Master Thesis

Optimized Header Compression for Real-time Communication in a 5G Communication Network

Zihao Tao

Delft University of Technology

TU Delft

Optimized Header Compression for Real-time Communication in a 5G Communication Network

by

Zihao Tao

to obtain the degree of

Master of Science

in Electrical Engineering

Track Wireless Communication and Sensing

at the Delft University of Technology,

to be defended publicly on Tuesday August 22, 2023 at 10:00 AM.

Student Number:	5496780	
Thesis Committee:	Ir. Rogier Noldus	TU Delft, Ericsson
	Dr. Remco Litjens, MSc	TU Delft, TNO
	Dr. Qing Wang	TU Delft
Project Duration:	11, 2022 - 08, 2023	

Preface

During my two-year Master's degree study at the Delft University of Technology, I built a solid background in 5G technology, wireless communication, and computer networks. In this case, I decided to conduct my thesis project in these related areas. This thesis project summarizes my Master's study at TU Delft in the Wireless Communication and Sensing (WiCOS) track and The Network Architectures and Services (NAS) Group.

Firstly, I would like to thank Ir. Rogier Noldus as the daily supervisor of this thesis project. It's very kind of him to guide me to study. His patience, professionalism, and working efficiency in this thesis supervision have impressed me so much. It's my honor for me to work with him.

Secondly, I would like to thank Dr. Remco Litjens, MSc for being my thesis advisor, guiding my thesis project, and giving the necessary feedback. I'm pleased to learn from him to improve my simulation results and thesis.

Thirdly, I would like to thank Dr. Qing Wang for being a committee member of my thesis defense. It's my pleasure to obtain advice from him.

Finally, I would like to thank my parents for supporting my study in the Netherlands. My achievements in this degree study are attributed to their devotion and encouragement.

Abstract

With the development of 5G/6G communication technology, an increasing number of communication protocols for data transmission are created and implemented. At the same time, the redundancy of packet headers has become a matter to be optimized when transferring packets across different protocol layers, which causes high resource occupancy, low transmission efficiency and high latency, especially for real-time communications like Voice over IP (VoIP).

The currently standardized Robust Header Compression (RoHC) Algorithm seeks to provide an approach to compressing the protocol headers inscribed in data packets to decrease the amount of data transmission without reducing the communication quality. However, the existing RoHC algorithm is not efficient enough for the requirements of present header compression, as it's initially designed for specific headers like RTP, UDP, and IP. The packet headers must be optimized over the protocol stack for the real-time data stream and the GPRS Tunneling Protocol (GTP) tunnel sections in the 5G network.

An Optimized Header (OH) Compression Algorithm based on the existing RoHC algorithm is proposed in this thesis to compress the headers over the 5G protocol stack in a coordinated fashion for real-time transmission and GTP tunnels. Besides, further development of the OH algorithm is done to optimize protocol headers by merging duplicate header fields for both multiple QoS flows and multi-layer Protocol Data Unit (PDU) flows in the same PDU session. The overall optimization is conducted in the 5G System (5GS) over F1-U and N3 reference points.

The simulation results indicate that the OH algorithm improves the average header compression rate by 45% compared with the RoHC algorithm, as demonstrated via scenario-based 5GS simulations. The average number of transported bits is decreased by 30%, 15% and 26% for Opus coding audio stream, H.264-Opus coding video-audio stream and BLE Version 4.2 triple packets for smartwatch telemedicine stream, respectively, compared with the RoHC algorithm. Besides, the average algorithm execution latency is decreased by 2 μs compared with the RoHC algorithm. Moreover, the multi-stream and multi-layer optimization for the OH algorithm brings about a higher header compression rate and lower execution time in specific scenarios. The overall results indicate that the OH algorithm is capable of compressing the packet size effectively, decreasing the number of transported bits, and shortening the execution time, resulting in higher transmission efficiency and lower latency for VoIP services.

Acronyms

5G-NR 5G New Radio

5G-RAN 5G Radio Access Network

5GC 5G Core

5GS 5G System

5QI 5G QoS Identifier

AF Application Function

AMF Access and Mobility Management Function

AN Access Network

AS Access Stratum

BBU Baseband Unit

BH Backhaul

CID Context Identifier

CN Core Network

CP Control Plane

CRC Cyclic Redundancy Check

CU Centralised Unit

CUPS Control and User Plane Separation

DL Downlink

DN Data Network

DRB Data Radio Bearer

DU Distributed Unit

DYN DYNAMIC packet

E-UTRAN Evolved-UMTS Terrestrial Radio Access Network

eNodeB evolved Node B (4G base stations)

EPC Evolved Packet Core

ePDG evolved Packet Data Gateway

F1-U F1 User plane interface

F1-C F1 Control plane interface

F1AP F1 Application Protocol

FO First Order state (compressor)

GFBR Guaranteed Flow Bit Rate

GGSN Gateway GPRS Support Node

gNodeB-CU gNodeB-Centralised Unit

gNodeB-CU-CP gNodeB-Centralised Unit-Control Plane

gNodeB-CU-UP gNodeB-Centralised Unit-User Plane

gNodeB-DU gNodeB-Distributed Unit

gNodeB Next Generation Node B

GTP-C GPRS Tunnelling Protocol for Control plane

GTP-U GPRS Tunnelling Protocol for User plane

IR Initiation and Refresh state or IR packet

L2 Layer-2

L3 Layer-3

LSB Least Significant Bits

ML Machine Learning

MME Mobility Management Entity

MOH Multi-stream Optimized Header Algorithm

MTU Maximum Transmission Unit

N1 Non-Access Stratum interface

NAS-MM NAS-Mobility Management

NAS-SM NAS-Session Management

NAS Non-Access Stratum

NF Network Function

NG-RAN Next Generation-Radio Access Network

NR New Radio

O-mode bidirectional Optimistic mode

OH Optimized Header compression algorithm

PDU Protocol Data Unit

PTP Point To Point

QFI QoS Flow Identifier

QoE Quality of Experience

QoS Quality of Service

R-mode bidirectional Reliable mode

(R)AN (Radio) Access Network

RoHC Robust Header Compression

RRC Radio Resource Control

RTP Real-Time Protocol

SA Standalone

SAP Service Access Point

SM Session Management

SMF Session Management Function

SN Sequence Number

SRB Signalling Radio Bearers

SSRC Sending Source in RTP header

TEID Tunnel Endpoint Identifier

U-mode Unidirectional mode

UE User Equipment

UL Uplink

UP User Plane

UPF User Plane Function

URLLC Ultra Reliable Low Latency Communication

VOH Vertical Optimized Header compression algorithm

VoIP Voice over IP

Contents

1

Introduction

This chapter gives the overview background introduction for the presented research study. Section 1.1 introduces the development of 5G and its future requirements. In Section 1.2, the present problem of the redundant packet headers is mentioned to bring out the essence of the research study. Besides, the RoHC algorithm is discussed for its shortcomings in the present requirements of header compression. Motivation and challenges are mentioned respectively in Section 1.3. Three research objectives are proposed in Section 1.4. Section 1.5 gives a brief description of the methodology for this research. The thesis outline is finally described in Section 1.6.

1.1. Development of 5G Communication and Basic Concepts

1.1.1. Development of Wireless Communication

The First Generation (1G) of mobile wireless communication employed analog technology for public phone service. Text messaging was made possible by the Second Generation (2G) technology, which was digital-based. According to a study by Advanced Computer Network Lab, the main differences between 1G and 2G cellular networks (i.e., GSM in 2G) are that the former were almost entirely analog and lacked many features. At the same time, the latter was digital and could encrypt communication to prevent eavesdropping, permit detection and correction of identified errors, provide clear voice reception, and authorize channels that users can share simultaneously [1].

The Internet service was first added to the 2.5G General Packet Radio Service (GPRS). The Third

Generation (3G) of technology saw a considerable rise in data transmission speed and capacity, as well as the development of facilities for multimedia support. In the Fourth Generation (4G) technology, the size of the bandwidth was increased while the transmission cost was reduced [2]. The Fifth Generation (5G) of mobile wireless communication satisfies a lot of requirements, including faster transmission speed, higher reliability, and massive connections, which brings about more chances for development in the future [1].

The International Telecommunication Union (ITU) [3] emphasizes three main features of the 5G system: enhanced Mobile Broadband (EMBB) [4], Massive Machine-Type Communication (MMTC) [5], and Ultra-Reliable and Low-Latency Communications (URLLC) [6]. Effective transmission of packets with tiny payloads in packet-switched communication networks will probably be necessary for MMTC and URLLC. Additionally, the Internet of Things (IoT) paradigm will demand the effective transmission of a significant number of individual small-data sensor readings for a variety of applications, including industrial [7], medical [8], and tactile internet [9] respectively, which calls for the timely transmission of a significant volume of discrete small-data sensor readings [10].

User Equipment (UE), Radio Access Network (RAN), Core Network (CN), and IP Multimedia Subsystem are all parts of the overall 3GPP system architecture, which is being developed by 3GPP SA Working Group 2 (SA2). The group outlines the critical components of the system architecture and how they interact. It also specifies the essential functions and information flow between these components. The initial release of 5G-Advanced is Rel-18. Following the characterization and prioritization of Rel-18 material at the SA#94-e meeting in December 2021, SA2 began to work on Rel-18 Stage-2 in February 2022. Some Rel-18 study materials are based on new requirements, while others result from failing to meet Rel-17 needs requiring more work [11, 12]. The plan for Release 17 and 18 is illustrated in Figure 1.1 and Figure 1.2, respectively.

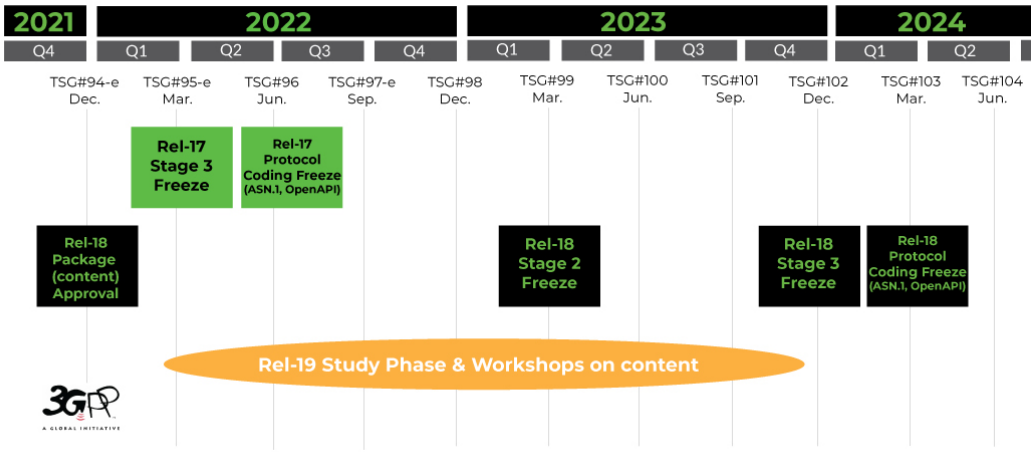


Figure 1.1: Rel-17 and Rel-18 [11]

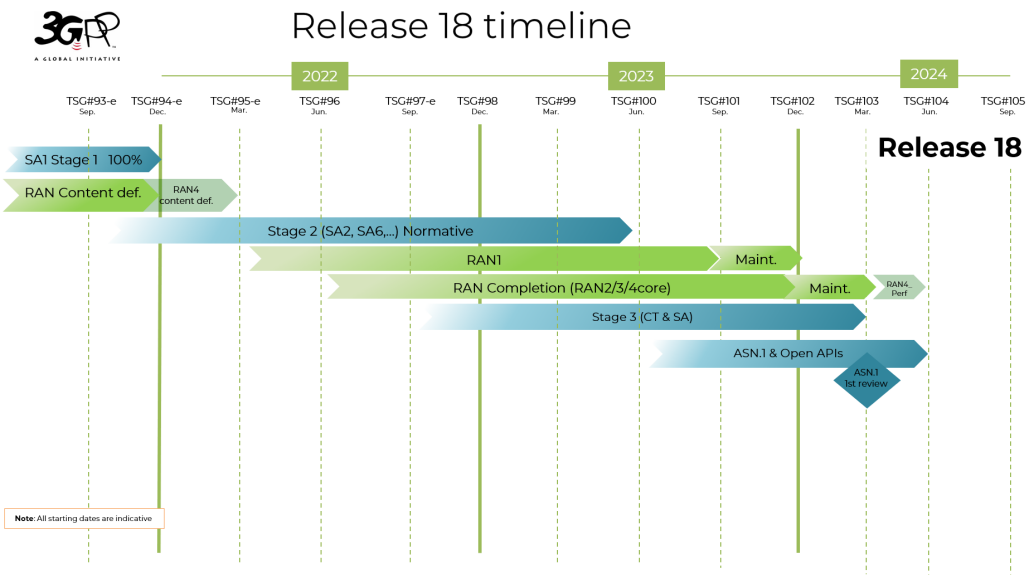


Figure 1.2: Rel-18 Timeline [13]

1.1.2. Basic Concepts of Protocols

The fourth version of the Internet Protocol (IPv4) is used to identify devices on a network by their IP addresses. It employs 32-bit addresses, written as a string of four integers separated by dots (for example, 192.168.0.1), and is the most popular layer 3 protocol on the Internet. It runs at the network layer of the Open Systems Interconnection (OSI) paradigm. Every device connected to the Internet has an IP address. There are a total of over 4.3 billion unique IP addresses [14]. This quantity, however, is no longer adequate to meet the rising demand for IP addresses due to the explosion in connected devices, which prompted the creation and adoption of IPv6, which uses 128-bit addresses and offers a virtually limitless number of unique addresses [15].

User Datagram Protocol (UDP) is a communication protocol used in computer networks for transmitting and receiving datagrams (packets) through the Internet Protocol. It runs at the transport layer of the OSI paradigm and is a connection-less protocol. UDP doesn't offer error checking, dependability, or sequencing. These tasks are instead handled by the application layer. Online gaming, video streaming, and VoIP are just a few examples of time-sensitive applications where UDP is frequently utilized since it does not have the complexity associated with the more sophisticated Transmission Control Protocol (TCP) [16]. One of its main advantages is that UDP has low latency because of its simple design. Since there is no system for mistake detection or recovery, there is also a higher risk of data loss or corruption. Therefore, UDP is typically not appropriate for applications that demand great data integrity or reliability [17]. But still, UDP is used for VoIP.

Real-time Transport Protocol (RTP) transmits real-time data over IP networks, including audio and video. It is frequently used in real-time transmission-required applications, including video conferencing, streaming media, and online gaming. In addition to payload type identification, sequence

numbering, time-stamping, and delivery monitoring, RTP offers end-to-end delivery services for real-time data. It can be used with various applications because it supports several codecs for audio and video coding, including compression [18]. One of RTP's main characteristics is its capacity to offer Quality of Service (QoS) assurances, which guarantees a minimum bandwidth and a maximum delay to ensure that real-time data is sent with high consistency and dependability. RTP can also be integrated with other protocols, such as RTCP (RTP Control Protocol), to add further control and monitoring features. In general, RTP is crucial to numerous real-time applications that demand efficient and dependable audio and video data transfer across IP networks [19].

OSI is proposed by ISO (International Organization for Standardization). It explains how network elements should interact, broken down into seven levels, each in charge of certain operations. The seven-layer OSI, from lower to higher, includes Physical Layer, Data Link Layer, Network Layer, Transport Layer, Session Layer, Presentation Layer, and Application Layer [20]. The simplified OSI model often refers to the TCP/IP model, which includes Network Interface Layer, Internet Layer, Transport Layer, and Application Layer [20]. The Network Interface Layer combines the Physical and Data Link layers. The Internet Layer corresponds to the Network layer in the OSI model for routing and IP addressing. The Transport Layer remains the same in the OSI model for end-to-end connection and transmission control by transport protocols. The Application Layer combines the OSI model's Session, Presentation, and Application layers. It includes all processes that use the transport layer protocols to deliver data. The OSI model, TCP/IP model and the corresponding protocols are shown in Table 1.1 [21].

Table 1.1: OSI Model and TCP/IP Model

OSI Model	TCP/IP Model	Common Protocols for TCP/IP Model
Application	Application	HTTPS, HTTP, DNS
Presentation		
Session		
Transport	Transport	TCP, RTP, UDP, GTP
Network	Internet	IP
Data Link	Network	PPP
Physical	Interface	Ethernet

The GPRS Tunneling Protocol, or GTP, is a network protocol that facilitates the transport of IP data packets in various networks, including GPRS and the third, fourth, and even fifth-generation networks. It enables a flow of data and control information between the core network and mobile devices, such as smartphones. Due to the constant movement of user devices on mobile networks, this protocol was created specifically for them. However, a user's IP address must remain consistent to provide ongoing service. GTP allows user devices to keep their IP addresses while traveling, eliminating the need to change them every time the device travels [22].

1.1.3. Applications of Communication over IP

The analog telephone was used to provide one-way transmissions like fax. Compared with the analog telephone, the circuit-switched digital telephone offers bidirectional services with better qualifications. Nowadays, VoIP is a technology that enables voice communication over IP networks. VoIP transmits voice samples as digital packets over the Internet rather than conventional phone lines.

VoIP functions by digitizing analog voice samples to be transferred as digital packets across IP networks. At the receiving end, the packets are subsequently put back together to form voice samples. This kind of voice sample transmission, known as packet switching, enables voice samples to be conveyed more effectively and inexpensively than with conventional telephone networks [23]. VoIP's affordability is one of its key benefits. It can be significantly less expensive than conventional telephone networks because VoIP leverages the existing internet infrastructure for communication, eliminating the need for additional telephone lines or circuit-switched equipment, especially for long-distance or international conversations. VoIP can also be more flexible, providing services once the devices are connected to the Internet. Email, instant messaging, and multimedia are just a few examples of internet services and applications that VoIP can easily interact with. This integration and scalability increase communication flexibility, which can also provide further value-added services like voicemail, conference calling, and remote collaboration [24].

Numerous applications and services enable real-time communication over the Internet. Communication over IP applications includes the following:

1. Voice and video conferencing: To conduct virtual meetings, conferences, and webinars with participants in different places, many firms employ VoIP and video over IP technology.
2. Unified communications: Unified communications platforms, which combine diverse communication services, including messaging, audio and video calls, and file sharing into a single platform, frequently include VoIP and video over IP. For example, video call on Whatsapp or Wechat uses unified communications.
3. Remote learning and online classes: Numerous educational institutions provide remote learning and online courses to students who cannot attend in-person classes using VoIP and video over IP.
4. Telemedicine: Telemedicine applications use IP service to let medical professionals consult with patients via the Internet and keep track of their health.
5. Streaming media: To offer high-quality video material over the internet, streaming media services like Netflix, YouTube, and Amazon Prime Video use video over IP.

Figure 1.3 and Figure 1.4 show two examples of communication over IP applications, respectively.

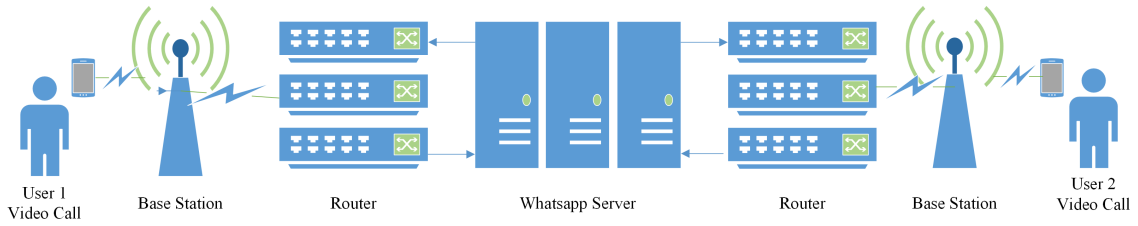


Figure 1.3: VoIP Application for Audio and Video Call Using Whatsapp

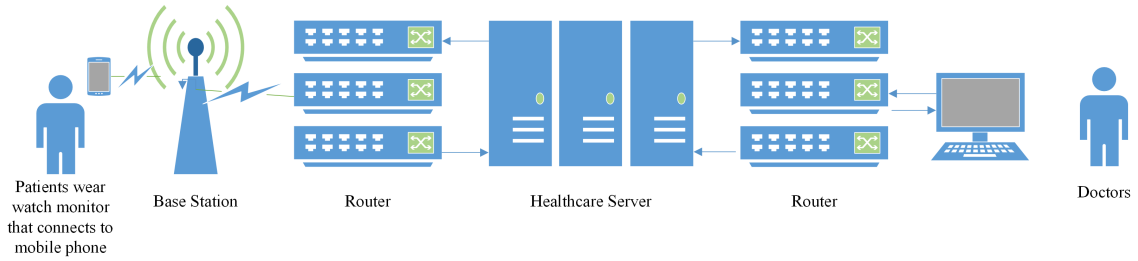


Figure 1.4: Communication over IP for Healthcare Smartwatch

In the example shown in Figure 1.3, user 1 calls user 2 in the Whatsapp smartphone application. The call is achieved by VoIP technology. It's feasible to transmit the audio signal only or both the audio and video signal simultaneously. As Appendix B describes, the data in each transmitted packet over the connections can vary from 20 to 1200 bytes.

In the second example illustrated in Figure 1.4, the patient wears a healthcare watch monitor connected to the mobile phone by Bluetooth Low Energy (BLE) protocol. The monitoring data is then transmitted to the healthcare company server by IP communication. A conclusion of the data will be obtained by analyzing the collected information. This company's doctor will watch the data analysis results to monitor the patient's health status. The data transmitted from the patient to the server will be small, varying from 10 to 20 bytes of each packet payload since only indicator parameters are transmitted instead of audio or video data.

PDU stands for Packet Data Unit. PDU session describes a logical link that allows user devices to send data packets to the 5G core network. The PDU session is created to address the communication needs of a particular service or application [25].

1.2. Essence of Research Study

For the protocol header calculation, the IPv4 header holds 20 bytes, and the combined header will be 40 bytes when combining RTP and UDP headers for radio network real-time communication. The size will be larger regarding IPv6 applications, as a 40-byte header is allocated to each data packet for the IPv6 protocol, and the huge header represents a large overhead. The burdensome headers will decrease the performance of the packet size-sensitive services, for instance, the

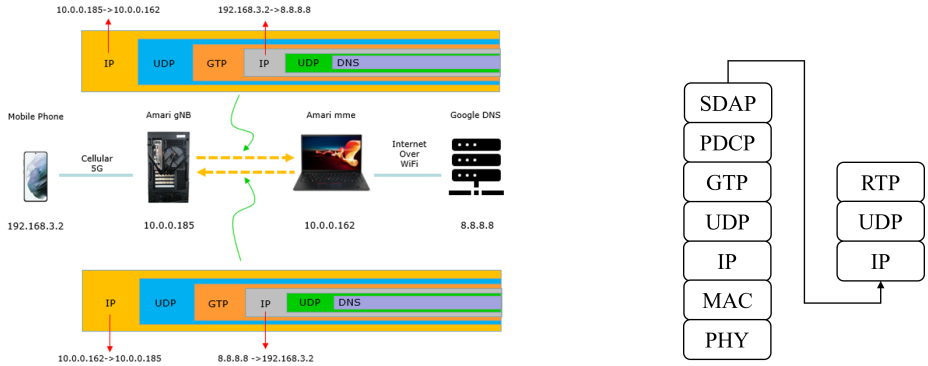
wireless service with a small payload.

Many modern packet-switched communication network applications require frequent sending of packets with small payloads while containing relatively substantial meta-data information in the packet header. For instance, several applications in the upcoming 5G cellular communication systems seek the transfer of small payloads, possibly in conjunction with low-latency packet service [10].

Figure 1.5a shows a typical application over multiple protocol layers. For mobile 5G communication, the packet is transmitted in a protocol stack like DNS/UDP/IPv4/GTP/UDP/IPv4 between the gNB and the User Plane Function (UPF). The header size of this protocol stack is 88 bytes. However, the payload of a typical G.729 coding VoIP packet is 20 bytes. The burdensome headers must be optimized to decrease bandwidth occupation and latency.

RoHC algorithm [26] seeks to provide an approach to compressing the protocol headers inscribed in data packets to decrease the amount of data transmission without reducing the quality of communication. The performance of the RoHC compression is shown in Figure 1.6a and Figure 1.6b.

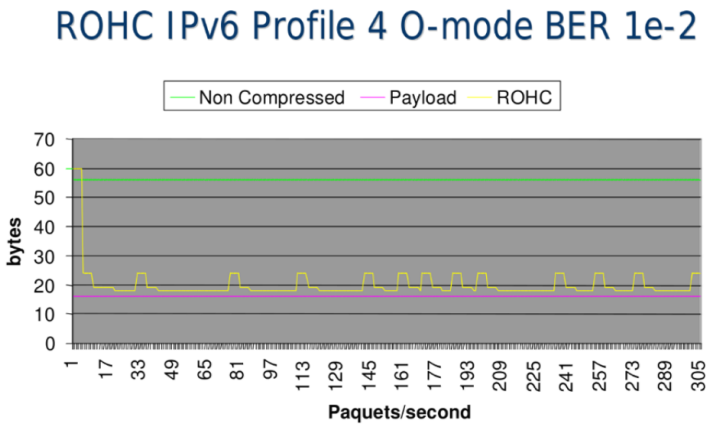
RoHC algorithm can compress the IPv6 (IPv4) header from 40 (20) to 1 byte. As shown in Figure 1.6a, the payload size is 16 bytes, the IPv6 header size is 40 bytes, the overall size is 56 bytes. The RoHC packet is larger than the uncompressed packet in the initial phase since some necessary information (related to the RoHC algorithm) is sent for initialization. Then the size of the RoHC packet decreases to 25 bytes and finally to 17 bytes. The fluctuation of the packet size because of the working mechanism of the RoHC algorithm is also viewed. A detailed explanation of the RoHC algorithm's working mechanism is given later.



