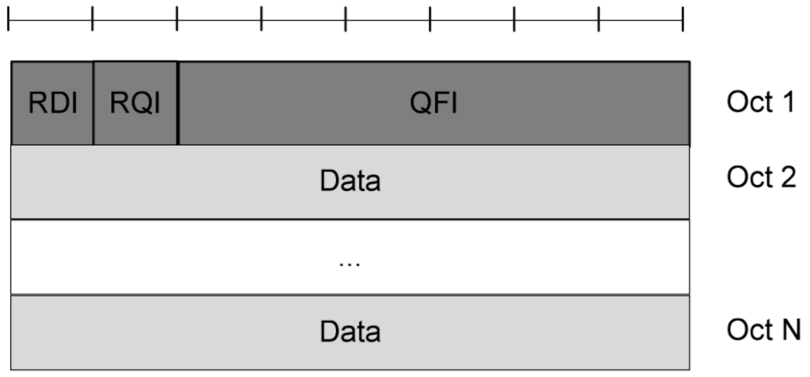


Figure 3.6: Format of an SDAP data PDU formed during Downlink and if the header is configured.

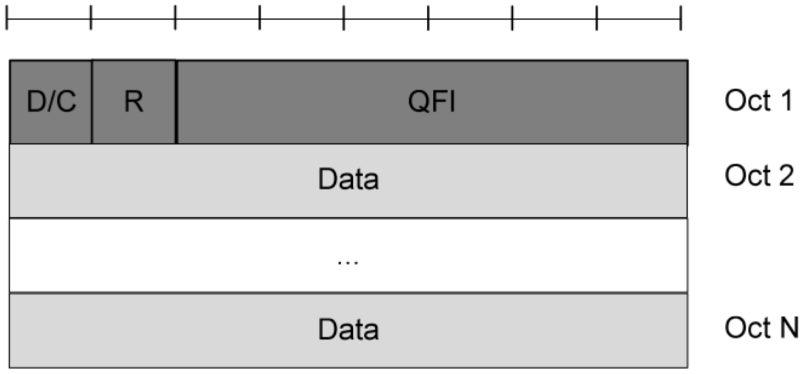


Figure 3.7: Format of an SDAP data PDU formed during Uplink and if the header is configured.

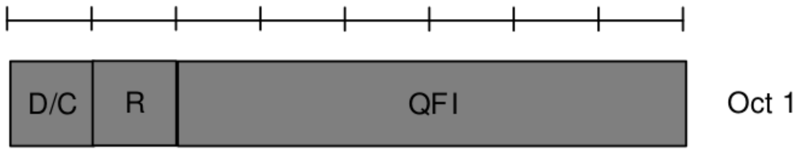


Figure 3.8: Format for an end marker control PDU.

CHAPTER 4

The SDAP API

4.1 Introduction

This chapter is about the SDAP application developed here at IITM 5G Lab according to 3GPP's specifications as described in the previous chapter. This API is meant to be placed in between the NG - interface and the PDCP sub-layer.

The main functionality of this API is the addition of the SDAP header. Each instance of this code can be treated as one SDAP entity and hence the code can be instantiated with different parameters in order to cater to each and every PDU session of an UE.

The API has been developed using the C programming language. (Ritchie, 1972–)

4.2 Functionality

- Transfer of UL data to the upper layers.
- Transfer of DL data to the lower layers.
- Addition of the SDAP header.
- Mapping of the QoS flows to the respective Data Radio Bearers.

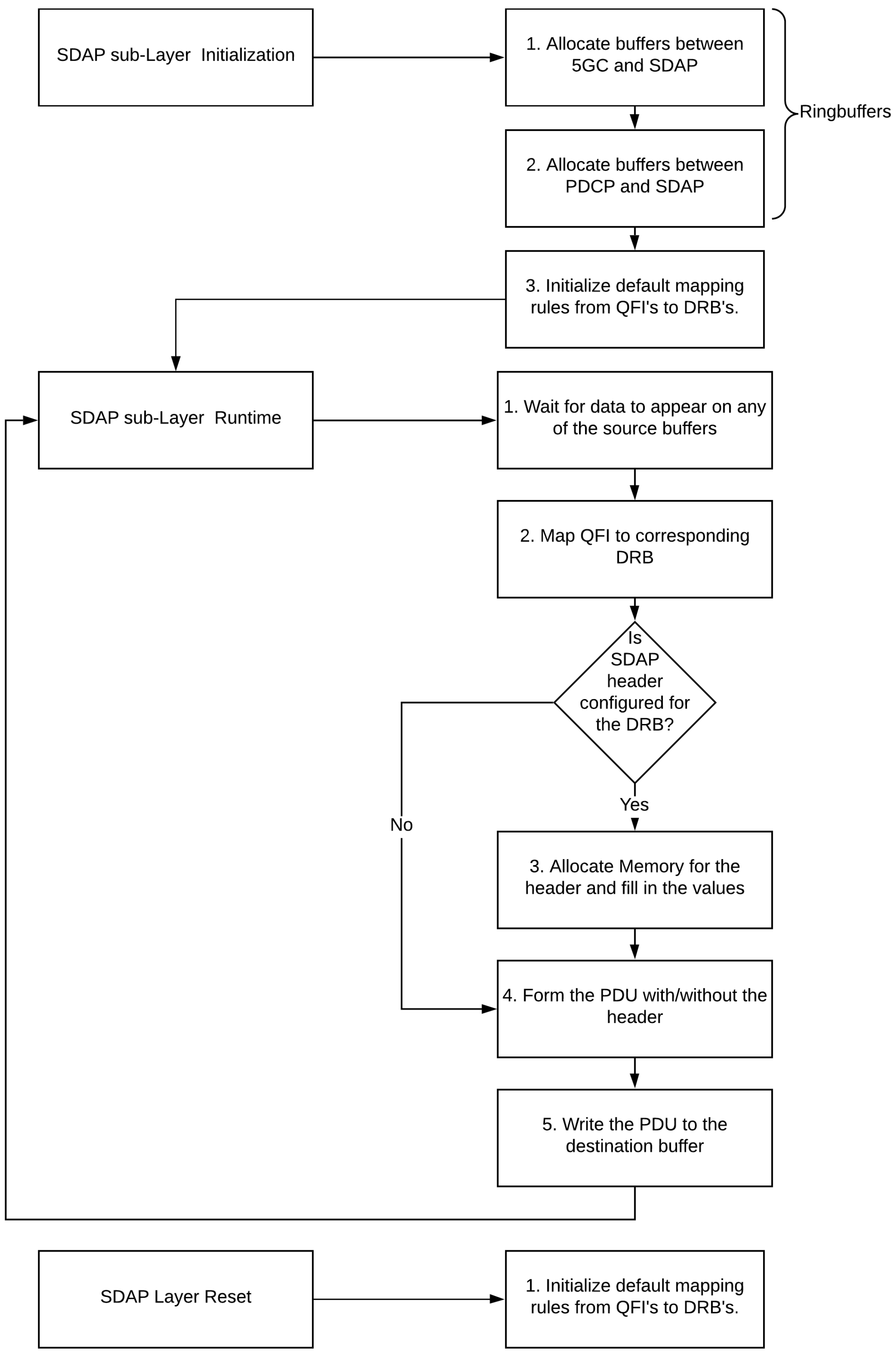

SDAP sub-Layer Initialization
1. Allocate buffers between 5GC and SDAP
2. Allocate buffers between PDCP and SDAP
Ringbuffers
3. Initialize default mapping rules from QFI's to DRB's.
SDAP sub-Layer Runtime
1. Wait for data to appear on any of the source buffers
2. Map QFI to corresponding DRB
Is SDAP header configured for the DRB?
Yes
No
3. Allocate Memory for the header and fill in the values
4. Form the PDU with/without the header
5. Write the PDU to the destination buffer
SDAP Layer Reset
1. Initialize default mapping rules from QFI's to DRB's.

Figure 4.1: Control of SDAP entity.