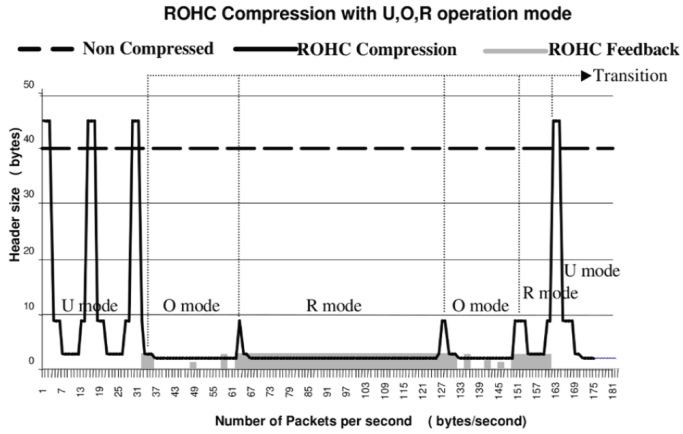
(a) Example of Protocol Stack for Real-life Application [27]

(b) Protocol Stack

Figure 1.5: Example of Protocol Stack



(a) Performance of ROHC compression for IPv6



(b) Compression Performance in Mode Transition

Figure 1.6: Performance Evaluation of RoHC [28]

As shown in Figure 1.6b, there are three different working modes in the RoHC algorithm, including U-mode, O-mode and R-mode. The algorithm uses the unidirectional transmission in U-mode for transmission mechanism simplicity, which causes bad fluctuations in packet size. The algorithm uses bidirectional transmission in the O-mode for receiving feedback from the receiver. In this case, the fluctuations in packet size become lower since the feedback will help to calibrate the operational phase. The algorithm uses bidirectional transmission with more feedback in the R-mode, eliminating packet fluctuations when transmitting because of enough feedback assurance.

However, the existing RoHC algorithm is not enough for the requirements of present header compression since RoHC header compression does not include newly applied headers like Packet Data Convergence Protocol (PDCP, shown in Figure 1.5b) and GPRS Tunnelling Protocol for the User Plane (GTP-U) in the 5G communications [29]. The packet headers are proposed to be optimized over the full stack for the real-time packet stream and the GTP tunnels in the 5G networks. For example, header compression for protocols like PDCP, Service Data Adaptation Protocol (SDAP), and GTP-U should be considered.

1.3. Research Motivation and Challenges

The research motivation and challenges are shown below:

1. In realizing 5G communication, the packet will transfer over the protocol stack of 5G, including RTP/UDP/IP/SDAP/PDCP/GTP/UDP/IP, as shown in Figure 1.5b. The header compression will be applied over the protocol stack (layer 3 and higher) for RTP transport through GTP tunnels. The data packet structure and the interaction mechanism are not identical in these protocols. Optimized compression should be achieved by coordinating the overall structures of protocols.
2. Two RTP streams will be initiated and applied in applying 5G communication for video calls. These RTP streams terminate in the same end-points, namely the UE of one session participant and the UE of the other (e.g., calling party and called party). The optimized compression will be applied for multi-stream media transport sessions since some of the relevant information in the RTP stream can be applied only in one of the RTP streams. The challenge of this research is locating identical information in multiple RTP streams in the same PDU session.
3. Header compression can also be done vertically for multi-layer protocol transmission, which passes through the inner and outer layer protocols. The challenge of this optimization is finding redundant information in vertical compression manners.

1.4. Research Objectives

In this section, the objectives of this study are presented based on requirements derived for 5G packet compression and obstacles that arise:

1. To develop an optimized header compression technique for real-time transport through GTP in 5G networks.
2. To develop an optimized header compression technique for multi-stream media transport sessions.
3. To develop an optimized header compression technique for multi-layer communication sessions.

1.5. Methodology

The following states the methodology employed in this study to achieve the objectives indicated in Section 1.4.

1. Literature study about GTP signaling of 5G communication and RoHC research.

2. Drawing up architecture and protocol diagrams for real-time communication in the 5G network according to the 3GPP standard.
3. Devising optimized header compression technique based on existing RoHC library.
4. Verification of the optimized header compression technique through simulations.
5. Deriving conclusions and recommendations.

1.6. Thesis Outline

This thesis is structured as follows: Chapter II presents the literature review, providing an extensive overview of existing work in 5G communication and RoHC. Chapter III describes data transmission in the 5G communication network, presenting the protocol stacks used in the GTP-U tunnels between the end-user (mobile device) and the application in the data network. Chapter IV introduces the overall structure of the RoHC algorithm and its mechanism for optimizing the packet header. Chapter V gives a comprehensive design of the optimized header compression algorithm proposed in the thesis. Chapter VI analyzes the data obtained in the experiment and discusses the results. The conclusion and some recommendations for further research are drawn in Chapter VII. Some additional information and introductions are in Appendix A, B, and C.

2

Literature Review

The 3rd Generation Partnership Project (3GPP) Release 15 defines the system known as the Fifth Generation of Mobile Telephony (5G) or 5G System (5GS). It was functionally locked in June 2018 and was described entirely by September 2019. The 3GPP standard also specifies the protocols and network interfaces that allow the whole mobile system, including call and session control, mobility management, service delivery, and more. This technology enables 3GPP networks to operate with many vendors and operators [12, 30].

To enhance user experience and expand functionality, 5G incorporates various dedicated technologies. These include ”Network Functions Virtualization” and ”Slicing” to enable increased modularity, ”EDGE Computing” to achieve faster response time and lower bandwidth usage, and ”Non-Terrestrial Network (NTN) Satellite Communications” for widespread coverage. These technologies aim to improve the overall performance and capabilities of the 5G network [12]. The 3GPP proposed a Non-Stand-Alone architecture (NSA) in Release-15 to facilitate a seamless transition from 4G to 5G. In this architecture, the existing Evolved Packet Core (EPC) supports both 5G New Radio (5G-NR) and Long-Term Evolution (LTE) radio technologies, along with their respective base stations. This allows for the coexistence of 4G and 5G systems, enabling a smooth migration to the 5G Stand-Alone (SA) architecture [12].

3GPP TS 38.300 Release 17 provides a comprehensive overview and description of the Next Generation Radio Access Network (NG-RAN). It specifically focuses on the radio interface protocol architecture of the New Radio connected to the 5G Core (5GC). The previous releases cover the Evolved Universal Mobile Telecommunications System (UMTS) Terrestrial Radio Access (E-UTRA) architecture connected to the 5GC. This document outlines the structure and components of the NG-RAN and its interface with the 5GC [31]. Similarly, 3GPP TS 38.401 Release 17 provides a detailed explanation of the overall architecture of the NG-RAN. It describes the various in-

terfaces within the NG-RAN, including the NG, Xn and F1 interfaces. The document gives insights into how these interfaces enable communication and coordination between network elements. It also outlines the layers and protocol stack of the 5G Access Network [32].

3GPP TS 29.281 Release 17 [33] defines GTP-U, which enables tunneling of user plane data over N3 (i.e., between the 5G-AN node and the UPF) and N9 (i.e., between multiple UPFs of the 5GC) in the backbone network. The user plane carries the network user traffic, while the control plane carries signaling traffic in 5GC. GTP encapsulates all PDUs for end-users. It offers encapsulation on a PDU session by PDU session basis. This layer also holds the QoS flow's associated marking. 3GPP TS 38.474 Release 17 [34] specifies the standards for user data transport protocols and related signaling protocols to establish user plane transport bearers over the F1 interface [33, 34]. The visualization of 5G network architecture is shown in Figure 3.1 and Figure 3.2.

RoHC version 1 in IETF RFC 3095 [26] was built on the idea of extensibility using several profiles, such as the IP, UDP-Lite, and TCP profiles, in 2001. With strong industry backing, RoHCv1 was adopted into the 3GPP, the UMTS, and the Worldwide Interoperability for Microwave Access (WiMAX) networks. RoHCv1 was then updated as RoHCv2 in IETF RFC 5225 [35] in 2008. The key new features of RoHCv2 are its simplicity and robustness under losses, as well as improved speed and compression [26, 30, 35].

Effnet company [36] offers a highly portable, efficient, and completely compatible software implementation of RoHCv2 technology, which is based on the 2008 and 2010 IETF specifications RFC 5225 [35] and RFC 5795 [37]. The research and development team of Effnet has actively participated in the procedure. The RoHCv2 is applied by the Effnet project. RoHCv2 specifies robustness methods that can be utilized by a compressor implementation to increase the likelihood of successful decompression when packets can be lost and/or reordered [36].

For exploring the performance of the RoHC algorithm, Minaburo [28] discusses RoHC interaction in the 3GPP architecture, proposing the data structure optimization in IPv6 RoHC. Naidu and Tapadiya [38] introduce the implementation of header compression in 3GPP LTE, increasing effective link bandwidth and decreasing packet processing requirements in power-sensitive downstream devices. Chen et al. [39] presented a brief 5-page conference paper survey of RoHC compression schemes in 2012 [10].

For the extension of RoHC, the 3GPP RAN2 working group has also indicated an interest in defining a RoHC-like compression profile for Ethernet in conjunction with IP, UDP and TCP. Chen et al. [40] have predicted that compression will improve by 20–26% for Ethernet frames that are 64 bytes long. Recently, it has been demonstrated that tailored RoHCv2 compression reduces Opportunistic Routing (OpR) message size in mesh networks by at least 50%, with the possibility of even greater compression under specific circumstances [41]. A scalable end-to-end header compression approach by Sun et al. [42] is based on a unique routing model that uses the locator/identifier separation idea through RoHC. Simulations show that the delay could be cut by 16% compared to the uncompressed case with many hops. Kwon et al. [43] proposed the Advanced

Television Systems Committee (ATSC) Link-Layer Protocol (ALP) based on customized RoHC as a unified adaptation layer between the physical and network layers for different input formats. Consequently, there is still enough room for improvement in RoHC [44, 10].

With the development of 5G technology and future 6G design, the conventional RoHC is apparently insufficient for compressing more redundant header sizes to decrease the packet size and improve the QoS. Even if the researchers mentioned above provide different customized compression solutions for decreasing redundant headers, none of them offers a comprehensive solution for compressing the RTP stream over the present 5G protocol stack. Hence, this thesis study proposes an optimized header compression algorithm for the RTP stream to shorten the protocol headers over the 5G protocol stack.

3

Data Transmission in 5G Communication Network

This chapter provides a detailed explanation of the study's foundational elements. Section 3.1 explains the 5G overall architecture, 5G system architecture, and the separation of NG-RAN and 5GC. Section 3.2 explains GTP-U, including the GTP-U tunnel, protocol stack, and GTP-U header structure. The PDU session and QoS flow are briefly mentioned in Section 3.3. Section 3.4 introduces the protocol stacks of 5G communications.

3.1. Architecture of 5G Communication

The architecture of 5G communication is the foundation for applications and service provisioning. For seamless transfer from 4G to 5G, the present NG-RAN architecture is illustrated in Figure 3.1. An NG-RAN node is either a gNB, providing NR user plane and control plane protocol terminations towards the UE, or a ng-eNB, providing E-UTRA user plane and control plane protocol terminations towards the UE [31, 45].

Through the Xn interface, the gNBs and ng-eNBs are interconnected. The gNBs and ng-eNBs are also connected to the 5GC via the NG interface, specifically to the Access and Mobility Management Function (AMF) via the NG Control plane (NG-C) and to the UPF via the NG User plane (NG-U) [46].

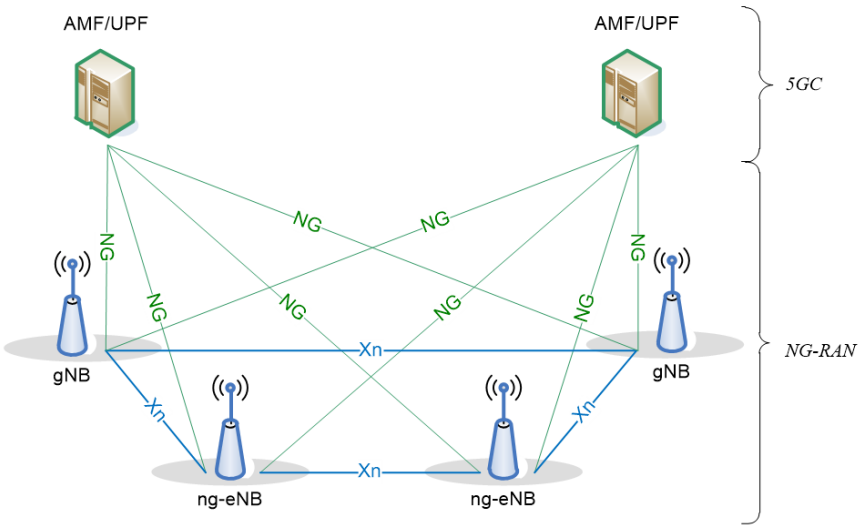


Figure 3.1: NG-RAN Architecture [31]

3.1.1. Overall Architecture of 5G Communication

Schematically, the 5GS employs the same components as previous generations: UE consisting of a Mobile Station and a USIM, a NG-RAN, and a 5GC [12]. 5GS consists of 5GC and NG-RAN.

The overall architecture of 5G communication is shown in Figure 3.2. The UE communicates with NG-RAN through the air interface (NR-Uu). The primary component of the NG-RAN is the gNB, where ”g” stands for ”5G” and ”NB” refers to ”Node B,” the name inherited from 3G onwards for the base station. The radio interface is named ”NR-Uu” for similar but distinct reasons. The gNB can be divided further into a gNB-Central Unit (gNB-CU) and one or more gNB-Distributed Unit(s) (gNB-DU), which are connected via the F1-U/ F1-C interface [12, 31].

The gNB-CU is a logical node that incorporates gNB services such as transferring user data, mobility control, radio access network sharing, positioning, session management, except for those capabilities specific to the gNB-DU. Over the front haul, the gNB-CU manages the operation of the gNB-DUs. A central unit (CU) may be called a Baseband Unit (BBU), REC, or RCC. The gNB-CU can be further split into gNB-CU-CP and gNB-CU-UP for flexible arrangements in diverse applications. Depending on the functional split choice, the gNB-DU comprises a subset of the gNB’s protocol layers. The entire user plane could be in the distributed unit in some cases. PDCP could also be in the centralized unit in some cases, depending on the requirements. Its operation is under the responsibility of the CU. Distributed Unit (DU) is sometimes called Remote Radio Unit (RRU)/RE/RU [31].

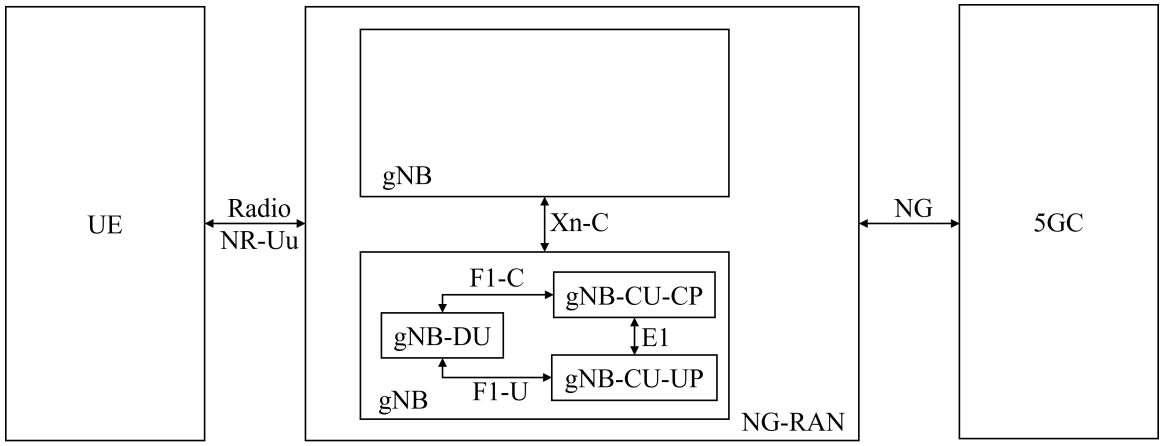


Figure 3.2: Overall Architecture of 5G Communication

The link between gNodeB-DU and gNodeB-CU-UP is established and managed by the gNodeB-CU-CP. If linked to the same gNodeB-CU-CP, a gNodeB-DU can connect to numerous gNodeB-CU-UP. Additionally, if linked to the same gNodeB-CU-CP, a gNodeB-CU-UP can support multiple gNodeB-DU. In 5G networks, the division of gNB-CU-CP and gNB-CU-UP enables improved resource optimization, scalability, and flexibility, enhancing network performance and effectiveness [32, 47, 48].

3.1.2. Architecture of 5GC

The 5GC architecture is based on a ”service-based architecture” (SBA) framework in which the architecture elements are defined in terms of ”network functions” (NFs) as opposed to ”conventional” network entities. Any given NF provides its services to all other approved NFs and/or ”consumers” who are authorized to access these services via the reference point of a common framework. This SBA strategy provides modularity and reusability. The 5G system architecture and the main NFs are shown in Figure 3.3 [12, 46].

The following NFs are shown [12, 46]:

- The Application Function (AF) governs the functions like network slice selection, network exposure and data exposure.
- The Session Management Function (SMF) manages and controls UE sessions and connections. It is responsible for setting up, managing, and terminating user sessions in the network. It also performs user authentication, policy control, and charging functions.
- The Unified Data Management (UDM) involves the consolidation, organization, and control of data across different systems, applications, and databases within an organization.
- The Policy Control Function (PCF) is responsible for enforcing policy rules and controlling the behavior of network resources and services.

- The Network Repository Function (NRF) controls the other NFs by providing NF register, deregister and update services.
- The security-related NFs include Network Exposure Function (NEF), Authentication Server Function (AUSF), and Security Anchor Functionality (SEAF).
- The Network Slice Selection Function (NSSF) manages and orchestrates network slices to meet the specific requirements and characteristics of different services and applications.
- The Access and Mobility Management Function (AMF) manages access and mobility-related functions for UE in a 5G network.
- The User Plane Function (UPF) is routed between the UE and external networks through this function. It performs tasks like packet forwarding, QoS enforcing, and billing. The UPF directly contributes to the high-speed, low-latency features that characterize 5G technology by regulating and controlling data flow in a 5G network.

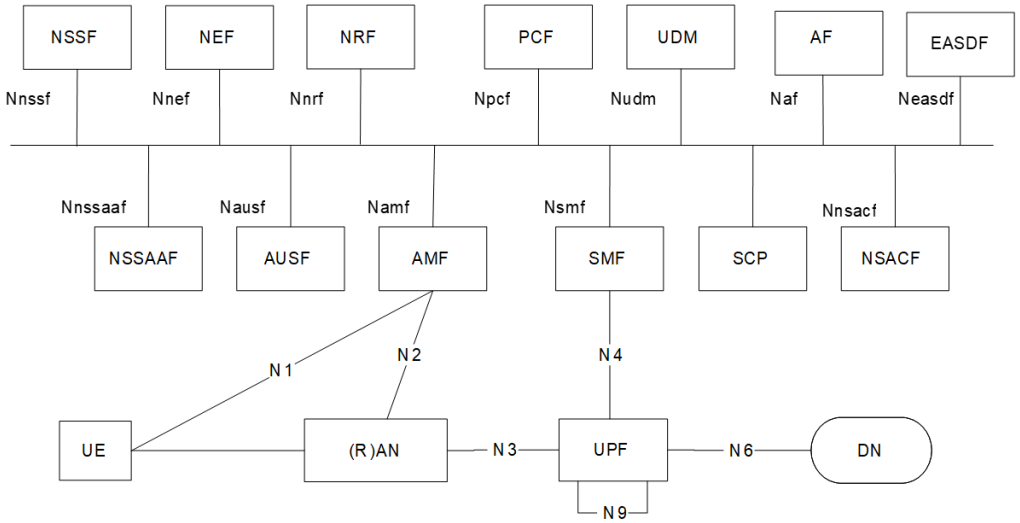


Figure 3.3: The 5G System Architecture [46]

The other NFs are explained in 3GPP TS 23.501 [46]. The main differences lie in SBA's flexibility, scalability, and dynamism compared to conventional reference point-based architecture's more rigid and predefined nature.

3.1.3. Components of 5GS

The 5G System divides network functions among distinct network nodes or parts. The goal of the 5GS functional divide is to optimize resource usage, enable effective network management, and deliver high-quality services to end customers. The UE, NG-RAN and the 5GC comprise the major components of the 5GS functional split. The latter two parts are shown in Figure 3.4 [31]:

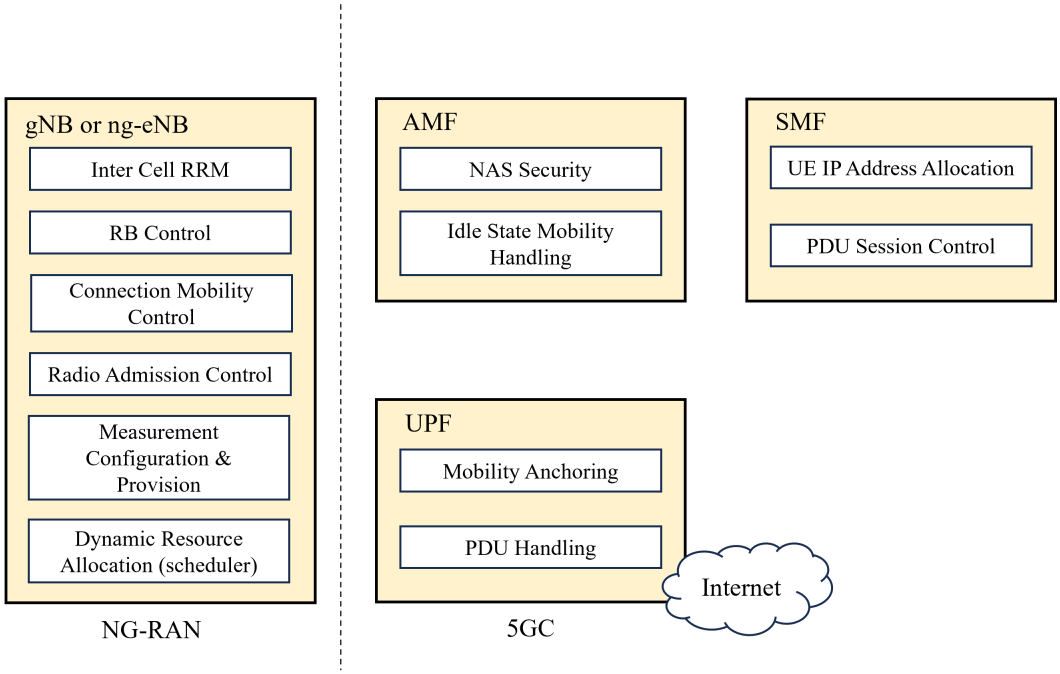


Figure 3.4: Functional Split between NG-RAN and 5GC [31]

1. Radio Access Network: The UE can connect to the 5G network thanks to the RAN's wireless access to the network. There are several sections of the RAN [31]:
   - Inter-cell Radio Resource Management (RRM): It describes the methods and formulas employed to control the radio resources, such as bandwidth and power in a cellular network.
   - RB Control: Logical channels between the network and the UE have been established. Radio bearers transport user data and control information between the UE and the network. Based on the network conditions and the QoS needs, the RB Control function controls the distribution, configuration, and release of these RBs. The relationship of PDU session, radio bearer and QoS flow is shown in Figure C.3 in Appendix C.
   - Connection Mobility Control: Commonly known as handovers, it manages and controls UEs' mobility as they move across different network areas or access technologies within the 5G network.
   - Radio Admission Control (RAC): It is about admitting or rejecting bearer establishments.
   - Measurement Configuration and Provisioning: It describes configuring and giving the UE the necessary parameters to measure the radio channel quality in cellular networks.
   - Dynamic Resource Allocation (DRA) or Dynamic Scheduling: It is a process used in cellular networks to optimize the distribution of radio resources to UE depending on their QoS needs and network conditions, such as frequency, time slots, and transmission power.
2. 5GC: The 5GC manages network operations and services, including routing, mobility, and session management. The 5GC is mainly divided into SMF, AMF and UPF [31].

The 5GS allows an adaptable and scalable architecture to serve various use cases and services and high-performance and effective network management.

3.2. GTP User Plane

The initial development of the GPRS tunneling protocol was for packet tunneling within GPRS. Later, improvements and changes were made to meet the needs of the 3G, 4G and 5G technology. GTP has three versions: GPRS Tunneling Protocol for the control plane (GTP-C), GTP-U, and GTP Prime (GTP') to obtain tunneling charge information from PCF. GTP-U tunnels are directional tunnels used to tunnel Transport PDU (T-PDU) between sending and receiving GTP entities. Two unidirectional GTP-U tunnels are utilized to create a bidirectional GTP-U tunnel between two entities. The receiving GTP entity assigns a Tunnel Endpoint Identifier (TEID) to each GTP-U tunnel. The TEID aids the GTP-U entity in identifying the tunnel corresponding to a specific UE through which T-PDUs must be delivered. Control plane protocols share the TEID between entities that send and receive data. These protocols include the NG Application Protocol (NGAP), E1 Application Protocol (E1AP), F1 Application Protocol (F1AP), and Packet Forwarding Control Protocol (PFCP) [33, 48].

By enabling effective user data packet transfers between the gNB and the core network, the GTP User Plane plays a critical role in 5G networks. It ensures that user data is appropriately encapsulated, transported, and delivered, which helps 5G users have a dependable and high-performance communication experience [33, 48].