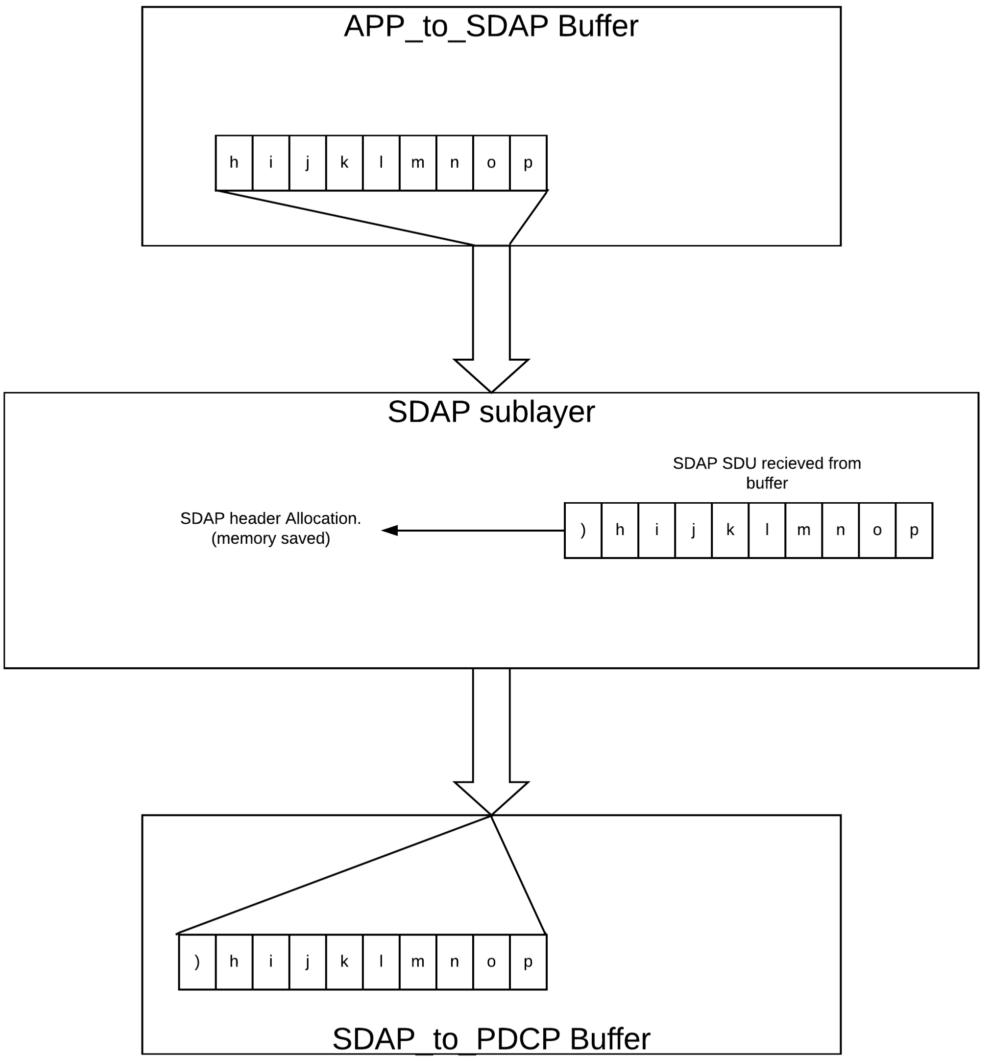


Figure 4.2: Sample DL data flow in an SDAP entity.

This is a depiction of a sample data flow happening in the SDAP Sub-layer. The data is being received from the source buffer and the header is added to it in the sub-layer. Later the packet is being stored in the destination buffer.

4.3 Code fragments and performance

4.3.1 Header Addition

```
uint8_t *make_UL_PDU(uint32_t sdu_len, ringBuffer *src_buffer,
    ringBuffer *dest_buffer, bool header_present, uint8_t D_C, uint8_t
     QFI)
{
    uint8_t *   pdu;
    uint8_t *   sdu;
    uint8_t *   temp;

    ul_header_p  header;

    if(header_present)
    {
        pdu = (uint8_t *) calloc(1,sdu_len + 1);
        if(!pdu) printf("PDU Memory allocation failed");
        else
        {
            if(D_C)
            {
                sdu = pdu + 1;
                readFromBuffer(src_buffer, sdu_len, sdu);
            }
            header = SDAP_ul_header_fill ( D_C, QFI );
            temp = (uint8_t  *) header;
            *pdu = *temp;
            writeToBuffer(dest_buffer, sdu_len + 1, pdu);
        }
    }
    else
    {
        pdu = (uint8_t *) calloc(1,sdu_len);
        if(!pdu) printf("PDU Memory allocation failed");
        else
        {
```

```c
            sdu = pdu;
            readFromBuffer(src_buffer, sdu_len, sdu);
            writeToBuffer(dest_buffer, sdu_len, pdu);
        }
    }
    return pdu;
}
```

Description

The above function creates the SDAP PDU by taking in the SDAP SDU and other header parameters as input. Memory is allocated according to whether the header is configured for the following DRB or not. After the PDU is created it is written the the destination DRB(buffer).

Timing Analysis

Execution time taken by this function to finish the following tasks is around 10 μs

4.3.2 Header Removal

```c
uint8_t * make_UL_SDU (uint32_t pdu_len, ringBuffer *src_buffer, ringBuffer             *dest_buffer, bool header_present, ul_header_p header)
{
    uint8_t *   pdu;
    uint8_t *   sdu;
    ul_header_p temp;
    if(header_present)
    {
        pdu = (uint8_t *) calloc(1,pdu_len);
        if(!pdu) printf("SDU Memory allocation failed");
        else
        {
            readFromBuffer(src_buffer, pdu_len, pdu);
            sdu = pdu + 1;
            header = calloc(1,sizeof(ul_header_t));
            temp = (ul_header_p) pdu;
            *header = *temp;
            writeToBuffer(dest_buffer, pdu_len - 1, sdu);
        }
    }
    else
    {
        pdu = (uint8_t *) calloc(1,pdu_len);
        if(!pdu) printf("SDU Memory allocation failed");
        else
        {
            readFromBuffer(src_buffer, pdu_len, pdu);
            sdu = pdu;
            writeToBuffer(dest_buffer, pdu_len, sdu);
        }
    }
    return sdu;
}
```

Description

The above function creates the SDAP SDU by taking in the SDAP PDU input. header is extracted according to whether the header is configured for the following DRB or not. After the SDU is created it is written the the destination QoS(buffer).