3.2.1. GTP-U Tunnels

Between a particular set of GTP-U Tunnel Endpoints, GTP-U Tunnels transport signalling messages and T-PDUs that have been encapsulated. The TEID in the GTP header determines which tunnel a specific T-PDU belongs to. This is how GTP-U multiplexes and demultiplexes packets between a particular pair of Tunnel Endpoints [33, 45].

There will be two IP layers in the protocol stack with the GTP-U protocol. The outer IP layer is lower than the GTP-U layer, and the inner IP layer is higher than the GTP-U layer. The outer GTPv1-U IP packet will be called the IP packet with a GTPv1-U packet. The inner IP packet of a GTPv1-U packet (T-PDU) is either [33]:

- A downlink IP packet transmitted to the UE/MS from the external network indicated by the Access Point Name (APN) across one or more layers.
- A Uplink IP packet transmitted from a UE/MS across one or more tunnels to the external network indicated by the APN.

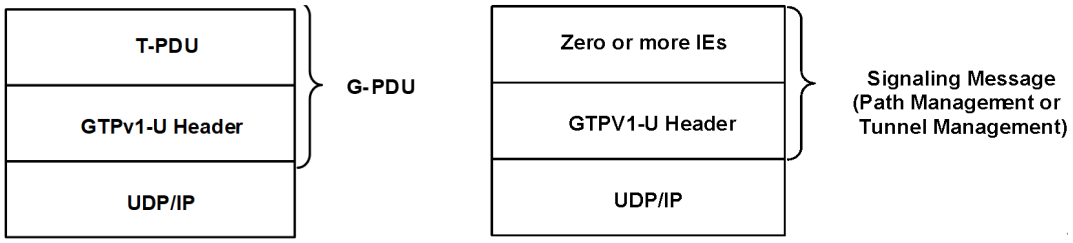
At the GTPv1-U sender, an inner IP packet must be encapsulated with a GTP header, a UDP

header, and an IP header. The sender must fragment the IP packet if the resulting outer IP packet is larger than the first connection's Maximum Transmission Unit (MTU) leading to the destination GTPv1-U endpoint [33, 45].

The GTPv1-U receiving endpoint is responsible for reassembling any IP fragments in datagrams it receives from the GTPv1-U sending endpoint. The outer IP header, outer UDP header, and outer GTP header, including any GTP extension headers, must all be the size of the receiving endpoint's IP reassembly buffer, which must be at least the inner MTU size. The inner IP packet is then extracted from the fully reassembled IP packet and delivered to the IP/UDP/GTPv1-U layers for further processing dependent on the capabilities of the receiving node [33, 45].

3.2.2. GTP-U Protocol Stack

Figure 3.5a shows the protocol stack for G-PDU. Figure 3.5b shows the protocol stack for a GTP-PDU signaling message.



(a) G-PDU Protocol Stack **(b)** Signalling Message Protocol Stack

Figure 3.5: GTP-U Protocol Stack [33]

3.2.3. GTP-U Header

The GTP-U header has a configurable length with a minimum of 8 bytes. The flags used to denote the presence of extra optional fields are the PN, S, and E flags. There are N-PDU numerals present, according to the PN flag. The S flag indicates the presence of the GTP Sequence Number (SN) field. The E flag indicates the Extension Header field, intended to allow future expansions of the GTP header described in this document without utilizing a distinct version number [45].

If one or more of these three flags is set, Sequence Number, N-PDU, and Extension Header must all be present. The sender must set all bits in fields that aren't needed to zero. Analysis of new areas is optional by the receiver. For instance, the N-PDU Number and Sequence Number fields must be present but not assessed since they will not have valid values if the E flag is set to 1 [45].

The outline of the GTP-U header is indicated in Table 3.1 [33].

- Version: This field determines the GTP-U protocol version. Set the version number to 1.

- Protocol Type (PT): This bit distinguishes between GTP and GTP prime when PT is ’1’ or ’0’. The GTP protocol is discussed in this document, and 3GPP TS 32.295.
- Sequence number flag (S): This symbol indicates that a meaningful value may be found in the Sequence Number field.
- N-PDU Number flag (PN): This flag indicates that the N-PDU Number field contains a meaningful value.
- Message Type: This field indicates the type of GTP-U message.
- Length: This field specifies the length in octets of the payload, which is the remainder of the packet after the necessary GTP header (the first 8 octets).
- TEID: This element uniquely identifies a tunnel endpoint within the GTP-U protocol entity receiving the packet. Locally, the receiving endpoint of a GTP tunnel assigns the TEID value that the transmitting endpoint must utilize.
- Sequence Number: When transmission order must be preserved, a rising sequence number for T-PDUs is broadcast over GTP-U tunnels if the Sequence Number field is used for G-PDUs (T-PDUs + headers).
- N-PDU Number: This field is utilized during the Inter SGSN Routing Area Update method and certain intersystem handover procedures (e.g., between 2G and 3G radio access networks).
- Next Extension Header Type: This field defines the type of Extension Header that follows this field in the GTP-PDU.

Table 3.1: Outline of the GTP-U Header [33]

	Bits							
Octet	8	7	6	5	4	3	2	1
1	Version			PT	(*)	E	S	PN
2	Messge Type							
3	Length (1st Octet)							
4	Length (2nd Octet)							
5	Tunnel Endpoint Identifier (1st Octet)							
6	Tunnel Endpoint Identifier (2nd Octet)							
7	Tunnel Endpoint Identifier (3rd Octet)							
8	Tunnel Endpoint Identifier (4th Octet)							
9	Sequence Number (1st Octet)							
10	Sequence Number (2nd Octet)							
11	N-PDU Number							
12	Next Extension Header Type							

3.3. PDU Session

PDU session provides a UE with a logical user-plane connection to the Data Network (DN), which transfers PDUs between the UE and the DN. The UE offers a unique PDU session ID that can be used to identify one of the UE's PDU sessions. The major purpose of the PDU session formation operation is to configure a default QoS flow between UE and UPF via gNodeB [48]. The visualization of the PDU session is shown in Figure C.3. More detail about the PDU session is shown in Appendix C.

3.4. 5G Protocol Stack

This section defines the general protocol stacks within 5GS entities, such as between the UE and 5GC Network Functions, between the 5G-AN and 5GC Network Functions, or between the UE and 5G-AN Network Functions.

3.4.1. Control Plane Protocol Stack

This section illustrates the protocol stack of the control plane within the 5G system.

3.4.1.1 Radio Protocol Architecture

Figure 3.6 illustrates the radio protocol stack for the control plane as follows (AMF only shows Non-Access Stratum Control Protocol (NAS) for simplicity):

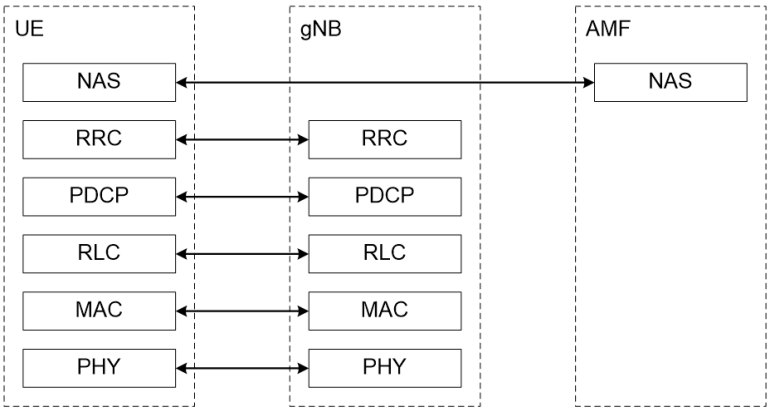


Figure 3.6: Control Plane Protocol Stack of NG-RAN [31]

Legend:

- NAS: It performs authentication, mobility management, and security control functions.
- Radio Resource Control Protocol (RRC): The RRC protocol is responsible for various ser-

vices and functions over the Uu interface. These services and functions include mobility management, QoS management, UE measurement reporting and control, radio link failure detection and recovery, and transfer of Non-Access Stratum messages to and from the UE.

- PDCP: The PDCP sublayer offers several essential services and functions. These include the transfer of data, whether it is in the user plane or the control plane. The PDCP sublayer maintains PDCP Sequence Number to ensure data packets' integrity and order correctly. The PDCP sublayer also performs header compression and decompression using the RoHC protocol. This compression technique helps reduce the overhead of transmitting packet headers, thus optimizing bandwidth utilization and improving overall network efficiency.
- Radio Link Control (RLC): The RLC sublayer plays a crucial role in providing services such as upper layer PDU transfer, independent sequence numbering, error correction through ARQ, segmentation and re-segmentation, reassembly of SDUs, duplicate detection, SDU discard, RLC re-establishment, and protocol error detection. These functions contribute to efficient and reliable data transmission within the 5G network.
- Medium Access Control (MAC): The MAC sublayer performs vital services such as mapping logical channels to transport channels and handling the multiplexing and demultiplexing of MAC SDUs. These functions ensure the proper organization and transmission of data within the 5G network architecture.
- NR-Physical (PHY): The PHY sublayer offers services such as beamforming, multiplexing, multiple antenna mapping, transport channel to physical channel mapping, as well as handling channel coding and Random Access procedures. These functions contribute to efficient and reliable data transmission within the 5G network architecture.

3.4.1.2 5G-AN - AMF

The protocol stack between 5G-AN (gNodeB) and AMF (5GC) is illustrated in Figure 3.7.

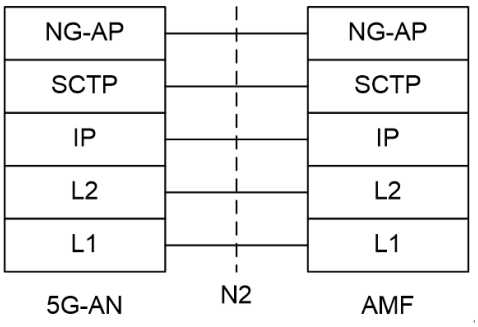


Figure 3.7: Control Plane Protocol between the 5G-AN and the AMF [46]

Legend:

- N2: Reference point between the 5G-AN and the AMF.

- NG Application Protocol (NG-AP): Application Layer Protocol between the 5G-AN node and the AMF. NG-AP is defined in 3GPP TS 38.413 [49].
- Stream Control Transmission Protocol (SCTP): This protocol ensures the transport of signaling messages between the AMF and the 5G-AN node. IETF RFC 4960 [50] specifies SCTP [46].

3.4.1.3 5G-AN - SMF

The protocol stack between 5G-AN and SMF (5GC) is shown in Figure 3.8.

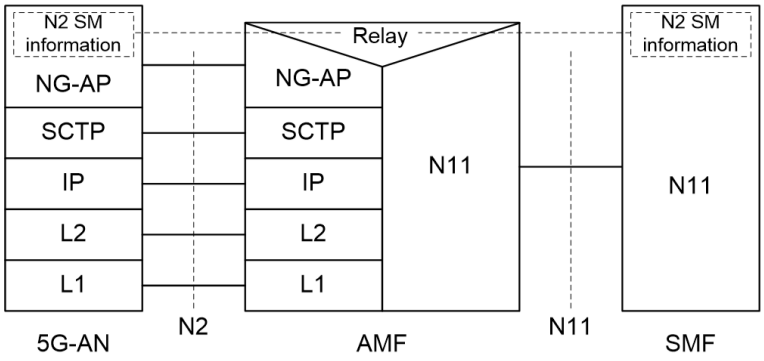


Figure 3.8: Control Plane Protocol between the 5G-AN and the SMF [46]

Legend:

- N2 SM information: This is the subset of NG-AP information that the AMF transparently relays between the 5G-AN and the SMF and is included in the NG-AP and N11-related messages [46].

3.4.1.4 UE - AMF

The control plane protocol stack between UE and AMF (5GC) is shown in Figure 3.9.

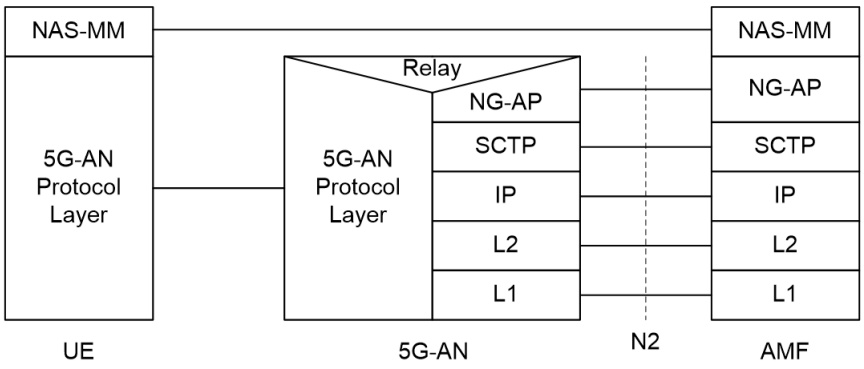


Figure 3.9: Control Plane between the UE and the AMF [46]

Legend:

- NAS-MM: The NAS protocol for MM functionality enables registration administration, connection management, and activation and deactivation of user plane connections. It is also responsible for NAS signal ciphering and integrity protection. 3GPP TS 24.501 [51] defines the 5G NAS protocol [46].
- 5G-AN Protocol layer: This collection of protocols and layers depends on 5G-AN. In the case of NG-RAN, the E-UTRA & E-UTRAN specifications define the radio protocol between the UE and the NG-RAN node (eNodeB or gNodeB) [12, 46].

3.4.1.5 UE - SMF

The control plane protocol stack between UE and SMF (5GC) is shown in Figure 3.10.

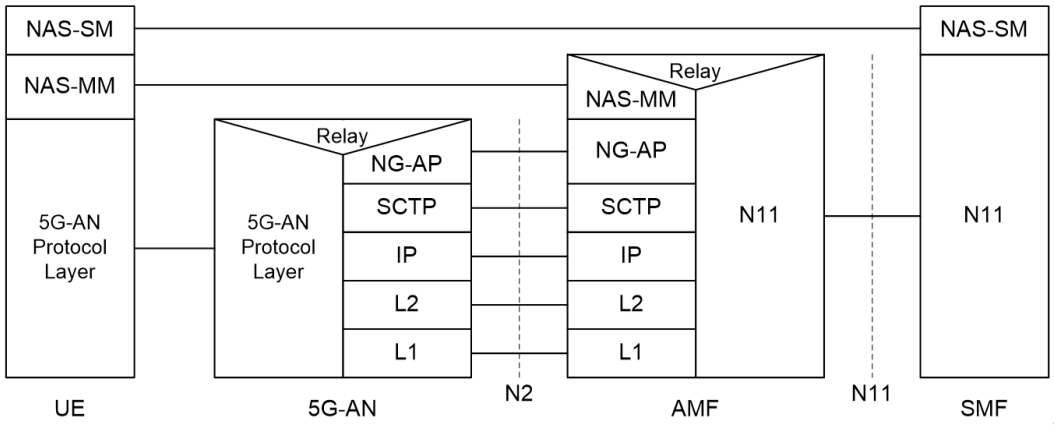


Figure 3.10: Control Plane between the UE and the SMF [46]

Legend:

- N11: Reference point between the AMF and the SMF.
- NAS-SM: The SM feature of the NAS protocol allows user plane PDU Session Establishment, modification, and release. It is transported through the AMF and is invisible to the AMF.

3.4.2. User Plane Protocol Stack

This section illustrates the protocol stack for the user plane transport related to a PDU Session, as illustrated in Figure 3.11.

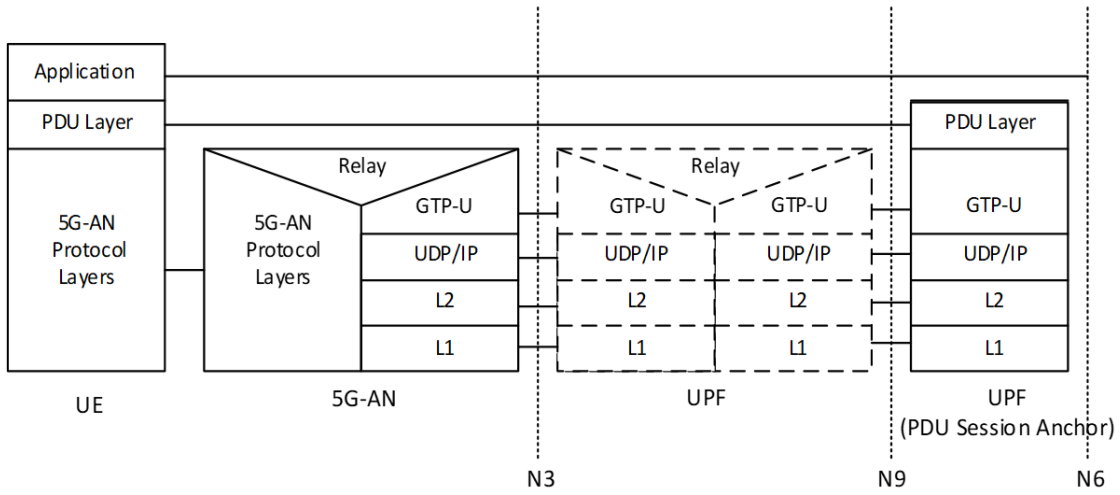


Figure 3.11: User Plane Protocol Stack [46]

Legend:

- PDU layer: This layer corresponds to the PDU carried by the PDU session between the UE and the DN. When the PDU session type is IPv4, IPv6, or IPv4v6, it corresponds to IPv4 packets, IPv6 packets, or both.
- GTP-U: This protocol supports tunneling user data through N3 (5G-AN node to UPF) and N9 (5GC UPF to UPF) in the backbone network [33]. GTP encapsulates all end-user PDUs. It provides encapsulation on a per PDU session level. This layer also carries the marking associated with a QoS Flow.
- 5G-AN protocol stack: This set of protocols/layers depends on the AN. L2 is also called the ”Data Link Layer”, and L1 is the ”Physical Layer”.

3.4.2.1 Radio Protocol Architecture

Figure 3.12 shows the protocol stack of NG-RAN for the user plane.

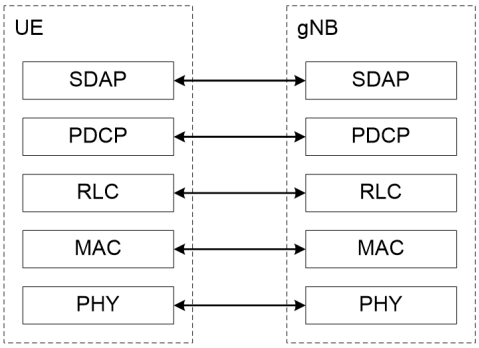


Figure 3.12: User Plane Protocol Stack of NG-RAN [31]

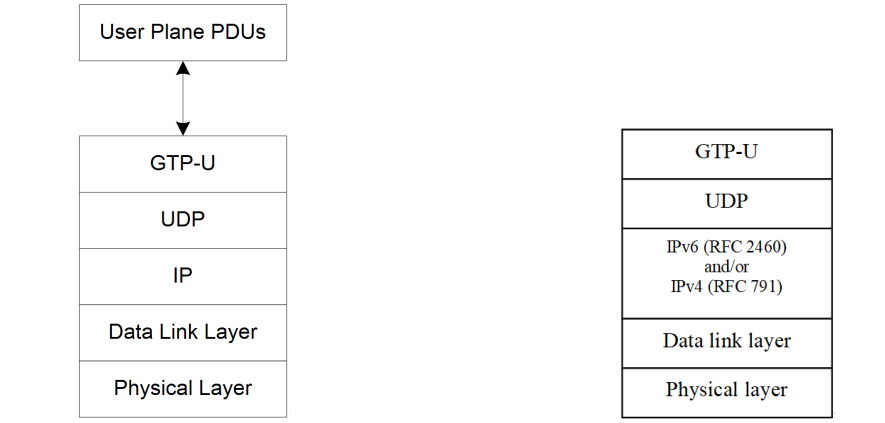
Legend:

- SDAP: The SDAP plays a crucial role in mapping QoS flows to data radio bearers and marking the QoS Flow Identifier (QFI) in DL and UL packets. These functions enable effective QoS management and ensure that the appropriate QoS parameters are applied to the corresponding data transmission within the 5G network architecture [31].

A single protocol entity of SDAP is configured for each individual PDU session.

3.4.2.2 NG and F1 Reference Point

The NG user plane Reference Point (NG-U) is defined between the NG-RAN node and the UPF. as shown in Figure 3.13a. The transport network layer over F1-U is shown in Figure 3.13b [31, 33, 45].



(a) User Plane Protocol Stack of NG-U **(b)** Transport network layer over F1

Figure 3.13: Reference Point of NG and F1 [31]

3.4.2.3 Overall User Plane Protocol Stack

Based on the description above, the overall protocol stack for the user plane between UE and application is illustrated in Figure 3.14. The communication between UE and gNB-DU is based on RLC/MAC/PHY layers. F1-U interface is applied for connecting gNB-DU and gNB-CU-UP via the GTP tunnel. SDAP and PDCP layers help to interact between UE and gNB-CU-UP. The N3 interface is used to bridge the communication between gNB-CU-UP and UPF. Application layer data is finally offered from UPF via the N6 interface crossing RTP/UDP/IP. A detailed description of GTP-U in this scenario is illustrated in section 3.3.

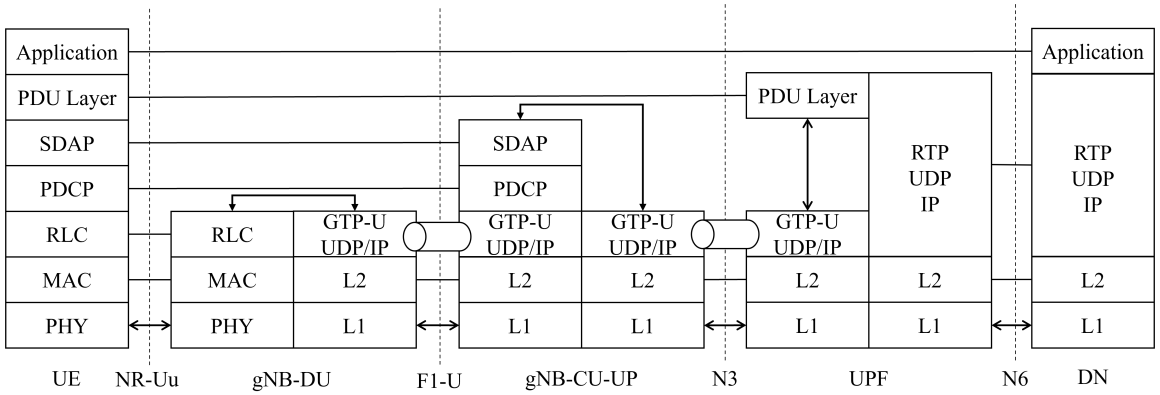


Figure 3.14: Overall Protocol Stack for User Plane

The data flow in the reference points is shown in Figure 3.15. The GTP tunneling headers in F1-U and in N3 are different.

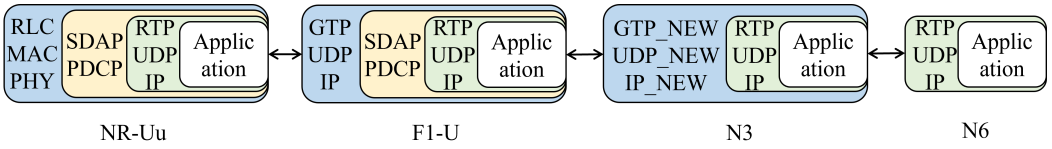


Figure 3.15: Data Flow in the Reference Points

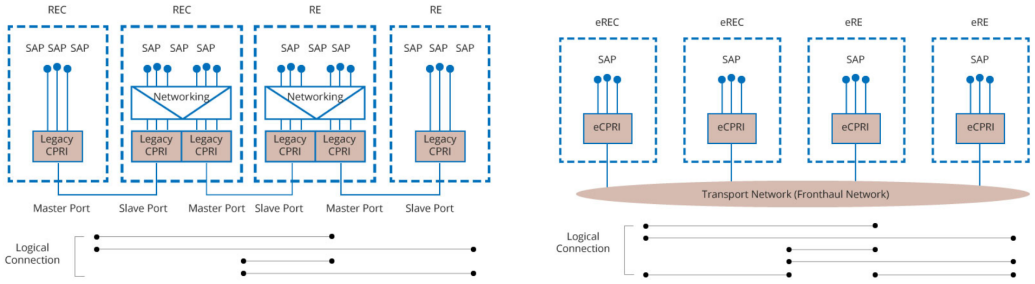
Besides, it's important to consider Layers 2 and 1 in the overall protocol stack shown in Figure 3.14. In order to facilitate high-speed data transmission and signal processing, the base station's BBU and RRU are connected by the fiber-optic interface protocol known as Common Public Radio Interface (CPRI) in 5G. The function of BBU is largely replaced by gNB-CU and gNB-DU in 5G. The first version of the protocol, which was cooperatively developed by several manufacturers of wireless communication equipment, was launched in 2003. In order to enable high-speed data and control information transmission over optical fiber at speeds surpassing 10 Gbps, with low latency and excellent reliability, the CPRI protocol establishes a standardized interface for coupling the BBU and RRU of a wireless base station [52].

The CPRI protocol is widely used in 4G and 5G wireless communication systems to support complicated signal processing and high-speed data transfer. It better satisfies the demands of various applications by distributing the wireless base station's signal processing duties between the BBU and RRU [53]. The CPRI protocol lowers network operating expenses while enhancing wireless networks' flexibility and reliability. While the data connection layer establishes frame formats, synchronization, and error checking, the physical layer sets variables such as fiber transmission rate, coding mechanism, and optical power [53].

CPRI and enhanced CPRI (eCPRI) are shown in Figure 3.16. CPRI establishes the essential interface requirements between radio base stations used in cellular wireless networks' REC (Radio Equipment Control) and RE (Radio Equipment). It is the preferred protocol for fronthaul commu-

nications between towers and base stations throughout multiple wireless network generations. For many different technologies, including GSM, WCDMA, LTE, etc., CPRI features effective and adaptable I/Q data interfaces [54].

Flexible fronthaul designs are needed when new 5G applications are launched. The eCPRI specification, released after CPRI, establishes the fronthaul transport network connection between eREC and eRE and is mostly utilized with 5G systems, including LTE-Advanced and LTE-Advanced Pro. Additionally, eCPRI can balance crucial aspects of latency, throughput, and reliability needs for sophisticated 5G applications, and it is anticipated that it will be employed to promote 5G by facilitating higher efficiency. Additionally, eCPRI is an open interface that enables carriers to cooperate in a more beneficial fashion, which may result in speedier globally connected networks [54].



(a) CPRI **(b)** eCPRI

Figure 3.16: CPRI and eCPRI [54]

4

Robust Header Compression

This chapter gives an overview description of the RoHC (including version 1 and version 2) algorithms. Section 4.1 introduces the header compression framework for compressor and decompressor state machines, three types of operations, and encoding methods. The data structure, packet types, operations in three modes, modes transitions, packet formats, and CRC calculation are described in section 4.2. The content in this chapter is based on IETF RFC 3095 and RFC 5225 [26, 35].

4.1. Header Compression Framework

4.1.1. Introduction

The core of the RoHC header compression algorithm is to compress the unnecessary information in protocol header fields to decrease the size of the sending packets. There are several types of fields in the protocol header. In a stable transmission flow, some fields are static, some fields are pre-defined, and some fields are dynamic with a predictable mechanism. In this case, sending a smaller part of these fields by the pre-defined protocols between the sender and receiver is possible. Based on the analysis of the properties of the fields in the protocol headers, the header compression based on the RoHC algorithm is feasible according to their characteristics.

A few definitions of the RoHC algorithm are shown below:

1. Compressor and Decompressor: RoHC algorithm comprises the compressor and decompressor for the header compression and decompression procedure.

2. Channel: RoHC defines the unidirectional transmission channels. Packet flows with RoHC headers are transmitted in the RoHC channel.
3. Context: The context is the buffer in the compressor and decompressor to store the necessary information of the protocol header fields. The context in the compressor and the decompressor will assist the compression and decompression procedures.
4. Context Identifier (CID): The identity of the RoHC packet.
5. Packet Type Indication: The header compression technique indicates the packet type. There are multiple types of packets, including the uncompressed packet, Initialization and Refresh (IR) Packet, Dynamic (DYN) Packet, and Compressed Packet.
6. Cyclic Redundancy Check (CRC) Checksum: CRC is used to verify the correctness of the contents in the RoHC headers.

RoHC algorithm mainly comprises two parts: the compressor and the decompressor. The compressor is used to compress the particular protocol headers for header compression based on the special mechanism and rules. The decompressor is applied to decompress the RoHC headers transmitted from the compressor to obtain the original headers based on the same mechanism and rules. The visualization of the compressor and the decompressor is shown in Figure 4.1.

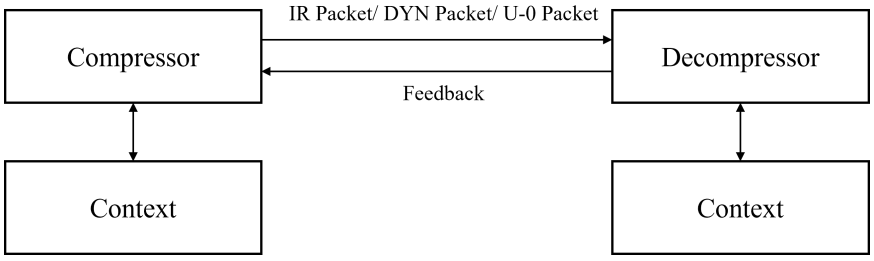


Figure 4.1: Visualization of RoHC Algorithm

In the initialization phase, one or more unidirectional channels are created between the compressor and the decompressor for packet transmission. The context is then created in the compressor and the decompressor. The context is the buffer to store the information of the protocol header fields to assist the compression and the decompression procedure. For instance, the context will store the fields of the protocol headers if they are static in a stable transmission flow. These static fields will not be sent once the RoHC algorithm works stable. The static fields will be recovered at the decompressor by reading the context. A detailed explanation is shown later. The IR packet is sent in this phase to synchronize the compressor and the decompressor context. The unique CID is assigned to the context and the IR packet for identifying different pairs of contexts in the compressor and the decompressor. Besides, the packet type indication is also allocated to the sending packet to identify the packet type.

The IR packet is the largest packet in the RoHC algorithm since it stores all the necessary data for decompression. The DYN packet is smaller than the IR packet because the static fields are excluded in the DYN packet. Once the initialization phase is finished, the compressor will move

to a higher state for sending smaller packets. The DYN packet is applied in this case. When this higher state is finished, the compressor moves to the optimal state. The optimally compressed packet (U-0) packet is sent in this phase. The feedback is also sent from the decompressor to the compressor from time to time for verification. The above-mentioned procedure is controlled by the state machine for compression (shown in Figure 4.2), which will be discussed later.

RoHC header compression may be described as the interplay of three state machines, one compressor, one decompressor, and one for working mode transition. The compressor and decompressor have three states connected in many ways while having slightly different meanings for each party. Both machines begin at the lowest compression level and gradually transition to higher compression stages [26].

RoHC algorithm can be applied in diverse environments [10]:

- Wireless Communication Networks: Due to the constrained bandwidth of wireless communication networks, RoHC can optimize network efficiency by reducing data transmission by compressing packet header information. For instance, RoHC can be used to reduce the size of the headers of multimedia data packets like voice and video in wireless networks like Wi-Fi, 3G, 4G, and 5G mobile communications.
- VoIP: In VoIP systems, voice data is usually packaged into small packets for higher transmission efficiency. The protocol header would be comparatively large in this case compared with the payload. The use of RoHC can significantly reduce the size of the protocol headers.
- IoT Devices: Many devices with constrained bandwidth are used in Internet of Things applications. RoHC is used in this case to compress the headers to reduce the size of the packets.
- Between Network Devices and Servers: RoHC can also be used in communications between servers and network devices (such as routers, switches, etc.) to decrease packet size to improve network transmission effectiveness.

4.1.2. Compressor States

4.1.2.1 Overview of Compressor States

The three compressor states for RoHC compression are Initialization and Refresh (IR), First Order (FO), and Second Order (SO), as shown in Figure 4.2. Starting at the lowest compression (IR) level, the compressor gradually moves to higher compression levels [26].

It's necessary for the compressor to build the context, which will assist the compression and the packet creation procedures. In the IR state, all the information in the context will be built. In the FO state, the dynamic field in the context will be fixed. In the SO state, the context will be unchanged, only for compressed packet building.

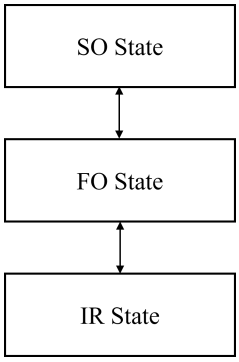


Figure 4.2: Three Compressor States [26]

4.1.2.2 IR State

The IR state is applied to build the content for the context at the decompressor part. The IR packet is sent in this state. All the fields in the protocol headers except the pre-defined message will be transmitted. The compressor remains in the IR state until the Static field in the context has been built [26].

4.1.2.3 FO State

The FO state is the next state after the IR state. The purpose of the FO state is to transmit a smaller packet than the IR state. The dynamic field in the protocol header will be transmitted in this state. It remains in the FO state until it is certain that the decompressor has obtained all of the new pattern's parameters [26].

4.1.2.4 SO State

SO state is the optimal state for compression. The state machine switches to the SO state when the algorithm is fully confident that all the required data has been collected from the former transmission. In this case, the SO state will provide the compression with the highest header compression rate. When the header no longer conforms to the uniform pattern and cannot be independently compressed based on prior context information, the compressor exits this stage and returns to the FO state [26].

An example of mode transition for the compressor is shown in Figure 4.3. The optimistic approach means continuously sending the packets until the state machine is confident that the context is built successfully.

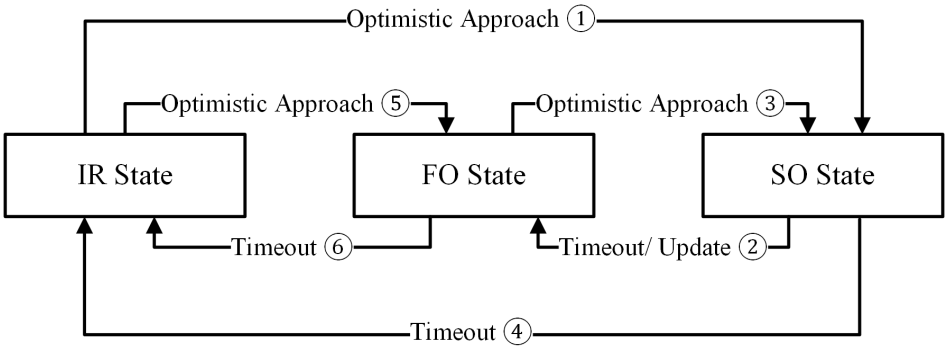


Figure 4.3: Mode Transition in Compressor [26]

In step 1, the state machine switches from the IR state to the SO state by the optimistic approach (sending IR packets), which means that the context of the compressor is successfully built. In step 2, the state machine switches from the SO state to the FO state because of possibly two reasons: an error occurs at the receiver (decompressor), the context needs to update; the timeout happens at the compressor. There is a timeout setting for the FO state and the SO state. The timeout is triggered if the corresponding feedback is not received in time. In step 3, the state machine switches back to the SO state by sending the DYN packet. In step 4, the timeout for the IR state happens, and the state machine switches to the IR state. In step 5, the compressor sends the IR packet, but only the static field is successfully built. In this case, the state machine switches to the FO state. In step 6, the timeout happens again, the state machine switches to the IR state. In this case, the state machine needs to reinitialize for compression.

4.1.3. Decompressor States

There are three states in the state machine of the decompressor as shown in Figure 4.4: No Context, Static Context, and Full Context. The state machine starts at the No Context state at the beginning without any useful data. After a period of IR packet collections, the machine can switch to the Static Context and then to the Full Context, or directly to the Full Context if it's confident that the decompressor will decompress the compressed packet [26].

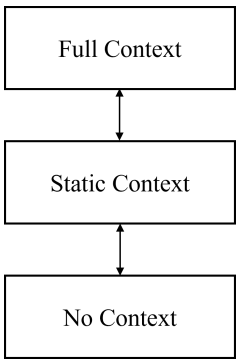


Figure 4.4: Three Decompressor States [26]

It's necessary for the decompressor to build the context. In the No Context state, the context is blank, waiting for the IR packet to build the content. In the Static Context state, the context only stores the content for the static fields. The DYN packet is waiting. In the Full Context state, the context is finished, and no more information is required.

An example of the state machine transition in the decompressor is shown in Figure 4.5.

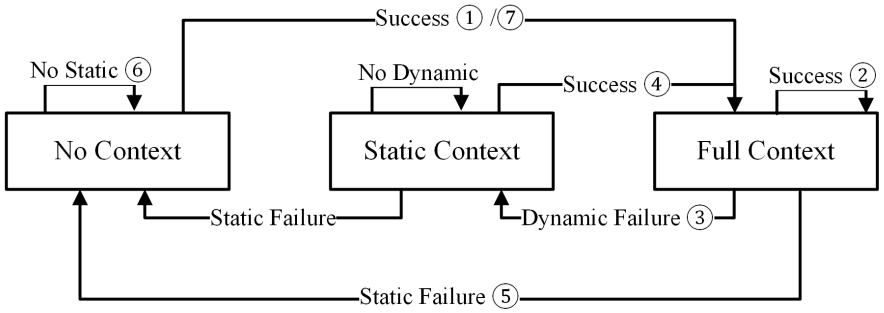


Figure 4.5: Mode Transition in Decompressor [26]

In step 1, the state machine switches from No Context to Full Context when the context of the decompressor is successfully built. In step 2, the state machine will keep the Full Context state if the decompression is successful all the time. In step 3, the state machine switches to Static Context when the dynamic fields error occurs, the state can fix the dynamic field of the context in this state. In step 4, the state machine switches back to Full Context when the dynamic field is fixed; otherwise, it will maintain in Static Context. In step 5, the static fields error occurs, the state machine switches to No Context state for fixing the static and the dynamic fields of the context. In step 6, the state machine holds still when fixing the context. In step 7, the state machine switches to Full Context for decompressing the packets again. In No Context state, only uncompressed packets can be read. In Static Context, the Static packets can be read. In Full Context, all the packets can be decompressed.

4.1.4. Modes of Operation

RoHC has three operational modes: Unidirectional, Bidirectional Optimistic, and Bidirectional Reliable, shown in Figure 4.6. In every operational mode, the state machine in the compressor is the same as illustrated in Figure 4.2.

The properties of the compression protocol's environment, such as feedback capabilities, error probabilities, distributions, the effects of header size changes, determine the best mode to use. All RoHC implementations must support and implement all three operational methods [26].

The decompressor decides when to switch from one compression mode to another. It also shows the possible mode changes. Subsequent sections detail how transitions are executed and com-

pression and decompression functionality exceptions during transitions. A detailed explanation of the mode transition is provided in Appendix C.

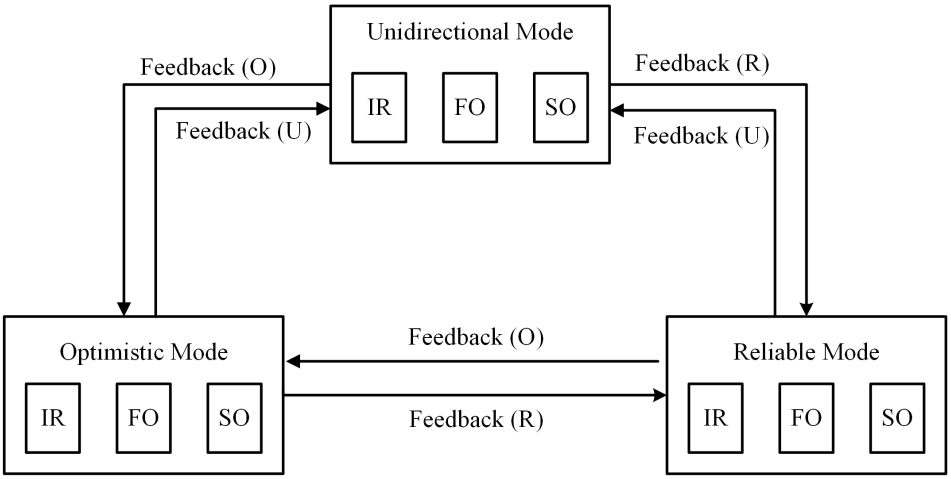


Figure 4.6: Three Modes of Operation for RoHC [26]

4.1.4.1 Unidirectional mode -- U-mode

In the unidirectional mode [26], packets are transferred from the compressor to the decompressor. This mode allows RoHC to be used on connections where a return path from decompressor to compressor is either unavailable or undesirable.

In U-mode, changes between compressor states are only caused by timeouts and strange change in the header field change patterns of the compressed packet stream. Due to the periodic refreshes and absence of input for error recovery, compression in the unidirectional mode will be less efficient and have a slightly higher likelihood of loss propagation than any bidirectional method. RoHC compression must begin in unidirectional mode [26].

4.1.4.2 Bidirectional Optimistic mode -- O-mode

The bidirectional optimistic mode [26] resembles the unidirectional mode. A feedback channel sends requests for error recovery and (optionally) acknowledgments of significant context changes from the decompressor to the compressor. Sequence number updates are not sent through this channel. In the bidirectional optimistic mode, periodic refreshes are not employed. O-mode attempts to enhance compression efficiency with minimal feedback channel use. It reduces the number of false headers sent to higher tiers due to lingering faults or inappropriate context. It is possible for context invalidation to occur more frequently than in R-mode, mainly when extended losses or error bursts occur [26].

4.1.4.3 Bidirectional Reliable mode -- R-mode

The bidirectional reliable mode [26] differs from the two preceding modes in several ways. The most significant changes are a more robust use of the feedback channel and stricter logic at both the compressor and the decompressor. This mechanism keeps the compressor and decompressor from losing context synchronization. All context adjustments, including sequence number field updates, are acknowledged with a response. In reliable mode, context is not updated for every packet [26].

4.1.5. Least Significant Bits (LSB) Encoding

Some fields in the protocol headers are changing but predictable in the transmission flow. These fields can be compressed by the LSB encoding mechanism.

The LSB encoding is applied to compressing these changing but predictable fields. After LSB encoding, only k least significant bits are transmitted instead of the real data. For the compressor and the decompressor, a reference value V_ref is stored in each context to assist the compression and the decompression. The interpretation interval based on the V_ref is shown as follows [26]:

$$f(v\_ref, k) = [v\_ref - p, v\_ref + (2^k - 1) - p] \tag{4.1}$$

where k is a positive integer, p is an integer.

The parameter p is introduced to shift the interpretation interval. The original data is obtained at the receiver by matching the received LSB value with the value in the interval. The V_ref is updated by the latest compressed/ decompressed value at the sender/ receiver [26].

4.2. The Protocol of RoHC

4.2.1. Data Structure

In the ISO/OSI network protocol layer paradigm, the RoHC compression method is often implemented above the data link layer. When used to hide the original headers of higher protocol levels from lower protocol layers, packet header compression is also frequently referred to as an adoption layer or a cross-layer approach. Additionally, packet header compression can be viewed as a source coding technique rather than a channel coding method in data transmission [10]. Figure 4.7 illustrates packet header compression via the simplified example of H.264/Advanced Video Coding (AVC) video transmission with RoHC.

Figure 4.7 illustrates the OSI/ISO model with an example of transmission of H.264/AVC Video

Frames (VF) via Network Abstraction Layer (NAL), RTP, UDP and IP compressed by RoHC. For packet transmission through the Ethernet data link layer and the underlying physical layer network, the RoHC compressor at the sender substitutes smaller RoHC compressed headers for the RTP/UDP/IP packet headers in the example shown. To recover the original RTP/UDP/IP headers, the RoHC decompressor at the receiver decompresses the compressed headers [10].

The protocol headers of the network protocol layer and higher protocol layers are commonly compressed through packet header compression; the data link layer protocol headers have rarely been considered [10].

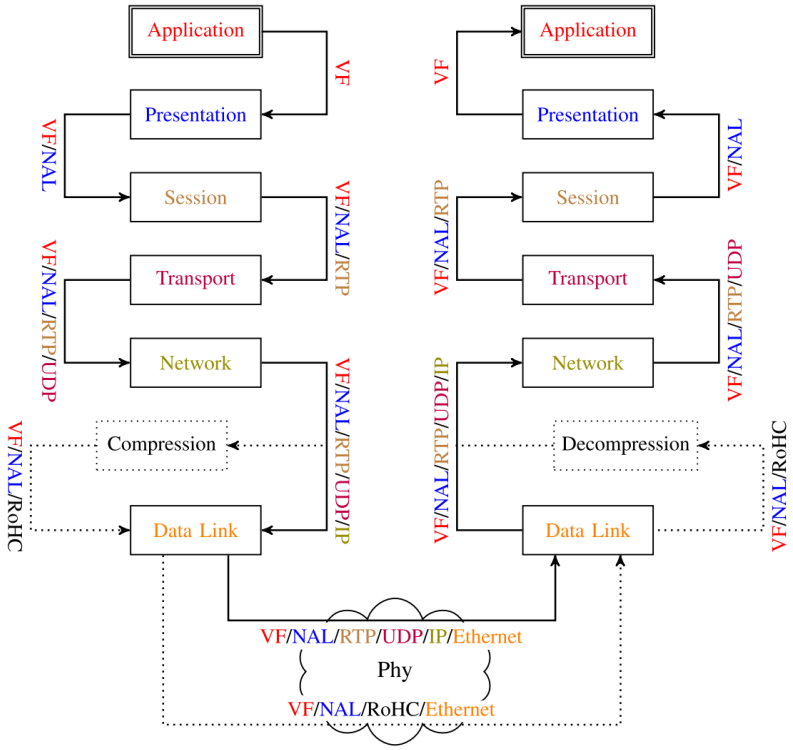


Figure 4.7: The OSI model's protocol stack is where packet header compression is typically found [10]

4.2.1.1. Per-context Parameters, Profiles

With IR headers, context-specific parameters are set. The profile identifier in an IR header dictates how the remainder of the header is to be interpreted. The profile parameter controls the packet type identifiers and types utilized in a particular context [26]. Sending uncompressed IP packets is achieved by profile 0x0000, profile 0x0001 is for RTP/UDP/IP compression, and profile 0x0002 is for UDP/IP compression in RoHCv1. For RoHCv2, profile 0x0101 is for RTP/UDP/IP compression, profile 0x0102 is for UDP/IP compression, profile 0x0104 is for IP compression, profile 0x0107 is for RTP/UDP-Lite/IP compression, and profile 0x0108 is for UDP-Lite/IP compression [26].

4.2.1.2. Contexts and Context Identifiers

Contexts are applied to store the necessary information for compression and decompression. The contexts guide the compression and decompression procedures by adding or deleting the fields in the protocol headers. In this case, the context is similar to a comparison chart [26].

The CID are then the identity of the context. In some cases, the context in the decompressor needs to be updated. The compressor will send the IR or DYN packet to sync the data in the context. The CID is created simultaneously to distinguish different contexts during transmission [26].

4.2.2. RoHC Packets and Packet Types

A comprehensive description of the RoHC packets is provided in Appendix C. Only key RoHC packets are illustrated here, including IR packets, DYNAMIC packets, and U-0 packets [26].

4.2.2.1. RoHC IR packet Type

In addition to associating a CID with a profile, the IR header often initializes the context. Typically, it can also update (parts of) the context. The generic format is indicated in Figure 4.8 [26].

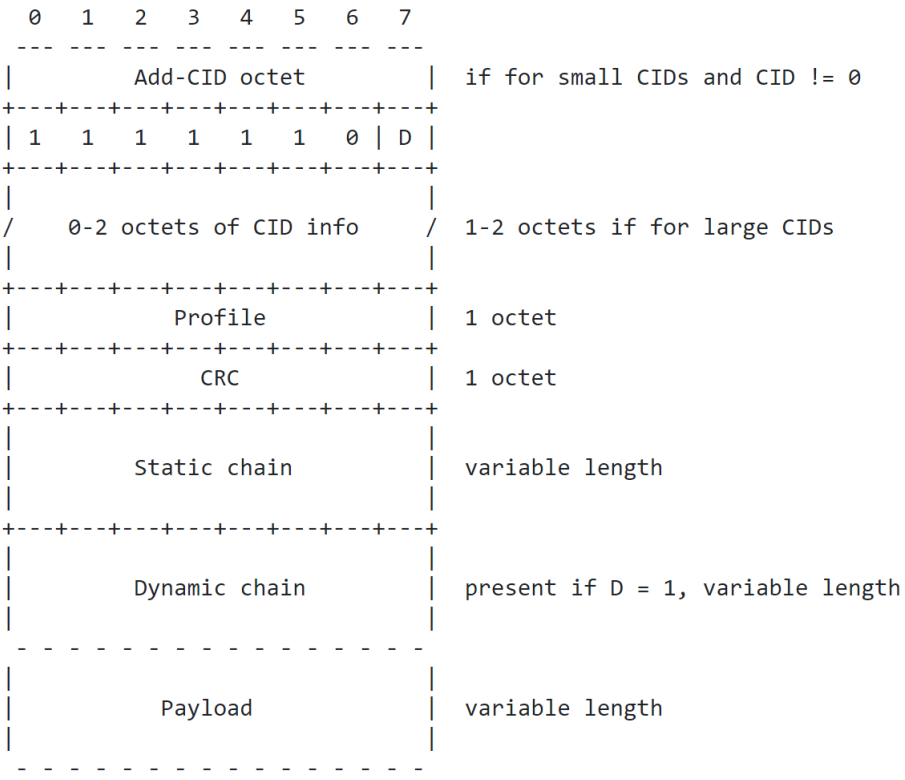


Figure 4.8: Basic Format of IR Packet [26]

where:

- D: Profile-specific information. The dynamic chain is used when D is 1.
- Profile: Information is particular to the profile. The text is interpreted based on the profile specified in the Profile field.
- CRC: Computed 8-bit CRC. Its range depends on the profile, but it must include at least the part of the packet that starts before the Profile field and ends with it. The CRC should be applied to any information that initializes the decompressor's context.
- Static chain: A contest field containing STATIC sub-headers.
- Dynamic chain: A contest field containing DYNAMIC sub-headers.
- Payload: The field that includes the payload of the package.

4.2.2.2. RoHC DYNAMIC Packet Type

The DYNAMIC header cannot ever initialize a context that has not been initialized. Nonetheless, it can modify the profile that is connected to a context. Consequently, the type should be reserved at the framework level. Typically, the IR-DYNAMIC header also initializes or refreshes the dynamic portion of a context. The general format is illustrated in Figure 4.9 [26].