Timing Analysis

Execution time taken by this function to finish the following tasks is around 21 μs

4.3.3 Mapping

```
#define MAX_QOS_ENT 64
#define MAX_DRB_ENT 29
#define DEFAULT_DRB 0
```

```c
typedef struct gNB_map_ent
{
    uint8_t      QFI;
    uint8_t      DRB;

}   gNB_map_t, *gNB_map_p;
```

Description

These are some of the parameters related to the mapping of the QoS flows to DRBs and vice versa.

CHAPTER 5

ASN.1

5.1 Introduction

ASN.1 is a universal language used for defining data transmitted by telecommunications protocols, regardless of language implementation and physical representation of these data, whatever the application, whether complex or very simple. ITU (2014)

5.2 Encoding rules

ASN.1 has sets of rules specifying how messages must be "encoded" for communication with machines running other languages. Each set of "encoding rules" has specific characteristics, such as compactness or decoding speed, which make it suitable for particular environments. All of the encoding rules are able to represent any messages we would like to exchange. Walkin (2010)

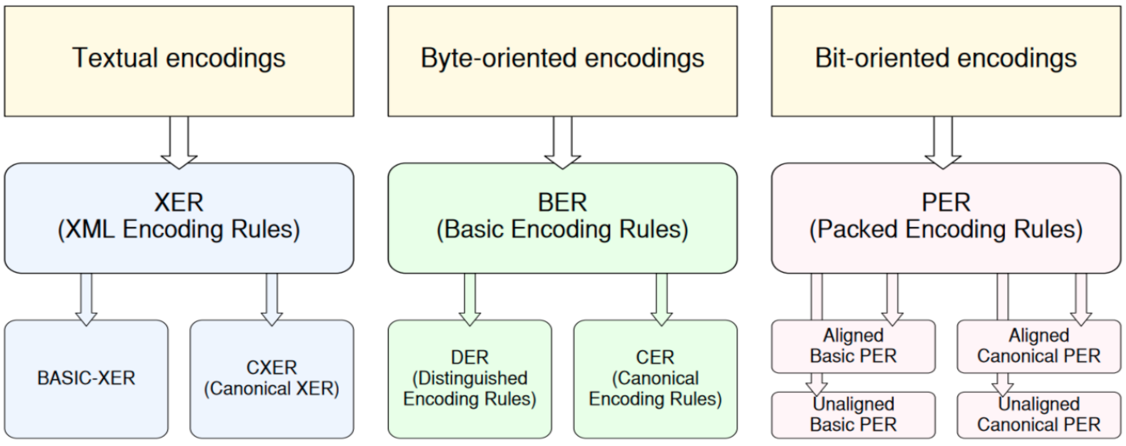
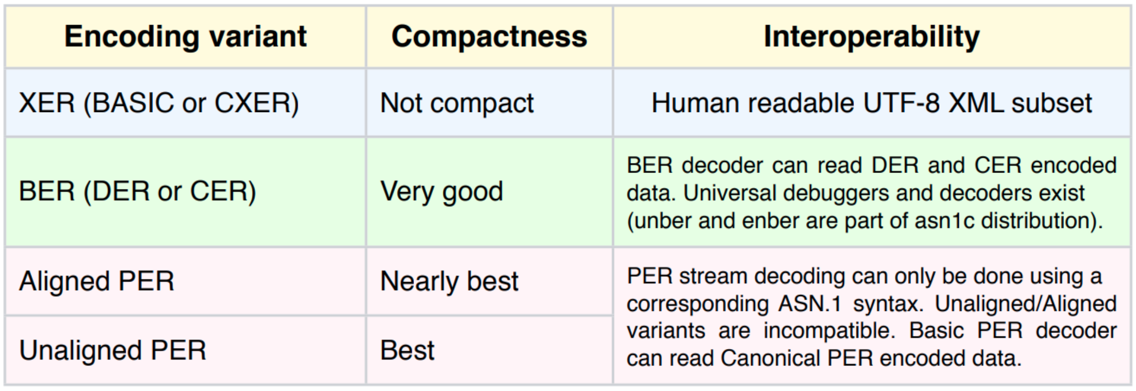


Figure 5.1: Encoding rules.

The ASN.1 standard defines a variety of methods to encode data. Depending on space, interoperability and efficiency requirements, a protocol designer selects textual, binary-based or compact bit-packed encoding rules.



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	

Figure 5.2: Comparison of different encoding rules.

5.3 ASN1 compilers

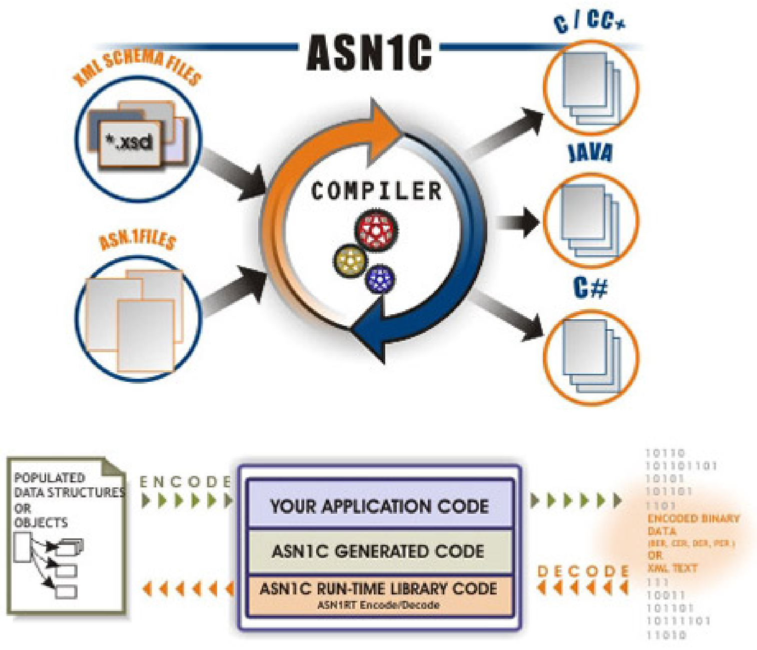


Figure 5.3: ASN1C Workflow.

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.

5.3.1 obj-sys compiler

This ASN.1 compiler converts ASN.1 and/or XML schema source files into C, C++, C#, Java, or Python source code. This code can then be used by developers 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). obj sys (2020)

5.3.2 OSS Noklava compiler

The OSS® ASN.1 Tools for C is a complete development toolkit for rapidly building applications using ASN.1. This compiler features 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 an assortment of utilities to ease and speed app development. Our ASN.1 products support LTE Advanced Pro including NB-IoT, C-V2X and LTE-M. OSS products support the 3GPP 5G specifications. OSS-Noklava (2020)