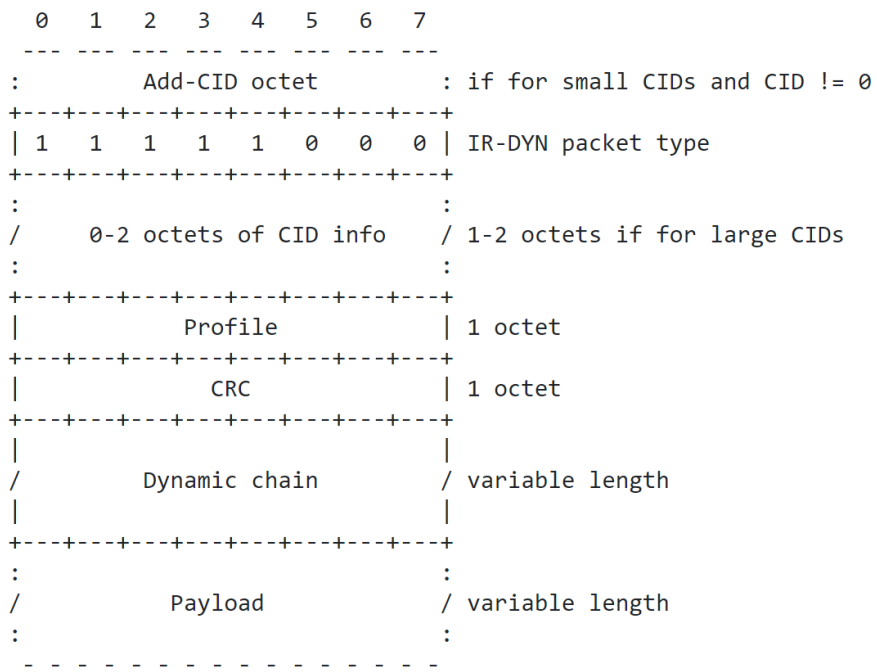


Figure 4.9: Basic Format of DYNAMIC Packet [26]

4.2.2.3. RoHC U-0 Packet Type

The RoHC U-0 packet [26] is used when the RoHC compression is optimal, for example, in the SO state. The U-0 packet is only one byte, including a 1-bit indicator + a 4-bit sequence number

+ a 3-bit CRC check [26].

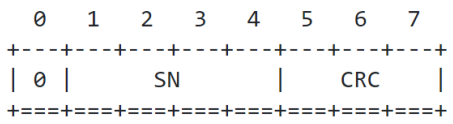


Figure 4.10: Format of U-0 Packet [26]

4.2.3. Header Compression CRCs

4.2.3.1. IR and DYN Packet CRCs

The CRC is computed over the IR or DYN packet, except Payload and including CID or any Add-CID byte. The CRC field in the header is set to 0 to calculate the CRC. The initial value of the CRC register should be all ones. The CRC polynomial for the 8-bit CRC is as follows [26]:

$$C(x) = 1 + x + x^2 + x^8 \tag{4.2}$$

where x means the digit of the polynomial.

For instance, the polynomial for the above equation is $2^0 + 2^1 + 2^2 = 7$ or 0x07 in hexadecimal. x^8 means the width of CRC is 8. This hexadecimal will be used as the divisor in CRC calculation.

4.2.3.2. Compressed Headers CRCs

Before compression, the CRC in compressed headers is computed over all octets in the full original header. The initial value of the CRC register should be all ones. The CRC polynomial for the 7-bit CRC is as follows [26]:

$$C(x) = 1 + x + x^2 + x^3 + x^6 + x^7 \tag{4.3}$$

where the polynomial is $2^0 + 2^1 + 2^2 + 2^3 + 2^6 = 79$ or 0x4f in hexadecimal. The width is 7.

The CRC polynomial for the 3-bit CRC is as follows :

$$C(x) = 1 + x + x^3 \tag{4.4}$$

where the polynomial is $2^0 + 2^1 = 3$ or 0x03 in hexadecimal. The width is 3.

A detailed calculation of CRC is illustrated in Appendix A.

4.3. Application of RoHC Algorithm

The application of the RoHC algorithm in the proposed algorithm in this thesis is shown in Table 4.1

Table 4.1: Application of RoHC Algorithm

Components	Useful Parts	Useless Parts
Working Mechanism of Compressor and Decompressor	Related to RTP/UDP/IP compression	Related to Encapsulating Security Payload Header (ESP) and UDP-lite
Packet Design	Related to IR, DYN, and U-0	Other Packets
State Machine	Related to the compressor and the decompressor	Related to the mode transition and feedback
Calculating Algorithm	CRC, LSB	Other algorithms like W-LSB

For the working mechanism in the compressor and the decompressor, the proposed algorithm only inherits the mechanism for the RTP/UDP/IP headers instead of ESP and UDP-lite headers. For the packet design, only IR, DYN and U-0 packets will be used in this thesis. For the state machine in the compressor, decompressor and mode transition, the mode transition state machine will be excluded since only the unidirectional mode will be used in this thesis. The robustness of the communication relys on the RTP and RTCP protocol instead of the compression algorithm. For the calculating algorithm, only CRC and LSB algorithm will be used.

4.3.1. Fields Classification

In a protocol header, there are fields for defining some properties. Some fields are static in the stable data flow, some fields are pre-defined before the data transmission, and some fields change over the data flow. The optimized header compression algorithm will compress these fields according to their characteristics. For simplicity and clarity, these fields in each protocol header are classified into five categories [26]:

1. INFERRED: These fields contain information that the compression scheme does not need to handle, such as values that may be inferred from other values like the checksum in IPv4 or the size of the frame carrying the packet.
2. STATIC: It is anticipated that these fields won't change throughout the packet stream. Like the IP version, static information must be conveyed only once.
3. STATIC-DEF: A packet stream is defined by the values of static fields. They are typically treated as STATIC, similar to SSRC in RTP.
4. STATIC-KNOWN: Similar to the sole version of RTP currently available, these STATIC fields are anticipated to have well-known values and do not thus require any communication.

5. CHANGING: These fields are anticipated to vary in some way, whether it be randomly, within a specific value range, or in another way. A comparison between the context in the compressor and the packet data is required to locate the changing fields. The corresponding fields will be updated in the subsequent transmission.

4.3.1.1. Fields Classification for IPv4

The IPv4 header fields are shown in Table 4.2:

Table 4.2: IPv4 Header Fields

4 Octets							
0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
Version	Header Length	Type of Service		Packet Length			
Identification				Flag			
Time to live		Next Protocol		Header Checksum			
Source Address				Destination Address			

- Version: This field's values must be handled by distinct IP stacks for packets with varying values. A packet stream must consequently contain only packets of the same IP version. The field is categorized as STATIC as a result.
- Header Length: The IP header length is consistent and well-known as long as options are absent. The fields would be STATIC if there were options, but it is presumed that there aren't any in this case. As a result, the field is categorized as STATIC-KNOWN.
- Type of Service: The type of service in IPv4 is inconsistent and possibly changes after a transmission period. The context check and update must be achieved to guarantee correctness. Consequently, it's classified as CHANGING.
- Packet Length: The link layer should offer packet length details. As a result, the field is categorized as INFERRED.
- Identification: This field is randomly created based on the time and transmission in case of packet slicing. Then it's classified as CHANGING. However, this field is useless in the OH algorithm since only small packets will be transmitted. There will be no packet slicing.
- Flag: Two bytes of fields related to the fragment are categorized as STATIC-KNOWN since packet slicing will not be used in this paper. It's defined as default ahead of time.
- Time to live: This field identifies the duration of the data packet. It's possibly changing over the stream. Consequently, this field is classified as CHANGING.
- Next Protocol: This field typically has the same value throughout all packets in a packet stream. The type of the following header is encoded. Only when extension headers are occasionally present and occasionally absent can the field's value fluctuate throughout a stream. As a result, the field is categorized as STATIC.

- Header Checksum: Individual hops are shielded from processing a damaged header by the header checksum. There is no need for this additional checksum because practically all IP header data can be compressed away; it can be created again at the decompressor side. As a result, the field is categorized as INFERRED.
- Source and Destination addresses: These fields must remain constant for all packets in the stream because they are necessary for the definition of a stream. As a result, the fields are categorized as STATIC-DEF.

4.3.1.2. Fields Classification for UDP

The UDP header fields are shown in Table 4.3:

Table 4.3: UDP Header Fields

4 Octets							
0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
Source Port				Destination Port			
Length				Checksum			

- Source and Destination ports: These fields must remain constant for all packets in the stream because they are necessary for the definition of a stream. As a result, the fields are categorized as STATIC-DEF.
- Length: This field is categorized as INFERRED due to its duplication.
- Checksum: The checksum in the UDP is calculated through the IPv4 header, UDP header, and payload. In this case, it's hard to predict the value of the checksum ahead of time. It's marked as CHANGING.

4.3.1.3. Fields Classification for RTP

Table 4.4 shows the RTP header fields.

Table 4.4: RTP Header Fields

4 Octets							
0-3	4-7	8-11	12-15	16-19	20-23	24-27	28-31
Version and Extension	CSRC Counter	Payload Type		Sequence Number			
Timestamp				SSRC			

- Version: There is just one version of RTP that is operational, version 2. As a result, the field is categorized as STATIC-KNOWN.

- Extension: Although extension headers are extremely uncommon, the extension flag will likely be found in every packet. But for safety reasons, this field is categorized as STATIC rather than STATIC-KNOWN.
- CSRC Counter: This field identifies the number of CSRCs in the CSRC field. However, this field is classified as STATIC-KNOWN since CSRC is not considered in this paper for simplicity.
- Payload Type: This field instructs the payload type, like video or audio packets. It will change over the stream if the function or the compression algorithm has changed. Consequently, it's defined as CHANGING.
- Sequence Number: This field is used to find the order of the packet stream in the receiver and combine them within the correct order. It's successively changing over the packet stream. In this case, it's defined as CHANGING. Moreover, it's also predictable in some cases since the regulation of creating sequence numbers is predictable. For example, it's feasible to compress this field using the LSB algorithm.
- Timestamp: This field is classified as CHANGING since it's updated over the stream. However, it's sometimes predictable by calculating the stride of the time to foresee the next timestamp in the next packet.
- SSRC: Since this field is essential to the definition of a stream, it must remain consistent during the whole stream of packets. As a result, the field is categorized as STATIC-DEF.

4.3.2. Compressed Headers in RoHC

The present RoHC compressor is used to compress RTP/UDP/IP protocols.

The IR header and the DYN header for RoHC are shown in Appendix C. The compressed header for RoHC is shown in Table 4.5 [26].

Table 4.5: Structure of the RoHC Header for RoHC Compressor [26]

Octet							
8	7	6	5	4	3	2	1
0	SN_RTP				CRC		
UDP Checksum_1							
UDP Checksum_2							

The RoHC header replaces the IP/UDP/RTP header combination. The sequence number in RTP is set to 4 bits after the zero indicator. The CRC is used to guarantee the correctness of SN_RTP. The two bytes of the UDP checksum follow later since the UDP checksum is unpredictable.

4.4. Proposal for A New Header Compression Algorithm

The Optimized Header (OH) Compression Algorithm, based on the RoHC algorithm, is proposed to optimize the headers of real-time IP packets across the 5G protocol stacks. Additionally, to further enhance header compression in multiple QoS flows (parallel voice and video flows, for example) transmissions within the same PDU session, the Multi-stream Optimized Header (MOH) Compression Algorithm is introduced. Furthermore, in telemedicine smartwatch IP services, the Vertical Multi-layer Optimized Header (VOH) Compression Algorithm is proposed to coordinate header compression across multiple protocol layers.

In the first scenario of applying the OH algorithm, the compression is toward the voice-over IP service. Compared with the RoHC algorithm applied in this service before, more protocol headers are compressible in the proposed OH algorithm, like GTP-U, PDCP, and SDAP headers. Therefore, a smaller OH header will be added to the payload, which leads to a smaller data packet. A smaller packet in data transmission will require less transmission throughput in the same packet sending rate, improving transmission efficiency. The simulation based on the OH algorithm is conducted for VoIP payload in 5GS over F1-U and N3 reference points since the proposed algorithm will only be implemented over both reference points. Further explanation will be provided in Chapter 5.

In the second scenario of applying the MOH algorithm, the compression is toward the duplicate information between headers in the voice-video combined IP service. For example, using two QoS flows for voice and video over IP services in the same PDU session. There will be extra redundancy between headers, like fields to define PDU information. In this case, a further optimization called the MOH based on the proposed OH algorithm is necessary.

In the third scenario of applying the VOH algorithm, the headers are compressed coordinately in a vertical manner for the smartwatch telemedicine service mentioned in Figure 1.4. For instance, the RTP stream transmits telemedicine IP packets, passing through two IP and two UDP protocols (the inner and the outer layer protocols, the inner protocols are with the GTP for tunneling). The compression is feasible between these inner and outer protocol headers. Therefore, a further optimization called the VOH algorithm based on the proposed OH algorithm is required.

Due to propagation delay, the distance and the transmission speed influence the transmission latency. A smaller packet means fewer transmission resources, like less transmitted bits per second, flexible packet sending rate for congestion, and fewer bytes for loading and routing, consequently speeding up the transmission procedure and decreasing the overall latency.

5

Optimized Header Compression Algorithm Design

In this chapter, the optimized header compression algorithm based on the RoHC algorithm is proposed to adapt to the new challenges in 5G communications. Compared with the original RoHC algorithm, the OH algorithm provides more compression options and compresses more fields of headers to improve transmission efficiency and decrease latency. 5G-applied protocols like GTP-U, PDCP, and SDAP will also be compressible in the OH algorithm for more efficient communications.

The consideration of the compression decision and the overall architecture of the optimized header compression algorithm is introduced first. Besides, the classification of the headers and the approaches to compressing the categorized headers are stated. Moreover, the working mechanisms of the OH compressor and the OH decompressor are illustrated, respectively. Finally, a theoretical compressing calculation for both voice-over-IP and video-over-IP is conducted to explore the optimal performance of the OH algorithm.

5.1. Overview of Optimized Header Compression Algorithm

Based on the analysis above in Chapter 3 and Chapter 4, it's apparent that there are enormous possibilities for compressing headers in the data flow. However, factors like stability, flexibility, and implementation costs should also be considered. Before deciding on the solution, a compromise between data efficiency, scalability, and device cost is required.

Firstly, further header compression over the air interface (RLC/MAC/PHY layers) is not viable.

On the one hand, a new compression algorithm achievement means more complex circuits or implementing peripheral devices. Finding the solution for the NR-Uu reference point is not feasible since UE-like mobile phones are less scalable. Applying a new circuit design to a tiny mobile phone will bring about too many design problems. On the other hand, the problem of protocol standardization is not easy to solve since a more sophisticated algorithm is required to compromise the header compression algorithm and the NR-Uu interface. Therefore, applying the optimized compression algorithm over the NR-Uu reference point is currently not feasible.

Applying an optimized header compression algorithm at F1-U reference points is possible for several reasons. The complexity of implementing the algorithm will be low because of the high flexibility of distributed and centralized units in gNB. DU and CU devices are primarily equipped in independent boxes, respectively. The difficulty of requesting more resources for the optimized header algorithm is acceptable. In the meantime, there is no need to use high-tech craft technology for manufacturing circuits because of the availability of space in the existing facilities. Consequently, the optimized header compression solution will be implemented at the F1-U reference point.

Besides, applying the optimized header compression algorithm between gNB-CU-UP and UPF at N3 reference points is feasible. The comparative low complexity and cost is one reason since applying the optimized header algorithm at the N3 reference point is similar to that at the F1-U reference point. The computing resource is available, and the cost is acceptable. Moreover, in real-life 5G network construction, the RAN and core network are segregated in different locations. In most cases, a multitude of gNBs are controlled by one UPF. Therefore, it's inevitable to place gNB and UPF over long distances to improve the utilization of resources. In this case, the high latency because of the bulk overhead will be much more apparent and tricky. Consequently, applying the optimized header compression solution at the N3 reference point is advantageous.

The N6 reference point between the UPF and the data network is not applicable for the optimized header compression because of the implementation difficulties. There are always too many routing actions between the UPF and the DN. A pair of compressors and decompressors will be required for each routing action. Therefore, applying the proposed design at this point is not only less economical but also less efficient. The final choice for using the optimized header compression solution is shown in Figure 5.1. Two GTP tunnel sections are marked in red to represent the choice of implementing the optimized solution at the F1-U and the N3 reference points.

A more detailed explanation of implementing the optimized header compression algorithm is shown in Figure 5.2. At most, three pairs of compressors and decompressors are created in the OH application, called OH_1, OH_2 and OH_3. OH_1 originates from the RoHC algorithm with profile RTP/UDP/IP. OH_2 is the extended version of OH_1, based on the same working mechanism in the RoHC algorithm, but for compressing the protocol headers (SDAP and PDCP protocol headers) that are not included in the RoHC algorithm. OH_3 is also the extended version of OH_1, based on the same working mechanism for compressing the protocol header like

GTP-U/UDP/IP that does not exist in the RoHC algorithm. OH_1, OH_2 and OH_3 work together to achieve the proposed optimized header compression algorithm. A detailed explanation of the working mechanism for the three compressor and decompressor pairs are provided later in this chapter.

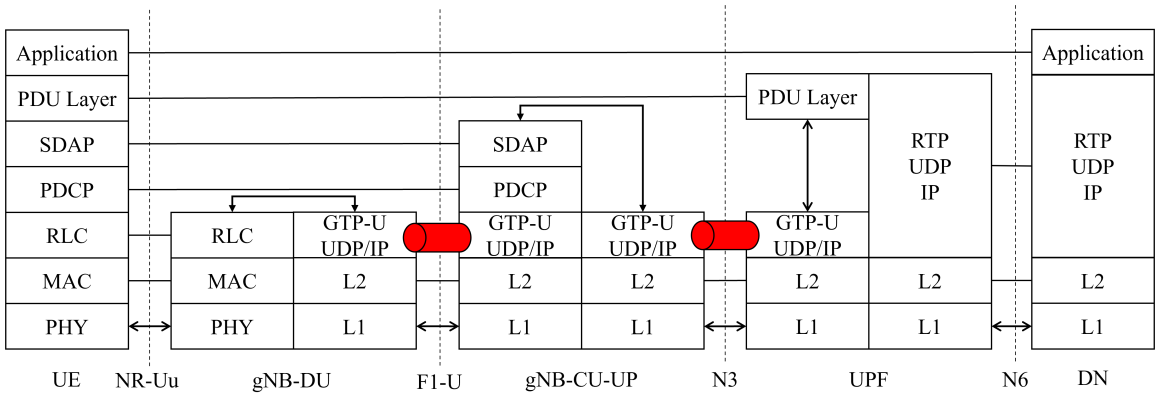


Figure 5.1: Overall UP Protocol Stack with optimizable marks

The pair of OH_1 is designed to compress the headers in the PDU layer (RTP/UDP/IP). The OH_1 is inherited from the original RoHC RTP/UDP/IP protocol algorithm. The initialization of OH_1 is executed in gNB-DU. The OH_1 header-optimized packets will be transmitted through the F1-U and the N3 reference points. The data flow arrives at the UPF. The OH_1 decompressor is created in the UPF to decompress the OH_1 header-optimized packets and offer the original IP/UDP/RTP data flow (in the PDU layer) to the UPF for the successive transmission. There is no action for OH_1 data flow at the gNB-CU-UP entity.

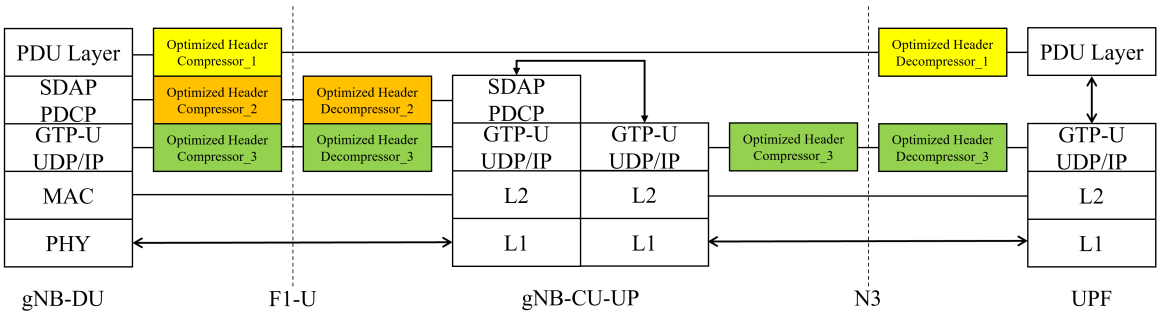


Figure 5.2: Implementation of Optimized header compression algorithm in UP Protocol Stack

The pair of OH_2 is applied to compress the 5G RAN protocols (PDCP and SDAP) headers. Unless the OH_1, the OH_2 is the brand new part in the OH algorithm. The initialization of OH_2 is executed in gNB-DU. The difference between the OH_1 and the OH_2 is in the decompressor. The OH_2 only performs the compression functions over the transmission between gNB-DU and gNB-CU-UP. The SDAP and PDCP layers serve as the data connection between UE and the gNB-CU. There will be no PDCP and SDAP data passing through CU. In this case, the decompressor belonging to OH_2 will be installed in gNB-CU-UP.

The pair of OH_3 is the most special since it's implemented twice in the F1-U and N3 reference points, respectively, which attributes to the difference of the GTP-U. In the F1-U reference point communication, the PDCP PDU is encapsulated in GTP tunneling protocols. The information in the GTP-U, like TEID, SN, UDP, and IP, will be used to build only the GTP tunneling between DU and the CU in the gNB. The PDU layer data, including IP packets, will be delivered to the UPF by gNB-CU-UP over the N3 reference point by GTP tunneling. However, some of the information in the GTP protocol will be updated since novel tunneling is applied. The field of the header, like TEID, is unique to the tunnel. Moreover, the payload type is also changed, which requires recalculating some fields like the checksum. Consequently, two OH_3 are used in the F1-U and the N3 reference points, respectively. A detailed explanation of the working mechanism for the OH_1, OH_2 and OH_3 will be provided later in this thesis.

Even if three types of OH compressors and decompressors are created and implemented together, they will not still be used independently. On the contrary, the working of three OH blocks will be managed coordinately for serializing the output from these OH blocks. In other words, the output from OH compressors will be collected and sent together. A more detailed explanation of the data flow is illustrated in Figure 5.3.

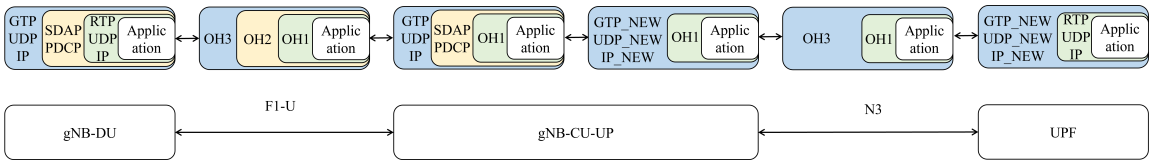


Figure 5.3: New Data flow with OH

Figure 5.3 illustrates the structure of the data packets over the F1-U and the N3 reference points. The far left image (from Application to IP) shows the packet components before compressing in the gNG-DU. The following image (from Application to OH3) illustrates the packet after the OH algorithm has compressed the header. This packet will be transmitted over the F1-U reference point to the gNB-CU. OH1, OH2, and OH3 data represent the data after being compressed by OH_1, OH_2, and OH_3 algorithms, respectively. Application represents the payload of the PDU session. After being decompressed by the algorithm, the packet is shown in the third image (from Application to IP). The protocols, including SDAP, PDCP, GTP, UDP, and IP, are decompressed. Protocols of IP, UDP, and GTP-U are applied to verify the correct GPRS tunneling connection between the DU and the CU in gNG. Afterward, the IP, UDP, and GTP-U protocols are updated for routing in the N3 reference point, as shown in the following image (from Application to IP_NEW). These updated protocols will be compressed again by another pair of OH_3 for transmission at the N3 reference point, which is stated in the next image (from Application to OH3 over OH1). The decompressors will decompress the whole data packet in the UPF entity, as shown in the right image (from Application to IP_NEW). The new IP, UDP, and GTP-U protocols will be applied to verify the correctness of the tunnel connection between the RAN and the core network. The IP packets will be forwarded to the data network for the application.

In this case, when the data transmits from the UE to the gNB-DU, the proposed optimized header compression algorithm does not work. At the gNB-DU, the protocol stack is RTP/ UDP/ IP/ SDAP/ PDCP/ GTP-U/ UDP/ IP. During the packet transmission from the gNB-DU to gNB-CU-UP, the PDU layer (RTP/UDP/IP headers) is compressed to OH_1 header, the SDAP/PDCP layers are compressed to OH_2 header, the GTP-U/UDP/IP headers are compressed to OH_3 header. At the gNB-CU-UP, the SDAP/PDCP headers are terminated since these services finish here. During the packet transmission from the gNB-CU-UP to UPF over the N3 reference point, the OH_1 header is intact, the OH_2 header is disabled, the OH_3 header is recreated by compressing the new GTP-U/UDP/IP headers. At the UPF, the decompressor restores the original headers to RTP/UDP/IP/GTP/UDP/IP. When the data transmits from the UPF to DN, the optimized header compression algorithm does not work. The original header will be sent for routing and locating.

5.2. Profile Analysis of OH Compression Algorithm

When the OH compressor (same for OH_1, OH_2 and OH_3) is initialized, some configurations, including the profile, must be finished beforehand. The profile of OH defines the compressing functions and choices. For example, OH_3 is used to compress headers, including GTP-U/UDP/IP. Its profile is defined as ”GTP/UDP/IP”, which differs from the profile definition of OH_1 for ”RTP/UDP/IP” header compression. In this case, the profiles of three pairs of OH should be considered and created to correspond to their functions. Then, the detailed structure of protocol layers must be analyzed and classified.