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

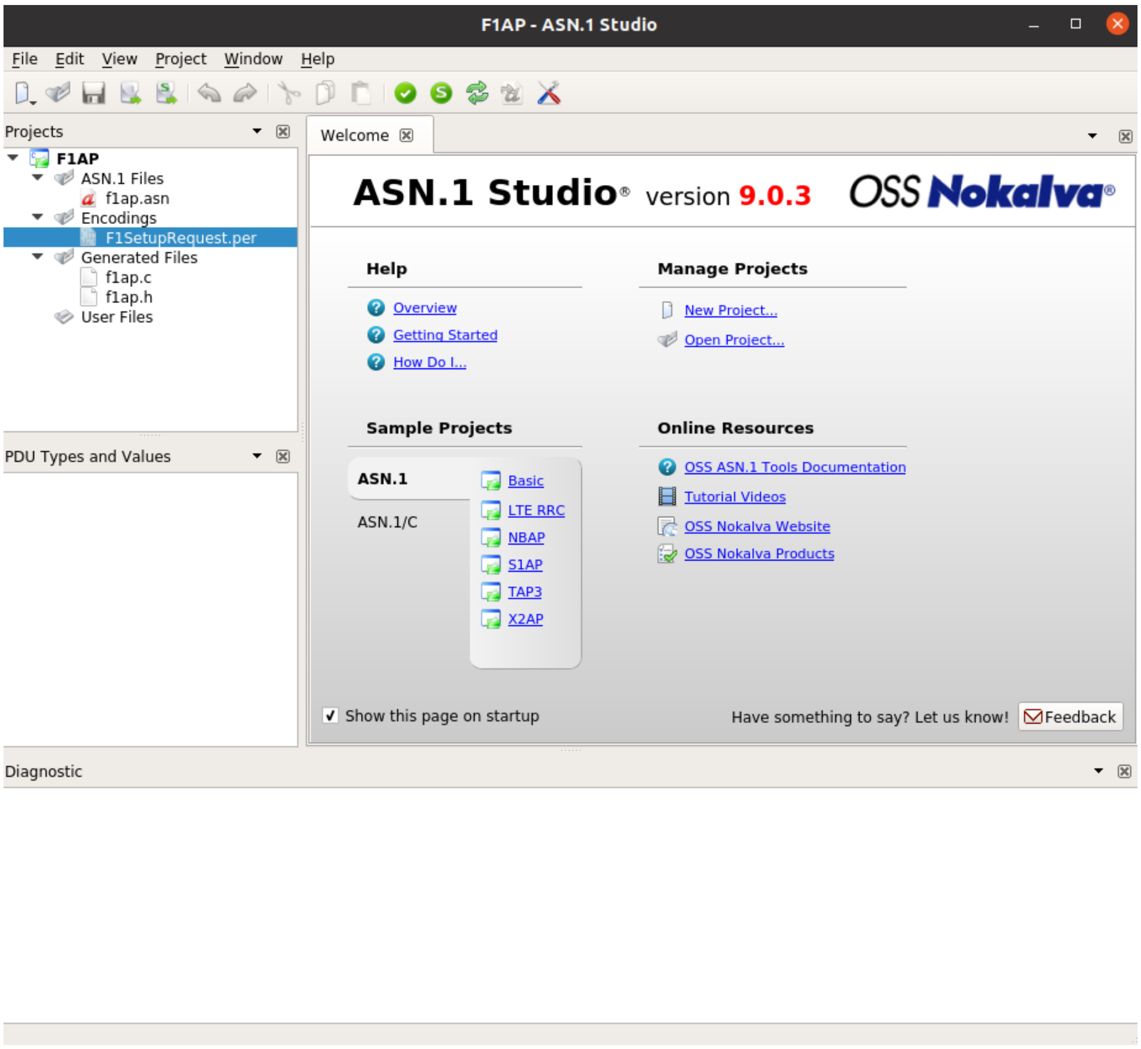


Figure 5.4: ASN1C Studio.

It makes checking constraints for IEs easier which in turn helps in faster debugging. Figure 5.6 depicts the F1 Setup Request IE with filled IEs (first message transferred between gNB-DU and gNB-CU).

5.4 F1-Setup Encoding

5.4.1 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.

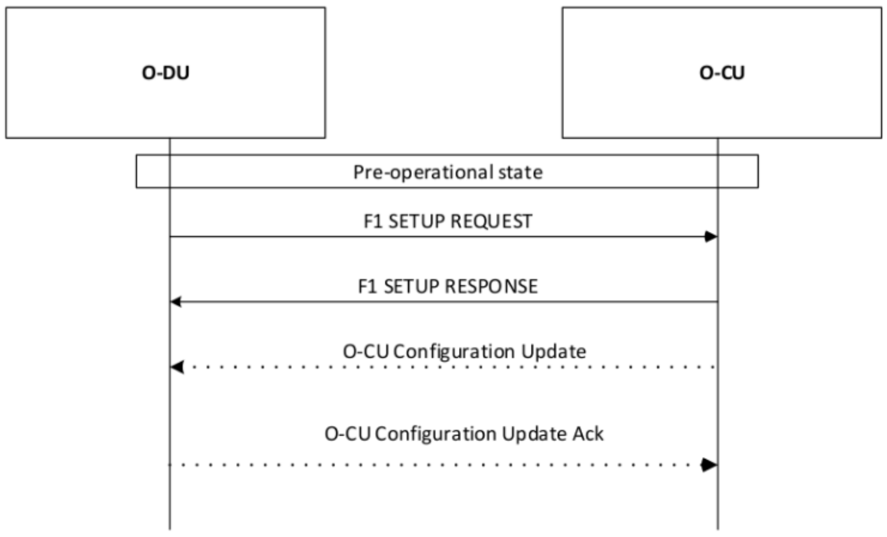
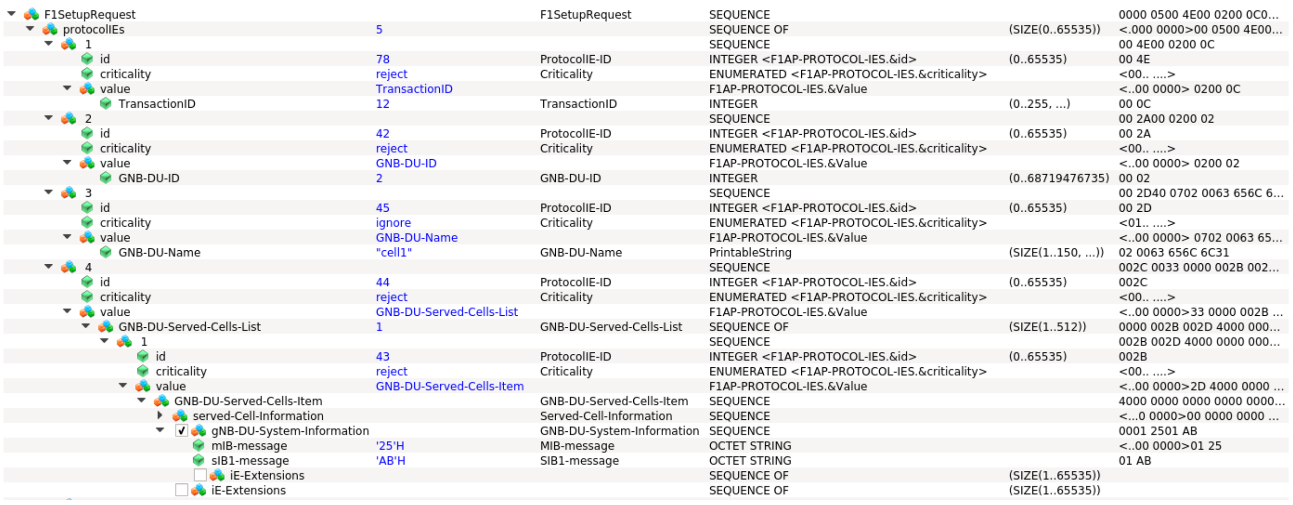