Based on the classification of header fields in Chapter 4, the fields of the other protocols used in the OH algorithm are analyzed as shown below.

5.2.1. Fields Classification for GTP-U

The GTP-U header fields are shown in Table 5.1.

Table 5.1: GTP-U Header Fields

<table>
<tr><td colspan="8">4 Octets</td></tr>
<tr><td>0-3</td><td>4-7</td><td>8-11</td><td>12-15</td><td>16-19</td><td>20-23</td><td>24-27</td><td>28-31</td></tr>
<tr><td colspan="2">Version and Flag</td><td colspan="2">Message Type</td><td colspan="4">Length</td></tr>
<tr><td colspan="8">TEID</td></tr>
<tr><td colspan="4">SN</td><td colspan="2">N-PDU Number</td><td colspan="2">Extension Header</td></tr>
<tr><td colspan="2">Header Length</td><td colspan="2">PDU Type</td><td colspan="2">PDU QFI</td><td colspan="2">UL QFI SN</td></tr>
<tr><td colspan="4">UL QFI SN</td><td colspan="4">Padding</td></tr>
</table>

- Version: Only one version of GTP-U will be used in this design, then it’s only transmitted

once, defined as STATIC.

- Extension Flag: It's used to enable the usage of the extension header. The PDU container extension header will be applied in the tail. Then it's only transmitted once and defined as STATIC.
- SN flag: This flag will enable the usage of the sequence number over the whole stream. In this case, it's only transmitted once and classified as STATIC.
- N-PDU Number Flag: It's used to identify the usage of the N-PDU number over the whole stream. This setting will continue throughout the entire transmission. In this case, it's only offered once and defined as STATIC.
- Message Type: This field is applied to define the type of payload message after the header. In most cases, the T-PDU type will be used and held over the whole stream. Consequently, it's classified as STATIC.
- Length: This field indicates the length of the overall packet, including the payload. It's then INFERRED because of the same reason mentioned in IPv4.
- TEID: Tunnel Endpoint Identifier is applied to identify the marked GTP tunnel between two entities. It will be stable over the whole stream. Then it's classified as STATIC-DEF.
- SN: This field will be treated the same as SN mentioned above, CHANGING classification.
- N-PDU number: This field will change as more PDUs are transmitted. It also holds similar behavior as SN. Consequently, it's treated as CHANGING.
- Extension Header: This field is defined as STATIC since only the extension header of the PDU container is considered in this paper.
- Header Length: This field is classified as STATIC since the type of extension header and usage of this header are pre-defined. Consequently, it's only transmitted once for clarification.
- PDU Type: The T-PDU will be used in this paper. For correct configuration, this field should be offered once. It is then defined as STATIC.
- PDU QFI: This field is STATIC since the PDU will be offered the QFI only once for defining the functions. The function of this PDU will continue until it's abandoned.
- UL QFI SN: This field is CHANGING for the same reason as the field of SN. It's changing over the stream.
- Padding: this field is STATIC-DEF since two bytes of padding are defined before initialization.

5.2.2. Fields Classification for PDCP

The PDCP header fields are shown in Table 5.2.

- D/C: This field defines whether it's a Data Radio Bearer (DRB) or a Signal Radio Bearer (SRB). This definition will be used for the whole stream. Consequently, it's classified as STATIC.
- R: This field is STASIC-DEF since it remains 0 all the time.

- PDCP SN: This field is CHANGING for the same reason as other SN in other protocol layers.

Table 5.2: PDCP Header Fields

2 Octets			
0-3	4-7	8-11	12-15
D/C and R	PDCP SN		

5.2.3. Fields Classification for SDAP

The SDAP header fields are shown in Table 5.3.

Table 5.3: SDAP Header Fields

Octet	
0-1	3-7
D/C and R	QFI

5.3. Scheme of Optimized Header

The OH algorithm has three header types: the IR header, the DYN header, and the OH header. The IR header is implemented for initialization in the updating phase for updating the STATIC and CHANGING fields. The DYN header is used for updating the CHANGING field when an error happens in the CHANGING field. The OH header is applied when the compression is optimal, which suggests that the OH header is the smallest among them. The IR header, DYN header and OH1 header used in the OH_1 compressor are originated from IETF RFC 3095. The other headers are newly proposed in this thesis. The application of OH1, OH2 and OH3 headers are shown in Figure 5.2 and Figure 5.3.

5.3.1. Optimized Header Scheme for OH_1 Compressor

The OH_1 compressor is used to compress RTP/UDP/IP protocols. The IR header and the DYN header for the OH_1 compressor are shown in Appendix C. The OH1 header for the OH_1 compressor is the same as RoHC compressed header shown in Chapter 4.

5.3.2. Optimized Header Scheme for OH_2 Compressor

The OH_2 compressor is used to compress SDAP/PDCP protocol headers. The IR header for the OH_2 is shown in Table 5.4. The detail is shown as follows:

Table 5.4: Structure of the IR Header for OH_2 Compressor

Octet							
8	7	6	5	4	3	2	1
CID							
1	1	1	1	1	1	0	1
Profile = SDAP							
8-bit CRC							
SDAP							
PDCP_1							
PDCP_2							

- 8-bit CID: A small CID is applied for OH_2 IR packet header.
- 1111,1101: It indicates the type of this header is IR header.
- Profile: This field states the protocol used here is SDAP.
- SDAP: The whole SDAP protocol is STATIC. This field is then the static chain.
- PDCP: Two Octets of PDCP. The first four bits are STATIC, and the rest bits are CHANGING. For simplicity, the overall PDCP is viewed as the DYNAMIC chain.

The DYN header for the OH_2 is shown in Table 5.5. There are two main differences between the IR header and the DYN header. Firstly, the DYN header removes the STATIC field (SDAP) since it does not transmit STATIC information. Secondly, The second octet in the DYN header is different, which is 1111,1000 to indicate the DYN header type.

Table 5.5: Structure of the DYN Header for OH_2 Compressor

Octet							
8	7	6	5	4	3	2	1
CID							
1	1	1	1	1	0	0	0
Profile = SDAP							
8-bit CRC							
PDCP_1							
PDCP_2							

The OH2 header is applied to replace the SDAP/PDCP headers, shown in Table 5.6. The sequence number in PDCP is set to 4 bits after a zero indicator. The CRC is used to check the SN_PDCP.

Table 5.6: Structure of the OH2 Header for OH_2 Compressor

Octet							
8	7	6	5	4	3	2	1
0	SN_PDCP				CRC		

5.3.3. Optimized Header Scheme for OH_3 Compressor

The OH_3 compressor is used to compress GTP-U/UDP/IP protocols.

The IR header for the OH_3 is shown in Table 5.7. The profile is set as GTP-U. There are 12 bytes STATIC chains, as described in Table 5.1. There are 6 bytes CHANGING chains, as described in Table 5.1. The overall length for GTP-U IR is 18 bytes.

Table 5.7: Structure of the IR Header for OH_3 Compressor

Octet							
8	7	6	5	4	3	2	1
CID							
1	1	1	1	1	1	0	1
Profile = GTP-U							
8-bit CRC							
12-byte STATIC							
6-byte CHANGING							

The DYN header for the OH_3 is shown in Table 5.8. The STATIC chains are removed, compared with the IR header. The packet type indicator is changed as well.

Table 5.8: Structure of the DYN Header for OH_3 Compressor

Octet							
8	7	6	5	4	3	2	1
CID							
1	1	1	1	1	0	0	0
Profile = GTP-U							
8-bit CRC							
6-byte CHANGING							

The OH3 header is shown in Table 5.9, which is used to replace the header of IP/UDP/GTP-U. The SN_Created field is developed randomly to indicate the order of the packet since there is no SN in either IP or UDP header. Compared with the OH1 header, there are new fields for the GTP-U header. Four 4-bit fields are added: SN_GTP-U, N-PUD Number, SN_UL_QFI, and CRC_GTP. The first three fields are created using the 4-bit LSB algorithm. The 4-bit CRC_GTP shares the same polynomial as the 3-bit CRC check.

Table 5.9: Structure of the OH3 Header for OH_3 Compressor

Octet							
8	7	6	5	4	3	2	1
0	SN_Created				CRC		
Checksum_1							
Checksum_2							
SN_GTP-U				N-PDU Num			
SN_UL_QFI				CRC_GTP			

In the overall OH packet through the F1-U reference point, the OH1 header will be in the innermost layer, the OH2 header in the middle layer, and the OH3 header in the outermost layer. In the overall OH packet through the N3 reference point, only the OH1 header and the OH3 header are applied.

In the proposed optimized header compression algorithm, the OH_1 compressor and decompressor working mechanism is inherited from the RoHC algorithm. The compressor and the decompressor working mechanisms of the OH_2 and the OH_3 algorithm are similar to the OH_1 algorithm. A detailed explanation of the working mechanism is shown in Chapter 5.4.

5.3.4. Optimized Header Scheme for MOH Algorithm

In this scenario, two QoS flows are created and transmitted in the same PDU session. One is for voice-over IP service; the other is for video-over IP service. Based on the design mentioned above, OH1 header, OH2 header, and OH3 header should be added over the payload for two QoS flows. However, there is still room for optimization since they are applied in the same PDU session.

Only the OH1 and the OH2 headers will be added to the payload for a video-over IP packet since the OH3 header will be the same as the synchronous voice-over IP packet. The OH3 header replaces the GTP-U/UDP/IP headers in the protocol stack. The OH3 header and the GTP-U/UDP/IP headers should be largely the same for the data flow in the same PDU session. Even if part of the OH3 header is not totally the same between two QoS flows like QFI_SN, there are strong links between them, which can be inferred from one to another because of the timestamp synchronization. In this case, the video packet will be sent faster, leaving more resources for handling the voice packet. The size of the header for video-over IP will decrease by 56% from 9 bytes to 4 bytes. However, the overall packet is not apparently compressed since the header size is tiny enough. The optimization is toward the response time of execution. The illustration of this optimization is shown in Figure 5.4.

Before the MOH algorithm optimization, the OH algorithm should be completely implemented twice for two QoS flows in the same PDU session. After the MOH algorithm optimization, only the OH_1 and the OH_2 parts of the OH algorithm will be implemented for the video QoS flow. The optimization results will be revealed by CPU execution time, leading to lower power consumption and processing latency. In most cases, the video packet is much larger than the audio packet, causing extra latency for the video packet. The multi-stream optimization aims to decrease the OH algorithm procedures for the video packets, releasing the extra latency in the video service.

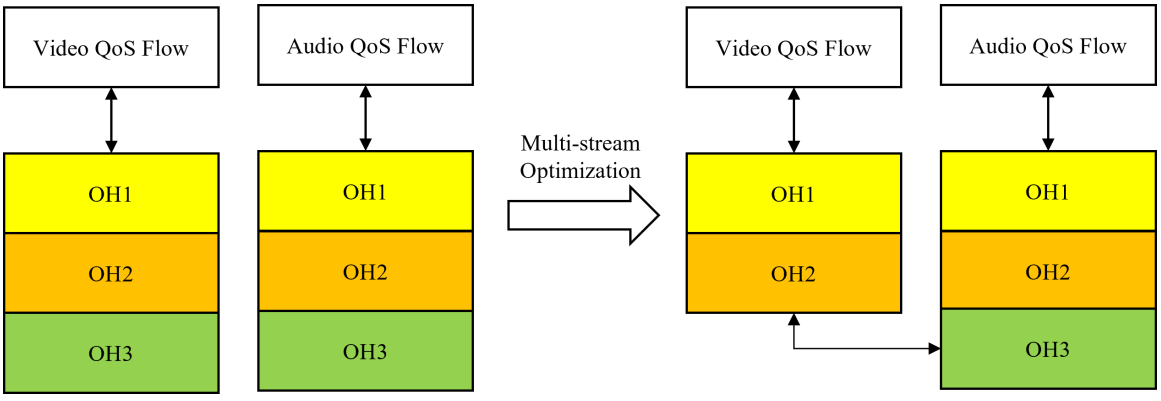


Figure 5.4: Multi-stream Optimization for QoS Flows

5.3.5. Optimized Header Scheme for VOH Algorithm

If the three types of headers are considered in a coordinated manner, it's feasible to compress them further in a vertical manner. The structure of the VOH algorithm is shown in Figure 5.5.

In the OH1 header, it's not necessary to handle the UDP checksum since the UDP checksum is already implemented in the outer header. In the OH3, it's not required to operate the SN_Created since it's developed for the IP or the IP/UDP packets when the SN order is absent. The sequence number in the GTP-U will replace the function of the SN-Created. Moreover, the redundancy of the CRC check can also be optimized. In this case, a more compact header scheme is shown in Table 5.10. The VOH header (from the VOH algorithm) is applied to replace IP/UDP/GTP-U/PDCP/SDAP/IP/UDP/RTP header components. The VOH algorithm will compress the IP/UDP/GTP-U/PDCP/SDAP/IP/UDP/RTP headers at the same time for lower latency and lower compressed header. The VOH algorithm uses the same working mechanism as the OH_1, OH_2 and OH_3 pairs.

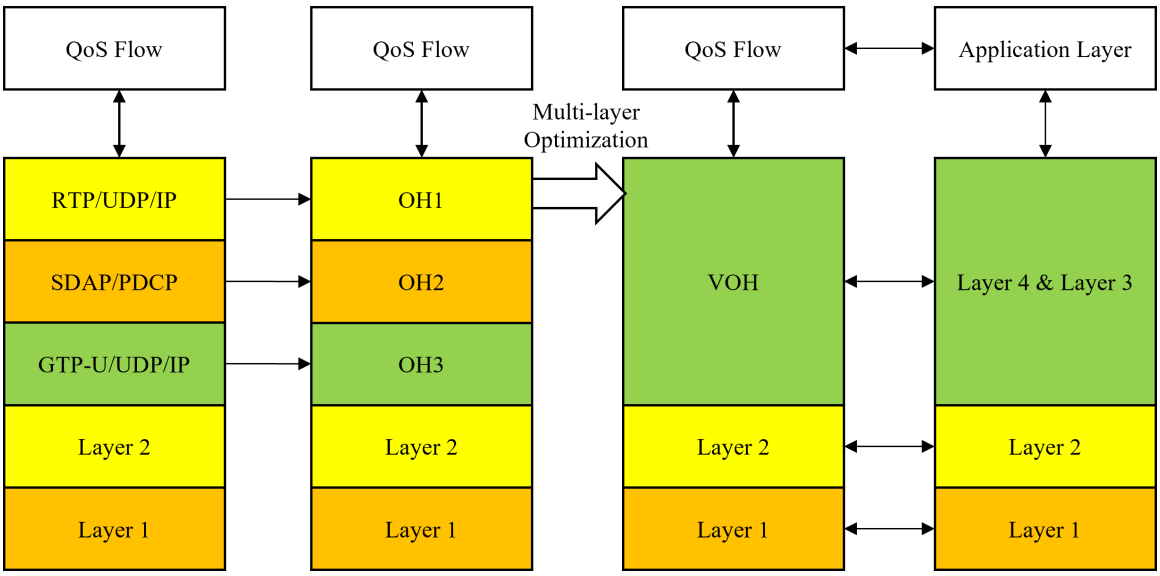


Figure 5.5: VOH Algorithm for QoS Flows

The N-PDU Number becomes three bits instead of 4 bits. It does not influence the accuracy since the original N-PDU Number is only 8 bits. The 4-bit CRC is developed and calculated based on the same polynomial as the 3-bit CRC. This CRC will calculate the overall header except for the UDP checksum. The UDP checksum calculates the outer UDP protocol. In the N3 reference point simulation, the SN_PDCP will be zero since there is no PDCP over the N3 simulation.

There are various advantages to multi-layer optimization vertically. Firstly, a higher header compression rate is achieved in this optimization from 9 bytes to 6 bytes. Secondly, the response time is much lower in this optimization because of the simplicity of the procedures. Thirdly, the algorithm robustness in this optimization is better than in the multi-stream OH optimization since the multi-layer OH algorithm is more integrated.

Table 5.10: Structure of the VOH Header for Compact Compression

Octet							
8	7	6	5	4	3	2	1
0	SN_RTP				N-PDU Num		
SN_GTP-U					SN_PDCP		
SN_UL_QFI					CRC		
UDP Checksum_1							
UDP Checksum_2							

5.4. Working Mechanism of Optimized Header Compression Algorithm

5.4.1. Mechanism of OH Compressor

Since the working mechanisms for the OH_1, OH_2, OH_3, and VOH pairs are mostly the same, the working mechanism for compression and decompression will be discussed simultaneously. The flowchart of the optimized header compression algorithm-based compressor is shown in Figure 5.6. The implementation of the OH algorithm compressor and the decompressor is shown in Figure 5.2. The working flow is described as follows:

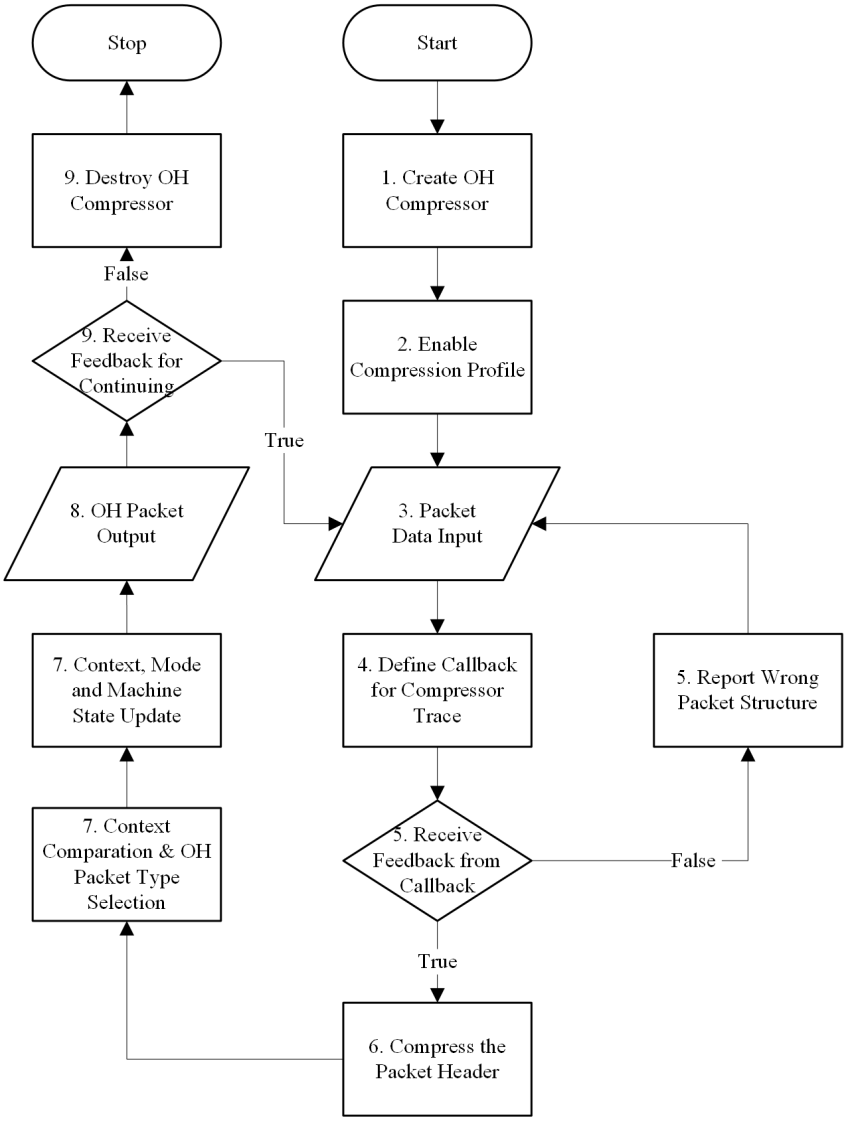


Figure 5.6: Flowchart of the OH Algorithm Compressor

1. The OH algorithm will be initialized to create a default OH compressor for the following compressing actions.
2. The Profile is selected and allocated based on different protocol layers for the initialization of the compressor. Profile 0x01 for OH_1, profile 0x02 for OH_2, profile 0x03 for OH_3, profile 0x04 for OH_4.
3. The headers and the payload (application layer) will be read into the buffer of the compressor. The Application Interface (API) will be defined to ensure the data reading.
4. Define a trace callback for scanning the data packet. The procedure guarantees the integrity of packets, ensuring the data packet is suitable to the applied profile.
5. Receive the return boolean value from the trace callback. The procedure will go to the next step if positive feedback is received. Otherwise, the program will report the wrong status first and return to the third phase to recollect data packets. There are some reasons for obtaining

a false, like the incomplete data packet, the wrong checksum, the incorrect length value, or even a mismatch between the packet and the profile.

6. In this step, the redundant headers will be stored in the compressor buffer first, apart from the payload. Each header will be compressed simultaneously to obtain the optimized headers. The approaches to shorten the header will be different for various categories of fields:

   - STATIC-KNOWN: This header part will be compressed directly without considering anything since it's already defined before the compression.
   - STATIC: This part of the header must be transmitted at least once, even if it's stable for the whole stream. In this case, a counter for the compressor is required to remember the number of sent packets. In the actual implementation of the algorithm, at least three times of IR packet transmission is required to initialize the static field of the compressor for stabilization. After the initialization phase, comparing the STATIC fields in the context of the compressor and the STATIC field of headers is compulsory before compressing in case of changes of STATIC fields in the headers. If a mismatch happens between the headers' compressor and the STATIC fields, the counter will be set to 0. Then the IR packets should be sent at least three times for static field initialization.
   - STATIC-DEF: This part of the header will be treated the same as STATIC. The difference between them is the function. STATIC-DEF field is used to define the communication tunnel between two entities.
   - INFERRED: This header part will be calculated in the decompressor part when decompressing the OH packets. This field type includes the checksum and the header length in various protocols. The INFERRED field in the header is checked in trace feedback scanning. The packet with the wrong INFERRED field will be abandoned before compressing.
   - CHANGING: This header part will change with the data stream transmitting, like timestamp or sequence number. Luckily, these fields of headers are not unpredictable. Take the timestamp in the RTP protocol as an example. The timestamp in the first transmitted packet is unpredictable, but the later timestamp is usually predictable. The timestamp is used to identify the packet sending time, created one by one. In the compressor, it's feasible to define a function to calculate the time stride of the timestamp field. In other words, the time stride represents the difference in timestamps between two successive packets. The value of the stride would be one or larger, but it should be stable for a while. In this case, the timestamp of the next RTP is predictable by adding the time stride to the timestamp of the last RTP. In the compressor part, the predicted timestamp will be calculated to compare with that in the next RTP packet. If the comparison feedback is true, the timestamp field in the RTP will be compressed.

     Consequently, a stable approach to compress the CHANGING field is created. In implementing CHANGING field compression, the CHANGING field will be updated thrice in the IR packet. Then a tiny packet called the DYNAMIC packet will be created to update the CHANGING field, specially without sending the STATIC field. This procedure is to calculate a stable stride without wasting too many bytes. The DYNAMIC packet

is usually sent three times as well. Moreover, it's possible to find a mismatch between the actual CHANGING field and the predicted CHANGING field. In this case, an update based on the DYNAMIC packet is required to recalculate the stride.

7. In this phase, the comparison between the fields in the headers and the fields in the compressor context will be implemented to verify the possibility of compression. If the mismatch happens in the CHANGING type of fields, a DYNAMIC packet will be created and sent to update the decompressor's context fields. If the CHANGING fields mismatch happens again, even if they are updated, the IR packet must update the whole decompressor and compressor fields. When a mismatch occurs in the STATIC type of fields, the IR packet will be created to update the compressor and the decompressor. Suppose the mismatch still happens even if the IR packet has been sent several times. In that case, the pair of OH algorithms should be terminated, which means the current packet sending is not suitable for compressing by the OH algorithm.
8. The optimized headers based on the OH algorithm will be combined with the payload to build an OH packet. The OH packet will be delivered to the following entity over the reference point. The corresponding decompressor will decompress the OH packet to obtain the original data packet.
9. The phase will return to packet data input if a callback of continuing is heard. Otherwise, the procedure will stop, and the compressor will be terminated.

Before sending the OH packets, it's recommended to send several original data packets to build a solid connection between two entities over the reference point.

5.4.2. Mechanism of OH Decompressor

The OH decompressor is applied to decompress the OH packet created by the OH compressor. The decompressed data packet from the decompressor is used to verify the correctness of the destination. For example, some comparisons are done to confirm whether the destination port in the decompressed packet is the same as the port number of this entity. If the comparison is passed, the obtained packet will be forwarded to the following entity for the PDU applications. Figure 5.7 shows the flowchart of the OH algorithm's decompressor. The implementation of this decompressor is shown in Figure 5.2.