Figure 5.5: F1-Setup Request.

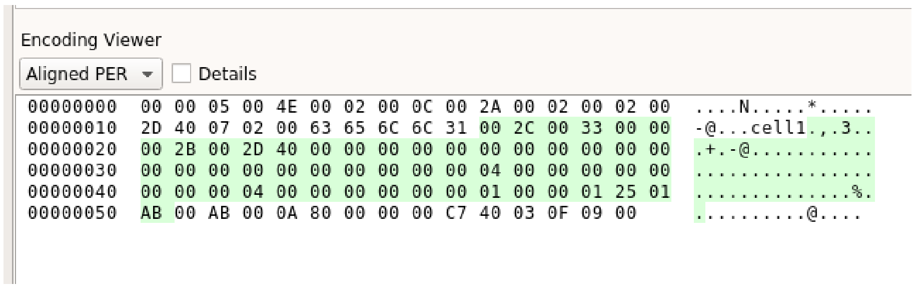
5.4.2 Procedure

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...
  protocolIEs                         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.&id>              (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.&id>              (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.&id>              (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.&id>              (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.&id>              (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                                                      SEQUENCE OF                                  (SIZE(1..65535))
              ☐ iE-Extensions                                                        SEQUENCE OF                                  (SIZE(1..65535))
```

Figure 5.6: 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      ..........@....
```

Figure 5.7: Encoded Data in Hex.

5.4.3 Decoded Data

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

```
value F1SetupRequest ::=
{
  protocolIEs
  {
    {
      id 78,
      criticality reject,
      value TransactionID : 12
    },
    {
      id 42,
      criticality reject,
      value GNB-DU-ID : 2
    },
    {
      id 45,
      criticality ignore,
      value GNB-DU-Name : "cell1"
    },
    {
      id 44,
      criticality reject,
      value GNB-DU-Served-Cells-List :
      {
        {
          id 43,
          criticality reject,
          value GNB-DU-Served-Cells-Item :
          {
            served-Cell-Information
            {
              nRCGI
              {
                pLMN-Identity '000000'H,
                nRCellIdentity '00000000 00000000 00000000 00000000
    000 ...'B
              },
              nRPCI 0,
              servedPLMNs
              {
                {
                  pLMN-Identity '000000'H
                }
              },
              nR-Mode-Info fDD :
                {
                  uL-NRFreqInfo
                  {
                    nRARFCN 0,
                    freqBandListNr
                    {
                      {
                        freqBandIndicatorNr 1,
                        supportedSULBandList
                        {
                          {
                            freqBandIndicatorNr 1
                          }
                        }
                      }
                    }
                  },
                  dL-NRFreqInfo
                  {
                    nRARFCN 0,
                    freqBandListNr
                    {
                      {
                        freqBandIndicatorNr 1,
                        supportedSULBandList
                        {
                          {
                            freqBandIndicatorNr 1
                          }
                        }
                      }
                    }
                  },
                  uL-Transmission-Bandwidth
                  {
                    nRSCS scs15,
                    nRNRB nrb11
                  },
                  dL-Transmission-Bandwidth
                  {
                    nRSCS scs15,
                    nRNRB nrb11
                  }
                },
              measurementTimingConfiguration '00'H
            },
            gNB-DU-System-Information
            {
              mIB-message '25'H,
              sIB1-message 'AB'H
            }
          }
        }
      }
    },
    {
      id 171,
      criticality reject,
      value RRC-Version :
      {
        latest-RRC-Version '000'B,
        iE-Extensions
```

```
        {
          {
            id 199,
            criticality ignore,
            extensionValue OCTET STRING : '0F0900'H
          }
        }
      }
    }
  }
}
```

5.4.4 Conclusion

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

APPENDIX A

Socket Programming in C/C++

A.1 Introduction

Socket programming helps in transferring data between two nodes on a network. There are two types of nodes required to initiate data transfer between them. The two node types are a server and a client. Server creates a socket capable of listening on a particular port whereas the client forms a socket which reaches out to the server from a particular port on its side. This appendix chapter has been cited from Sinha

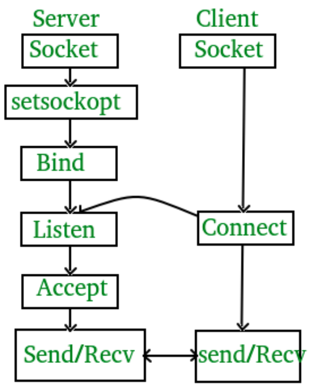


Figure A.1: Describes various states followed by the server and client.

A.2 API's for server

A.2.1 Socket creation

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

sockfd: socket descriptor, it is an integer similar to that of a file descriptor used to read or write data.

domain: it is an integer used to set the communication domain. example, AF_INET (IPv4 protocol) , AF_INET6 (IPv6 protocol)

type: used to set the communication type (Transport layer protocol).

- SOCK_STREAM: TCP(reliable, connection oriented)
- SOCK_DGRAM: UDP(unreliable, connection-less)

protocol: Selects the network layer protocol for the transmission. 0 for IP protocol.

A.2.2 Bind

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

The bind function then binds the socket to the address and port number specified in the addr data structure.

A.2.3 Listen

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

It makes the socket in the server to wait for any incoming connections. The backlog parameter helps to set the maximum number of incoming connections to be handled simultaneously. If the number of connections exceeds the backlog number, the incoming connection is refused with an error message.

A.2.4 Accept

```
int new\_socket= accept(int sockfd, struct sockaddr *addr, socklen_t
                                                            *addrlen);
```

Out of all the pending connections, the first connection request is catered to by creating a new socket descriptor for that connections thereby enabling the transfer of data through that descriptor.

A.3 API's for client

A.3.1 Socket creation

Exactly same as that of server's socket creation

A.3.2 Connect

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

It is used to connect the socket mentioned in the sockfd descriptor to the address specified in addr. Server's connection parameters are mentioned in the addr custom structure.

APPENDIX B

Timing Analysis

B.1 Introduction

Timing is a critical criteria when it comes to a protocol stack development because of the various latency constraints that have to be met by the executing program.

A simple way to measure the amount of time taken by any segment of code will be by calculating the number of clock cycles taken by that segment to finish execution.

We use the time.h header file and the various functions and constants present in the C language in order to accomplish this task. This appendix chapter has been cited from Kumar

B.2 <time.h>

The time.h header defines four variable types, two macro and various functions for manipulating date and time. Some of the relevant features useful to us are listed below.

clock_t

This is a type suitable for storing the processor time.

CLOCKS_PER_SEC

This macro represents the number of processor clocks per second.

clock_t clock(void);

Returns the processor clock time used since the beginning of an implementation defined era (normally the beginning of the program).

B.3 Calculation

```
    time_spent = (double) (end - begin)/CLOCKS_PER_SEC;
```

B.4 Sample Program

```
#include <bits/stdc++.h>
using namespace std;

void fun()
{
    for (int i=0; i<10; i++);
}

int main()
{
    clock_t start, end;

    start = clock();

    fun();

    end = clock();

    double time_taken = double(end - start) / double(CLOCKS_PER_SEC);
    cout << "Time taken by program is : " << fixed
         << time_taken << setprecision(5);
    cout << " sec " << endl;
    return 0;
}
```

REFERENCES

1. **Bi, H.** (2018). Lte; 5g; evolved universal terrestrial radio access (e-utra) and nr; service data adaptation protocol (sdap) specification (3gpp ts 37.324 version 15.1.0 release 15). URL `https://www.etsi.org/deliver/etsi_ts/137300_137399/137324/15.01.00_60/ts_137324v150100p.pdf`.

2. **Dahlman, E.**, **S. Parkvall**, and **J. Sköld**, *5G NR: The Next Generation Wireless Access Technology*. Elsevier, 2018.

3. **ITU** (2014). Introduction to asn.1. URL `https://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspxf`.

4. **Kumar, V.** (). Measure execution time with high precision in c/c++. URL `https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/`.

5. **obj sys** (2020). Asn1c asn.1 compiler. URL `https://www.obj-sys.com/products/asn1c/index.php`.

6. **OSS-Noklava** (2020). Asn.1 made simple — what is asn.1?r. URL `https://www.oss.com/asn1/resources/asn1-made-simple/introduction.html`.

7. **Ritchie, D.** (1972–). The C programming language.

8. **Sinha, A.** (). Socket programming in c/c++. URL `https://www.geeksforgeeks.org/socket-programming-cc/`.

9. **Stallman, R.** (2020). *GNU make*. URL `https://www.gnu.org/software/make/manual/html_node/`.

10. **Walkin, L.** (2010). Open source asn.1 compiler asn1c quick start sheet. URL `http://lionet.info/asn1c/asn1c-quick.pdf`.