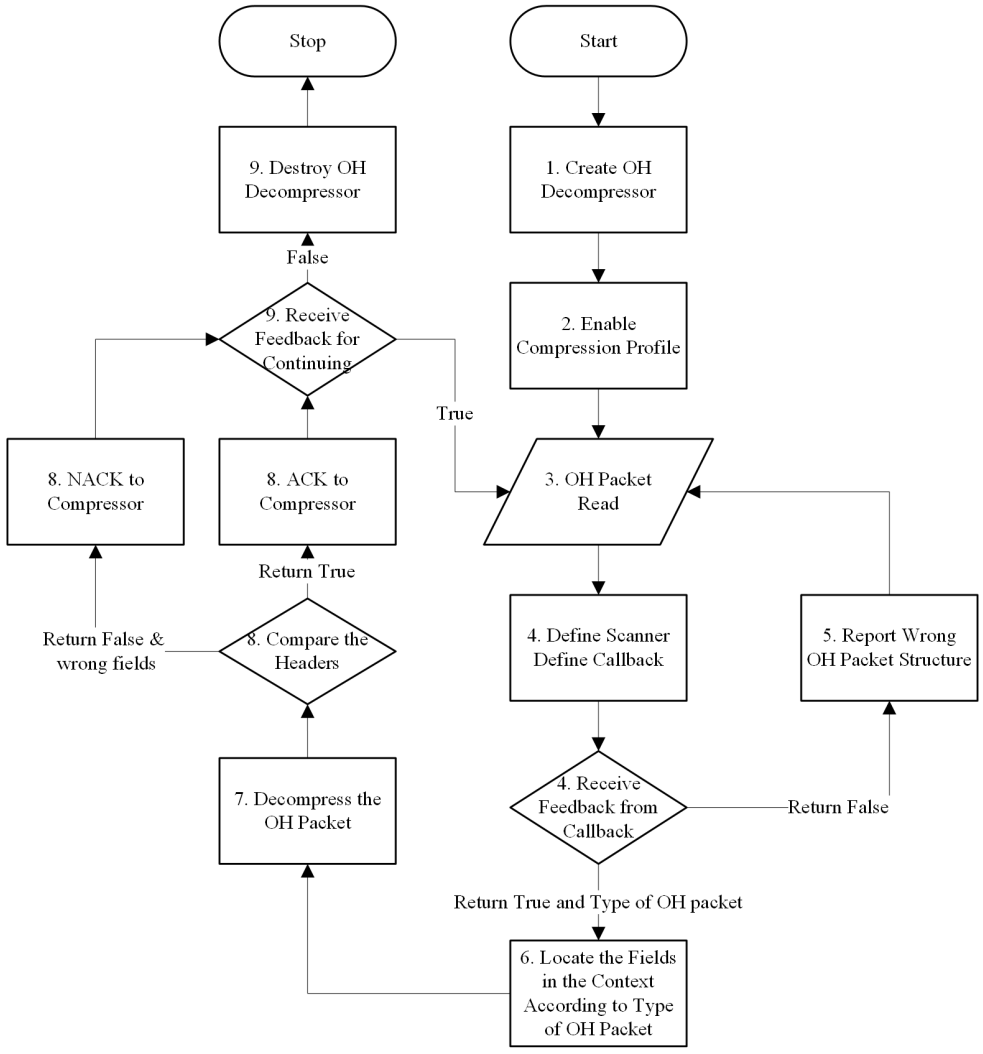


Figure 5.7: Flowchart of the OH Algorithm Decompressor

The working flow of the decompressor is described as follows:

1. The OH decompressor is created to decompress the OH packet from the compressor.
2. The decompressor is configured by the profile, which is the same as the compressor.
3. The received OH packet is stored in the decompressor buffer and application layer. These OH packets will be read respectively by the decompressor.
4. Define a scanner function to scan the whole OH packet to check the integrity and recognize the type of OH packet. A callback function is developed to return the boolean value and type of OH packets.
5. If the received packet type fails to be recognized, the callback function returns false. The decompressor algorithm will report the wrong OH packet structure and return to the third phase. If the recognization succeeds, the callback function will return a positive result and the type of OH packet.

6. In this phase, the fields in the context will be extracted according to the type of OH packets. For the original packet, nothing will be extracted. For the IR packet, the STATIC-KNOWN field will be read. All the fields except the CHANGING field will be extracted for the DYNAMIC packet. All the fields will be extracted for the OH packet (When the compression is optimal).
7. The decompressor will decompress the OH packet in this phase. The fields extracted from the context will be applied to make up the STATIC fields in the headers. The CHANGING and INFERRED fields will be calculated using the data in the context.
8. The decompressed header by the decompressor will be used to compare with the information stored in the entity. For instance, the destination port of the IPv4 will be used to compare with the port number of the destination entity. If all the data matches at the receiver, the decompressor will send positive feedback called ACK to the compressor to notify a successful transmission. Otherwise, the negative feedback, called NACK, will be sent to the compressor to indicate the types of fields that occur the error. There are two types of NACK to show the STATIC error or the DYNAMIC error.

   In most cases, the decompressor will only send the STATIC error since the DYNAMIC error is too hard to detect. The compressor will choose the type of OH packets corresponding to the feedback from the decompressor. It's important to note that the OH and feedback packets are in two parallel channels. In implementing optimized header compression, two tunnels will be created between two entities. There will be a compressor belonging to OH_A and a decompressor belonging to OH_B on the same side. The sending packet can hold the feedback packet of the decompressor and the OH packet of the compressor at the same time for transmission efficiency.
9. The decompressor will receive feedback for continuing work no matter whether the last OH packet is successfully decompressed or not. If the feedback is true, the decompressor will continue to read the OH packet in the buffer. Otherwise, the decompressor will be destroyed, and the compression algorithm will be terminated.

5.5. Evaluation Metrics

This thesis conducts the performance evaluation for the OH algorithm, RoHC algorithm, and uncompressed communication. The first approach is the compression rate for headers and packets, which calculates the percentage of compressed data when using the OH and RoHC algorithms. The second approach is the overall throughput in the same packet-sending rate, which shows transmission efficiency using the OH and the RoHC algorithms. The third approach is to evaluate the algorithm's complexity.

5.5.1. Metric for Compression Rate

An idealistic upper bound on the potential network resource savings can be derived by assuming that the header is totally compressed (the size of the header is zero) for each created packet (i.e.

IPv4 packet or UDP packet) [55]:

$$S_{max}(i) = 1 - \frac{P_i}{UH + P_i} = \frac{UH}{UH + P_i} \tag{5.1}$$

where UH denotes the size of the uncompressed protocol headers, i.e., the protocol encapsulation overhead, P_i denotes the payload data size of the i-th packet and $S_{max}(i)$ denotes the potential compression rate for this case. The average potential compression rate for an N-packet sequence or session is calculated as [55]:

$$\bar{S_{max}} = \frac{1}{N} \sum_{i=1}^{N} S_{max}(i) \tag{5.2}$$

For a more plausible scenario, the packet i's compressed header size, $CH_i \geq 0$. In this instance, we derive the real compression rate (or alternatively the gain) of the encapsulation overhead (headers) as (i) [55]:

$$S_H(i) = \frac{UH - CH_i}{UH} \tag{5.3}$$

where $S_H(i)$ denotes the header size compression rate.

(ii) the actual saving for individual packets as [55]:

$$S_P(i) = \frac{UH - CH_i}{UH + P_i} \tag{5.4}$$

where $S_P(i)$ denotes the packet size compression rate.

5.5.2. Metric for Throughput

In network communication, throughput describes the volume of data that can be sent across a network at a specific time. Bits per second (b/s), kilobits per second (kb/s), or megabits per second (Mb/s) are the most common units of measurement.

Numerous elements, such as bandwidth, packet size, and network congestion, impact throughput. While packet size refers to the amount of data contained in each packet, throughput refers to the maximum amount of data transmitted in a given time. The throughput of packets (including the header and the payload) per cell can be calculated to compare the performance of different algorithms.

5.5.3. Metric for Algorithm Execution Latency

The execution of the algorithm will cause extra latency attributed to the function implementation and data loading in the algorithm, which is called algorithm execution latency. This latency will be more apparent with more data executed per second since the execution of the algorithm is serial, which will cause latency for the communication. Therefore, it's also necessary to focus on the latency from the algorithm execution in optimizing the algorithm.

Besides, the algorithm execution time will be related to the power consumption. Higher execution latency means more computing resources are used for this round of calculation. Consequently, the decrease in execution latency will bring about less power consumption [56].

The algorithm execution latency is calculated using CPU time-stamping by comparing the time the algorithm finishes compressing (or decompressing) with the time the algorithm starts to compress (decompress).

5.6. Implementation of OH Compression Algorithm

The open source RoHC library [29] is learned in this thesis for building up the basics of the code.

Matlab analyzes simulation results [57]. MathWorks created the programming language and environment known as MATLAB. It is frequently used for data analysis, visualization, and algorithm development in engineering, science, and mathematics. The interactive atmosphere and high-level programming language for numerical calculation, data analysis, and visualization are provided by MATLAB. MATLAB allows users to perform matrix operations, create and manipulate arrays, and perform mathematical operations. It also includes built-in functions for various mathematical operations, such as signal processing, image processing, and optimization.

5.6.1. Illustration of IP Packets Compression

An example of the inner or outer IP packet compression using the OH algorithm is shown in Figure 5.8a. The packet sending rate is set to 2 packets/second for testing. The payload is 8 bytes. The original IPv4 packet is 28 bytes. The four working phases are marked in Figure 5.8a. In phase one, the OH algorithm sends the original IP packet to initialize the connection between the sender and the receiver. In phase two, the IR packet is applied to fill in the contexts in the compressor and the decompressor to initialize the OH algorithm. In phase three, the DYNAMIC packet sends the DYNAMIC fields only for stabilization and double check. In phase four, a stable OH connection is built between the compressor and the decompressor. The OH packet of 9 bytes is sent over the connection. The OH header is only one byte for SN and CRC checks.

In phase one, the size of the sending packet (28 bytes) is the same as the size of the IPv4 packet

since the OH algorithm needs to send at least three original packets before initialization. In phase two, the size of the sending packet becomes lower because the IR packet is applied. The 26-byte IR packet includes all the 10-byte STATIC and the 4-byte DYNAMIC fields of the IPv4 header. The header of the IR packet is 18 bytes. The CRC check is also included in the IR packet. At least three IR packets are required to finish the STATIC field initialization. In phase three, the sending packet, which removes STATIC fields, is applied to build the CHANGING field even if these CHANGING fields are assigned in the IR packet. Therefore, the size of the sending packets dramatically decreases since the STATIC fields are not sent (8-byte headers + 8-byte payload). It's necessary to resend the value of the CHANGING fields again in the DYNAMIC packet for robustness. In phase four, the OH packet is created and sent. The header of the OH packet is only one byte. This OH packet header includes a one-bit indicator, four bits of fake SN, and three bits of CRC.

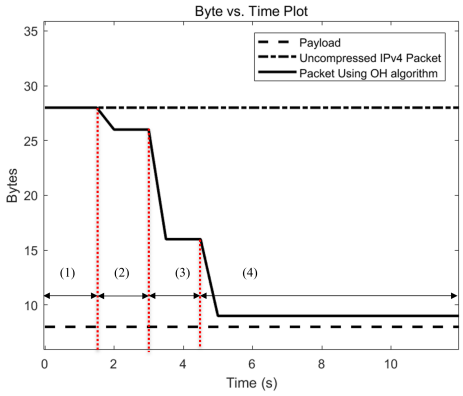
5.6.2. Illustration of UDP Packets Compression

An example of the inner or outer IP/UDP packet compression using the OH algorithm is shown in Figure 5.8b. The packet-sending rate is two packets per second for OH compression testing. The payload of this test is still 8 bytes. The original UDP packet is 20 + 8 + 8 = 36 bytes. The four working phases are marked in Figure 5.8b. In phase one, the OH algorithm compressor sends the uncompressed OH packets to build connections between two entities. The least number of uncompressed packets should be three. In phase two, the size of the sending packet (IR packet, 24-byte header + 8-byte payload) changes, which is a little smaller than the size of the uncompressed packet. The IR packet is created to build the STATIC and CHANGING fields in the compressor and the decompressor contexts. In phase three, the size of the sending packet is instantly decreasing for transmitting the DYNAMIC packets (10-byte header + 8-byte payload). In phase four, the size of the OH packet is finally stable, which is used to send the OH packet. The OH packet in this phase is three bytes, including one byte for IP and two bytes for UDP checksum.

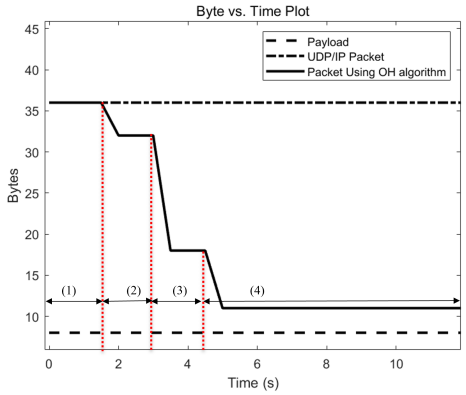
There are several differences between Figure 5.8a and Figure 5.8b. Firstly, the size of the uncompressed UDP packet is 8 bytes larger than the size of the uncompressed IPv4 packet since the UDP header is always over the IP header. Besides, the size of the OH packet for UDP is two bytes larger than that of the IPv4 packet.

5.6.3. Illustration of RTP Packets Compression

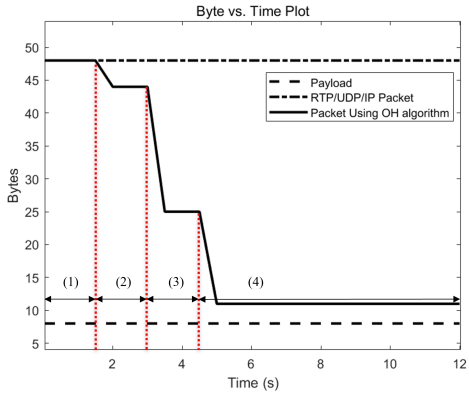
An example of the IP/UDP/RTP packet compressing based on the OH algorithm is illustrated in Figure 5.8c.



(a) OH Compression for IPv4 Packets



(b) OH Compression for UDP Packets



(c) OH Compression for RTP packets

Figure 5.8: Working Flow of OH Compression Algorithm for IPv4, UDP, IP packets, respectively. Phase (1) denotes the uncompressed packet before using the compression algorithm; phase (2) denotes the IR state of the compressor state machine; phase (3) denotes the FO state of the compressor state machine; phase (4) denotes the SO state of the compressor state machine

The packet sending rate is allocated to 2 packets/second. The payload is 8 bytes as well. The uncompressed RTP packet is 48 bytes, calculated by 20 + 8 + 12 + 8 since the RTP header is 12 bytes. The four working phases are marked in Figure 5.8c. The uncompressed packet is sent to connect the two entities in phase one. The RTP IR packet is created and transmitted through the N3 reference point in phase two. In phase three, the size of the OH packet for the RTP packet is instantly dropping. In phase four, the OH packet sends three bytes. The fields in the RTP protocol are either static or predictable because of real-time transmission. The RTP SN is stored in the first byte of OH packet. The overall procedure is the same as mentioned in IPv4 and UDP compression.

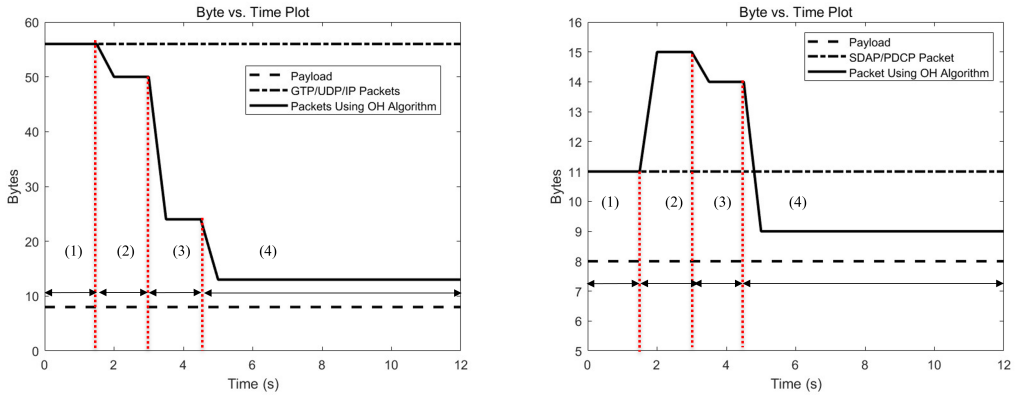
There are a few differences compared with the IP and the UDP compression. The size of the uncompressed packet is 12 bytes larger than that of the UDP packet since the RTP header is 12 bytes. Besides, the size of the OH packet for RTP is the same as UDP since the sequence number of RTP will be stored in the first byte of the OH packet. There will be no extra space required.

5.6.4. Illustration of GTP-U Headers Compression

An example of compressing GTP-U/UDP/IP headers using the OH algorithm in the OH_3 compressor is illustrated in Figure 5.9a. The original size of the header is 20 + 8 + 20 = 48 bytes. The payload is set 0 in this simulation. In phase one, the algorithm transmits the uncompressed header. In phase two, the 42-byte IR header for GTP-U/UDP/IP is transmitted to construct the context in the compressor. Due to many INFERRED and STATIC-KNOWN fields appearing in UDP and IP headers, the IR header size is smaller than the uncompressed header size. In phase three, the 16-byte DYN header is sent to solidify the CHANGING field in the context. The 5-byte OH header for GTP-U/UDP/IP is sent in phase four.

5.6.5. Illustration of SDAP/PDCP Headers Compression

An example of compressing SDAP/PDCP headers using the OH algorithm in the OH_2 compressor is shown in Figure 5.9b. The original size of the header is 3 bytes. The payload is set 0 in this simulation. In phase one, the algorithm sends the uncompressed header. The four working phases are marked in Figure 5.9b. In phase two, the IR header for SDAP is sent to build up the context, which is 7 bytes. In phase three, the 6-byte DYN header is transmitted to verify the CHANGING field in the context. In phase four, the 1-byte header for SDAP is sent, which means the OH algorithm is working properly.



(a) OH Compression for GTP-U/UDP/IP Headers

(b) OH Compression for SDAP/PDCP Headers

Figure 5.9: Working Flow of OH Compression Algorithm for GTP and SDAP Packets

5.7. Upper Bound Calculation of OH Compression Results

Before the real-time simulation for the OH algorithm and 5G networks, it's preferred to calculate the upper bound results based on the proposed design without considering outside interference.

5.7.1. OH Compression for VoIP Payload Packets

The upper bound calculation of the OH algorithm over the N3 reference point is shown in Table 5.11.

Firstly, the payload type of Voice Over IP is considered. The most commonly used codecs for VoIP are Opus [58], G.711, and G.729. G.711 uses a sampling rate of 8 kHz and a bit rate of 64 kbps, resulting in a payload size of 20 to 160 bytes. G.729, on the other hand, uses a sampling rate of 8 kHz and a bit rate of 8 kbps, resulting in a payload size of 20 bytes. Opus is an open coding algorithm for various payload sizes. Therefore, the average payload of VoIP can range from 20 bytes per packet for G.729 to 160 bytes per packet for G.711, depending on the codec being used. In the calculation shown in Table 5.11, the average size of the payload is set to 50 bytes coded by Opus. The sample time is 10ms, the bit rate is 32kb/s, and the minimum payload size equals 10x32/8 = 40 bytes. For stability, the payload size is viewed as 50 bytes since the Opus-coded payload size may change over time.

The header and packet compression rate is calculated by comparing with the uncompressed and RoHC header and packets. The comparison between the RoHC header and the uncompressed header is not included since the research is mainly about the OH algorithm header compression. The header compression rate for OH vs. RoHC is calculated by the header compression rate for OH vs. uncompressed subtracted from the header compression rate for RoHC vs. uncompressed.

Table 5.11: Calculation of OH Improvement when Using VoIP Payload at N3 Reference Point

Protocol	Uncompressed Header Byte	RoHC Header Byte	OH Header Byte	Compression Rate OH vs Uncompressed	Compression Rate OH vs RoHC
Outer IPv4	20	20	1	95%	95%
Outer UDP	8	8	2	75%	75%
GTP and Extension	20	20	2	90%	90%
Inner IPv4	20	20	1	95%	95%
Inner UDP	8	8	2	75%	75%
RTP	12	12	0	100%	100%
Sum	88	88	8	90.9%	90.9%
Sum With Payload	138	138	58	58%	58%

This calculation indicates the transmission over the N3 reference point since PDCP and SDAP protocols are not included. In this case, RoHC will be disabled since the current RoHC is installed inside the PDCP layer. Therefore, the RoHC algorithm does not work in this case. A detailed analysis of this calculation is as follows:

1. Outer IPv4: The outer IPv4 is located over the outer UDP message in the 5G communication. The uncompressed header of IPv4 is 20 bytes. The RoHC bytes are unchanged since no RoHC over the N3 reference point. The OH header is one byte, which covers the CRC check.

Consequently, the OH header removes 95% of bytes compared with the uncompressed header.

2. Outer UDP: The outer UDP is located over the GTP-U message in the GRPS tunnel. The uncompressed header of UDP is 8 bytes. The RoHC is disabled as well. The OH header is 2 bytes. Therefore, the OH header deletes 75% of the UDP header.
3. GTP-U and GTP extension: The uncompressed GTP-U message is 12 bytes, and the extension header for the PDU container is 8 bytes. The RoHC is unable to compress this type of header. After compression, the OH header of this type of header is one byte, which includes the CRC check. In this case, the OH algorithm saves 90% space compared with the uncompressed and the RoHC algorithm.
4. Inner IPv4: The inner IPv4 header is applied to build the connection between the UPF and the DN. The RoHC algorithm over the F1-U reference point can compress the Inner IPv4. However, the RoHC is terminated after the gNB-CU entity. Therefore, the Inner IPv4 is still uncompressed over the N3 reference point in the former algorithm. Using the OH algorithm, this IPv4 header is compressed to one byte, which removes 95% of redundant bytes.
5. Inner UDP: The inner UDP builds the response mechanism between the UPF and the DN. This protocol is uncompressed for the reason mentioned above in the inner IPv4. The OH header after compression is 2 bytes, including the UDP checksum. Consequently, 75% of the redundant bytes are removed.
6. RTP: The RTP protocol in this place will provide real-time service for the DN since the timestamp is included in this protocol header. It's uncompressed over the N3 reference point in the 5G communication networks. The OH header for RTP is zero byte. In this case, 100% of the redundant bytes are deleted for compression.
7. Payload: This calculation example defines the VoIP payload as 50 bytes for simplicity. This part of the data cannot decompress by the OH algorithm since the OH algorithm aims to compress the headers of the packets instead of the payload. Other algorithms may be applied to compress this part of the packet.
8. Sum: There will be 138 bytes for the uncompressed and the RoHC packets. The size of the OH packet will be 56 bytes. There will be 90.9% of redundant headers removed. In total, 58% of the original packets are removed to build up the OH packet, which leads to a great improvement in efficiency and latency.

The upper bound calculation of the OH algorithm over the F1-U reference point is shown in Table 5.12.

Table 5.12: Calculation of OH Improvement When Using VoIP Payload at F1-U Reference Point

Protocol	Uncompressed Header Byte	RoHC Header Byte	OH Header Byte	Compression Rate OH vs Uncompressed	Compression Rate OH vs RoHC
Outer IPv4	20	20	1	95%	95%
Outer UDP	8	8	2	75%	75%
GTP and Extension	20	20	2	90%	90%
PDCP/SDAP	3	3	1	66.7%	66.7%
Inner IPv4	20	1	1	95%	0 %
Inner UDP	8	2	2	75%	0 %
RTP	12	0	0	100%	0 %
Sum	91	54	9	90.1%	49.5%
Sum With Payload	141	104	59	58.2%	31.9%

1. Outer IPv4: See Table 5.11.
2. Outer UDP: See Table 5.11.
3. GTP-U and GTP extension: See Table 5.11.
4. PDCP/SDAP: The layers of PDCP/SDAP are applied to build the radio bearer, provide the RoHC algorithm, etc. The headers are three bytes. RoHC cannot compress them. The OH header for these headers is one byte, including the CRC check. The improvement is 66.7% fewer contents compared with the uncompressed and the RoHC packets.
5. Inner IPv4: The IPv4 can be compressed by the RoHC since the PDCP layer provides the function. In this case, there will be no improvement by the OH algorithm.
6. Inner UDP: The same result mentioned above about the inner IPv4.
7. RTP: The same result mentioned above about the inner IPv4.
8. Payload: See Table 5.11.
9. Sum: To conclude, there will be 141 bytes for the uncompressed packet overall. The RoHC will be 104 bytes. The OH packet will be 59 bytes. The header compression rate over the uncompressed header is 90.1%. The header compression rate over the RoHC algorithm is 49.5%. For a package, 58.2% of the content will be removed compared with the uncompressed packet. Moreover, 31.9% of the packet content will be saved compared with the RoHC packet.

5.7.2. OH Compression for Video over IP Payload Packets

It's possible to consider video service over IP since it's an extension service of VoIP.

Commonly used video codecs for Video over IP include H.264, H.265, VP8, and VP9. These codecs use different compression techniques to encode the video signal, resulting in different payload sizes. The payload size can also vary depending on the video resolution and frame rate. Higher resolution and frame rates require more data and, therefore, larger payload sizes. The

mobile device-based video over IP service does not pursue the image's high resolution because of the expense and the latency.

Assuming a video resolution of 360p (640x360 pixels), a frame rate of 30 frames per second, and a packetization interval of 20 ms, the size of the payload in a packet when using H.264/AVC for video over IP can be approximately 166-333 bytes if each frame of video would require approximately 5-10 KB of data. For a specific application case, the average payload size is 300 bytes.

The upper bound calculation result about the transmission over the N3 reference point is shown in Table 5.13.

Table 5.13: Calculation of OH Improvement When Using Video over IP Payload at N3 Reference Point

Protocol	Uncompressed Header Byte	RoHC Header Byte	OH Header Byte	Compression Rate OH vs Uncompressed	Compression Rate OH vs RoHC
Outer IPv4	20	20	1	95%	95%
Outer UDP	8	8	2	75%	75%
GTP and Extension	20	20	2	90%	90%
Inner IPv4	20	20	1	95%	95%
Inner UDP	8	8	2	75%	75%
RTP	12	12	0	100%	100%
Sum	88	88	8	90.9%	90.9%
Sum With Payload	388	388	308	20.6%	20.6%

The size of the uncompressed packet is 388 bytes. The size of the OH packet is 308 bytes. The packet compression rate over the uncompressed and RoHC packets is 20.6%.

The upper bound calculation result about the transmission over the F1-U reference point is shown in Table 5.14.

Table 5.14: Calculation of OH Improvement When Using Video over IP Payload at F1-U Reference Point

Protocol	Uncompressed Header Byte	RoHC Header Byte	OH Header Byte	Compression Rate OH vs Uncompressed	Compression Rate OH vs RoHC
Outer IPv4	20	20	1	95%	95%
Outer UDP	8	8	2	75%	75%
GTP and Extension	20	20	2	90%	90%
PDCP/SDAP	3	3	1	66.7%	66.7%
Inner IPv4	20	1	1	95%	0 %
Inner UDP	8	2	2	75%	0 %
RTP	12	0	0	100%	0 %
Sum	91	54	9	90.1%	49.5%
Sum With Payload	391	354	309	21%	11.5%

The overall results in both N3 and F1-U reference points for both voice-over-IP and video-over-IP

show that the performance of the OH compression algorithm is the best in the N3 reference point for voice-over-IP service. The first reason is that the original RoHC algorithm is disabled in the N3 reference point. Besides, the size of the payload for voice-over-IP service is much smaller. In this case, the performance of the OH algorithm is much more excellent, which achieves a 90.9% header compression rate and a 58% packet compression rate over the RoHC algorithm in the N3 reference point. The improvement in header compression and package compression are 49.5% and 31.9% over the RoHC algorithm in the F1-U reference point for voice-over-IP service. Both results suggest that the upper bound header compression rate will be increased by about 50% when applying the OH algorithm for voice-over-IP service, vastly improving transmission efficiency and decreasing latency.

For the video-over IP service, the header compression rate is the same as that in the voice-over IP service. In the N3 reference point communication, the packet compression rate is 20.6%. The package compression rates are 21% and 11.5% over the uncompressed package and the RoHC packet, respectively. The performance of OH compression is less apparent in video-over IP service than in voice-over IP service because of the larger payload size. Consequently, applying the OH algorithm in a lower-resolution setting for video-over-IP service is preferred.

6

Simulation Results and Analysis

In this chapter, multiple simulations of packet transmission based on the OH algorithm, RoHC algorithm, and uncompressed choice are designed and conducted to evaluate the performance of the optimized header compression algorithm. The experimental setting is introduced first to give an overview of the simulation environment. Then, the simulation results for different headers are presented to discuss the compressing procedure for each header. Moreover, the real-time simulation results in three scenarios are offered and discussed to prove the excellent performance of the proposed optimized header compression algorithm. A brief conclusion is given finally for analyzing the overall results.

6.1. Experimental Setup

6.1.1. Simulation Environment

To simulate and evaluate the optimized header compression algorithm proposed in this paper, three key entities are included in this simulation, including gNB-DU, gNB-CU-UP, and UPF. The data packet created in the user equipment is transmitted through the DU first, then the CU, and the UPF. In this case, it's feasible to develop three independent DU, CU, and UFP entities in the simulation software to imitate the packet stream passing through them. Moreover, it's also viable to evaluate the capacity of the packet stream based on the mentioned setup. The overall setting of the simulation is shown in Figure 6.1.

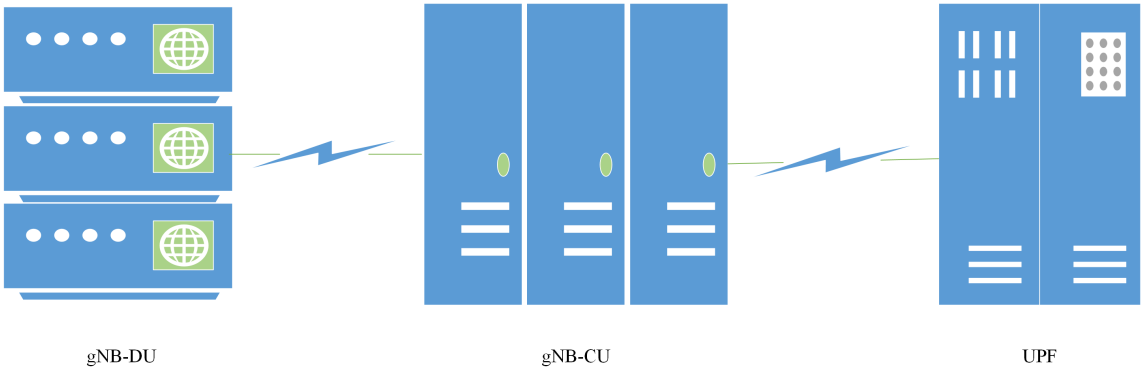


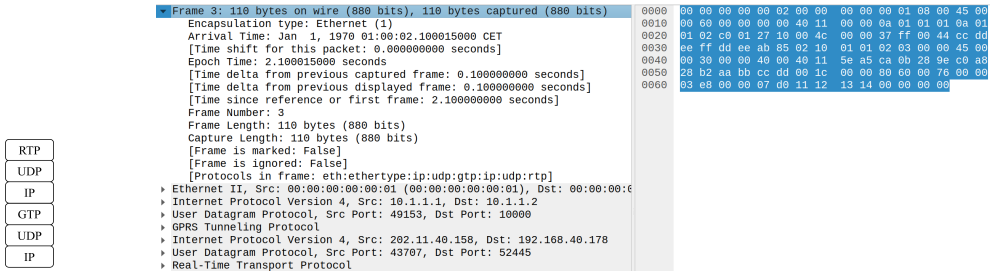
Figure 6.1: Experimental Setup

The 5G communication environment is built on NS-3 [59]. Various communication networks, protocols, and technologies are simulated and evaluated using the free and open-source discrete-event network simulator NS-3. It is designed to offer networking research and teaching in a precise and effective simulation environment. The C++-based network simulator NS-3 provides several prebuilt models for network nodes, channels, and protocols and a flexible foundation for building unique models. It also supports wireless networks, including propagation models and models for various wireless channel types. Researchers, students, and network engineers frequently use NS-3 to simulate and assess the effectiveness of different networking protocols, technologies, and applications. Additionally, it is employed in creating and testing network services and applications.

6.1.2. Data Packet Buildup

Before the simulation, creating a long trace of correct data packets based on the corresponding packet header protocol structure is necessary. These packets will be used in the following simulation. The IP packet capturing and analyzing tool called Wireshark [60] is used for communication protocol development, analysis, and troubleshooting. It offers comprehensive details on each packet sent across the network, enabling users to capture and view network traffic. Wireshark can examine traffic on wired and wireless networks and supports many network protocols. It may apply filters only to record specific traffic, capture packets on various interfaces, and show the data in multiple ways, such as graphs, summary views, and packet details. Wireshark has facilities for in-depth protocol analysis, protocol decoding, network statistics, and fundamental packet capture and analysis capabilities. Network administrators, security experts, and solution developers use it frequently to troubleshoot network problems, examine network traffic for security hazards, and create new protocols and applications.

A data packet over the N3 reference point is caught and illustrated in Figure 6.2b. This example packet possesses 110 bytes and has the protocols of IP, UDP, GTP, IP, UDP, and RTP, as shown in Figure 6.2a. The Ethernet protocol aims to build a simple link between two entities.



(a) Protocol Stack **(b)** Overview of Packet Structure

Figure 6.2: Protocol Stack and Packet Structure

Figure 6.3 illustrates the outer IP header's analysis. The fields of the IP header define multiple configurations, including the 4th version, 20 bytes header length, 96 bytes total length, 0 identification, disabled fragment, 64 bins of time to live, the following header of UDP protocol, a disabled checksum, the source address of 10.1.1.1, and the destination address of 10.1.1.2. The configurations mentioned above define the essential function of the IPv4 packets for data transmission. The header is applied to deliver the IP payload to the next hop on layer 3.

In addition, the actual total length changes according to different services like voice-over IP or video-over IP. The 20 bytes header length is fixed for IPv4. The identification field is zero because there is no fragment in this algorithm. The time to live field is randomly set because it's useless in this design. The checksum field is only disabled here for simplicity; it will be enabled in the actual simulations for verification. The following header of UDP is required since the following protocol in this stack will be the UDP. The source and destination addresses' values are created to locate the sending and receiving addresses.

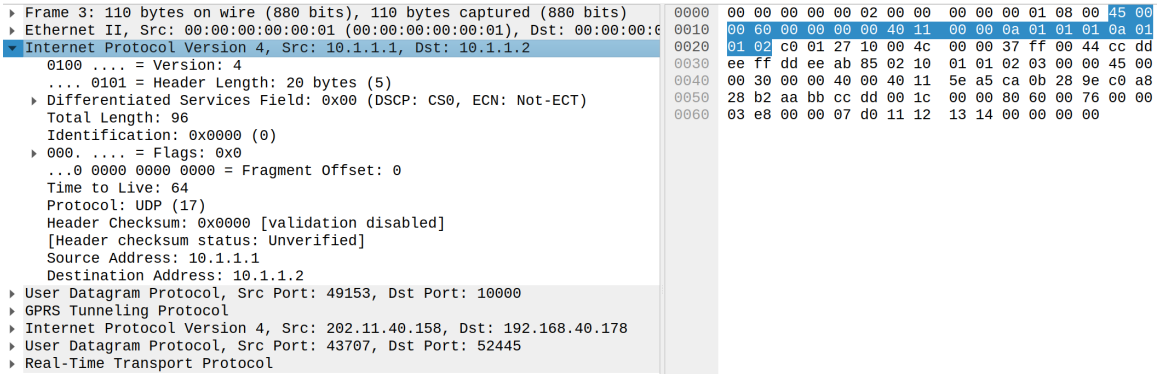


Figure 6.3: Outer IP Header

Figure 6.4 shows the outer UDP header's analysis. The source port is 49153, and the destination port is 10000. The total length calculated from the UDP header is 76 bytes. The checksum of the UDP is set to zero for temporary simplicity. The payload size is 68 bytes. Each of the four fields is 2 bytes. The UDP header is used for sending datagrams over the Internet. It provides a connectionless, unreliable, and unordered delivery of data packets, making it useful for applications

that require fast data transmission with minimal overhead.

The source and destination ports identify the sending and receiving ports. The destination port will be checked at the receiver. The total length is 76 bytes, calculated by subtracting 20 bytes from the IP packet size. The checksum is disabled for simplicity. It will be enabled in the real-time simulation for correctness.

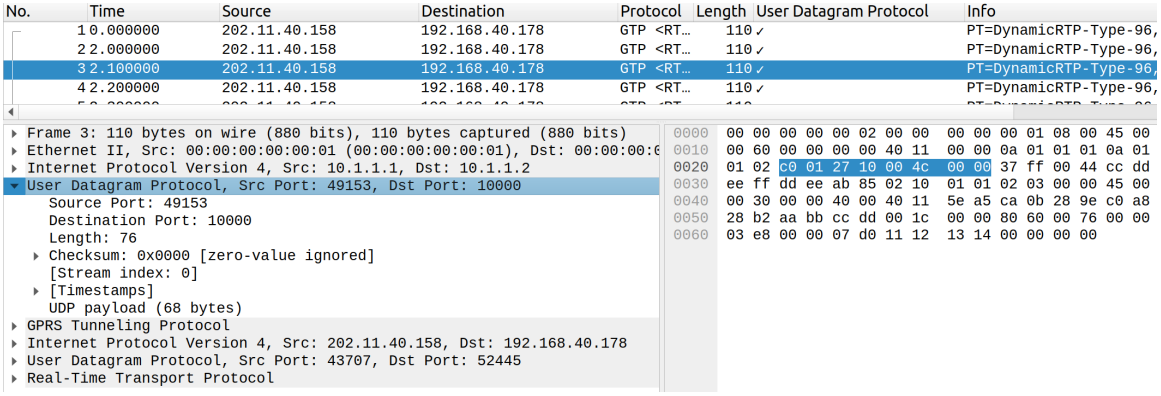


Figure 6.4: Outer UDP Header

The illustration of the GTP-U header is stated in Figure 6.5. The flag is 0x37 for the following: the version of the GTP-U is fixed as one; the protocol type is set to one; the flags of the extension and the sequence number are assigned to one; the N-PDU number flag is set to one. The message type is set to T-PDU. The total length of valuable data is set to 68. The unique TEID is set to 3437096783. The sequence number is assigned to 56814 in this packet for ordering. The N-PDU number is set to 0xab. The following extension header is defined as the PDU session container. The GTP-U header is 12 bytes with 8 bytes extension header. The total length is 20 bytes.

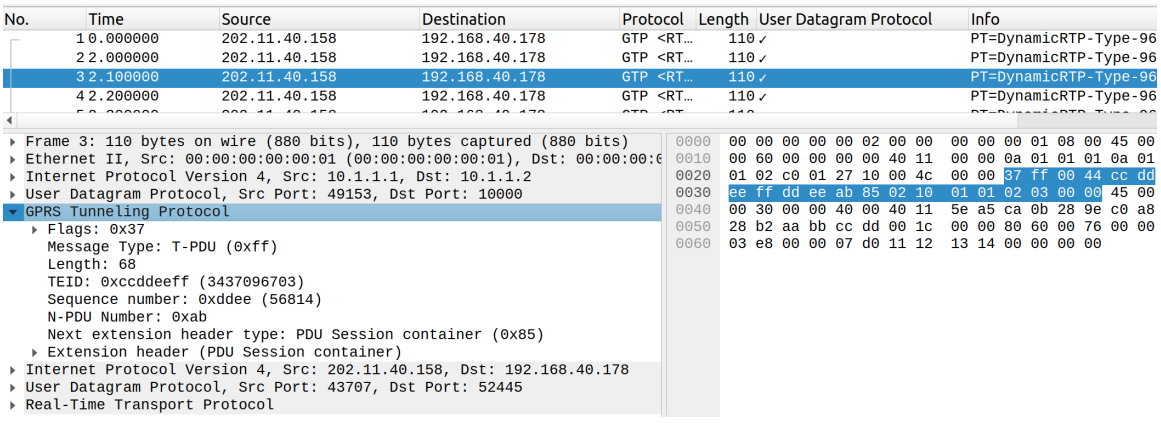


Figure 6.5: GTP-U Header

There is only one version of GTP-U. It's then fixed as one. There are two types of GTP: the normal GTP and the prime GTP. The prime GTP is applied in special tunnels for specific applications, which is not needed in this thesis. The normal GTP is used in this algorithm. The extension flag

is enabled for using the PDU session container extension header. The sequence number flag is enabled to utilize the sequence number to identify the order of the packets in the GTP tunnelling. The N-PDU number flag is enabled to apply the N-PDU number to identify the PDU session in the GTP tunneling. The message type is set to transport PDU for sending the user plane data. The total length is set to limit the size of the valuable data. The TEID is used to identify the unique number of the GTP tunnel between two entities. The sequence number is used to queue the packets by accumulation. The following extension header is defined as the PDU session container for encapsulating the PDU session packets.

The explanation of the PDU-IP header is shown in Figure 6.6. This header is also called the inner IP header since it's already encapsulated in the PDU session. This IP header is not used for routing in gNB. It will be unpacked in the UPF to locate the application in the data network. Most of the fields are identical to the outer IP except the address. The source address is 202.11.40.158. The destination address is 192.168.40.178. The source address is an IP address assigned to the UPF. The destination address is the IP address of the pursuing application entity in the data network. The length is also different, calculated by subtracting 20 bytes from the size of the GTP packet.

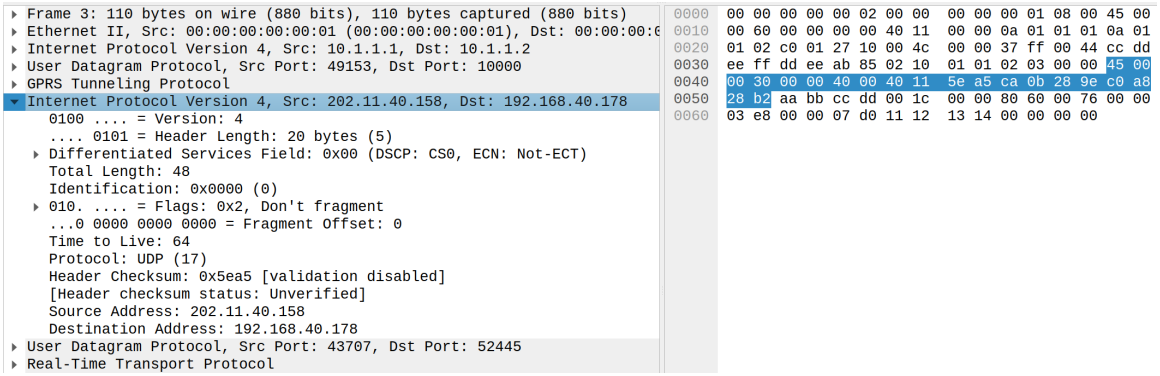


Figure 6.6: PDU-IP Header

The analysis of the PDU-UDP header is illustrated in Figure 6.7. The inner UDP header is also called because it's encapsulated in the PDU session. This UDP header builds the transport layer between the UPF and the DN. The source port is 43707. The destination port is 52445. The value of the destination port will be used to locate the corresponding receiver in the data network. The checksum is temporarily disabled for simplicity. The length is 28 bytes, calculated by subtracting 20 bytes from the size of the inner packet.

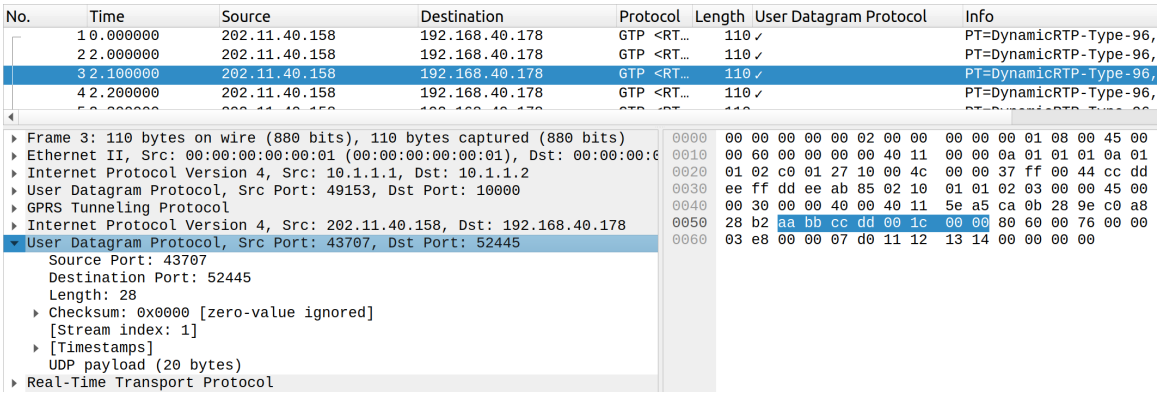


Figure 6.7: PDU-UDP Header

The illustration of the PDU-RTP is shown in Figure 6.8. The RTP header transmits audio and video data over the Internet in real-time. The protocol provides end-to-end network transport functions suitable for applications transmitting real-time data, such as VoIP (Voice over Internet Protocol) and video conferencing. The version of RTP is set to 2. The padding and the extension are disabled. The CSRC count is set to zero for simplicity. The marker is disabled. The payload type is set to the dynamic RTP type. The sequence number is set to 118. The timestamp is 1000. The SSRC is set to 2000. The later bytes are payload.

There is only one version of RTP. The version field is fixed. There is no need to use padding and the extension RTP in this design. The CSRC count should be one or two depending on the service in real-life simulations. The marker marks the critical packet in the stream, which is useless in this design. The payload type is set to dynamic for universal applications. The sequence number is applied to order the packet stream. The timestamp is used to synchronize the transmission. The SSRC should include one or two CSRCs for identifying the source of the payload.

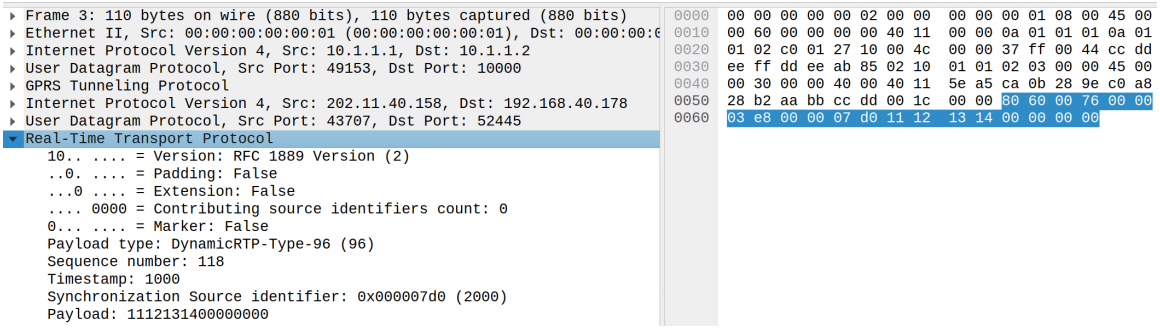


Figure 6.8: PDU-UDP Header

The layers of a protocol stack are organized hierarchically, with each layer depending on the layer below it and providing services to the layer above it. The protocols in the stack will be read and applied from outside to inside to provide corresponding services for offering the payload to the pursuing destination. Some of the fields in the protocols are updated from time to time. Therefore, it's essential to use the checksum to verify the correctness of data at the receiver.

6.2. Simulation Results over F1-U Reference Point

The simulation over the F1-U reference point is conducted in this section. The RoHC algorithm is enabled in this part. There are three types of simulations for different cases.

In the first scenario, the OH algorithm is implemented to optimize the packet flow for the voice-over IP service. In the second scenario, the MOH algorithm is conducted to optimize the parallel packet flows for the voice-video-over IP services. The QoS flow-sending strategy is round-robin, meaning that the packet belonging to each flow will be transmitted by turn. In the third scenario, the VOH algorithm is implemented to optimize the single QoS flow for the smartwatch telemedicine IP communication service.

6.2.1. OH Algorithm for Voice over IP Payload

The simulation results of the uncompressed packets, RoHC-based packets, and OH-based packets are shown in Figure 6.9, respectively.

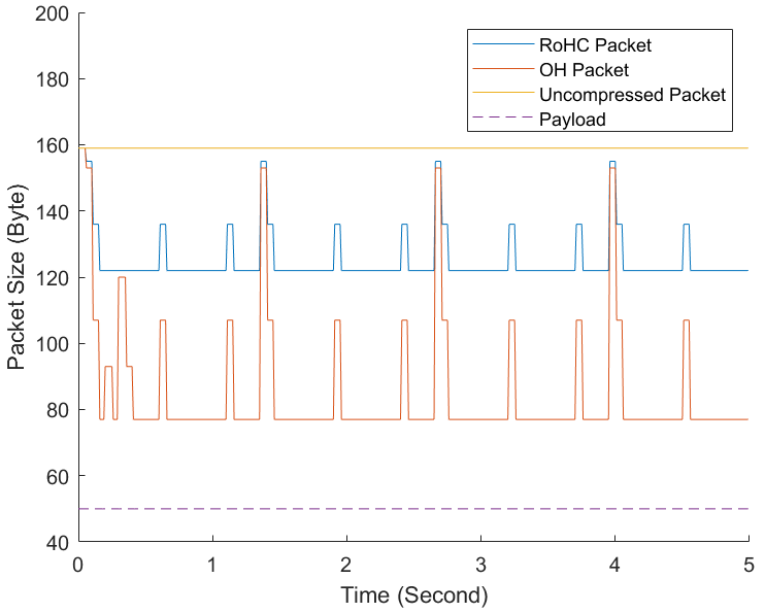


Figure 6.9: Flow of Packets in Scenario 1

The y-axis indicates the size of the packet received at the receiving point. The x-axis indicates the receiving time. The size of the uncompressed packet is 159 bytes (18-byte Ethernet header + 91-byte compressible protocol headers + 50-byte RTP payload). The Ethernet header is generated by the software, and it's not considered in this thesis since it's the layer 2 protocol header. The smallest size of the RoHC packet is 122 bytes (18-byte Ethernet header + 3-byte RoHC header + 51-byte compressible protocol headers + 50-byte payload). The size of the OH packet is 77 bytes (18-byte Ethernet header + 9-byte OH header + 50-byte payload).

The shape of the uncompressed packet flow (shown in color yellow) is a straight line (159 bytes, the Ethernet header is included) since the size of the sending packet is identical, and the simulation environment is assumed to be ideal without interferences like packet loss.

The flow for the RoHC packet varies from 122 bytes to 159 bytes (shown in color blue) because of the periodic refreshing. For communication robustness, the optimal state requires switching back to the sub-optimal state in the state machine after a period of time. For instance, the state machine must change to the IR state after sending packets (that are smaller than the IR packet) 130 times, no matter the present state. Another reason for this size shifting is the error in the fields of the context. If the STATIC field in the context differs from that in the present sending packet, the algorithm must send the IR packet and switch back to the sub-optimal IR state. If the CHANGING field in the context is different from that in the present sending packet and the present state is SO state, the algorithm also must transmit the DYN packet and change back to the FO state. Therefore, the jumping header size is due to the robustness of communication and STATIC and CHANGING fields reparation.

The flow for the OH packet varies from 77 bytes to 159 bytes (shown in color red) because of the same reason mentioned above: refreshing for the robustness of communication and field reparation. Moreover, it's interesting to note that the maximum packet size for the three types of packets is identical since sending the uncompressed packet is necessary to build a solid connection before compressing the packets. Besides, the shape of the packet flow for the OH algorithm is similar to that for the RoHC algorithm because of the mechanism. Both algorithms implement a similar working mechanism (for instance, the periodic refreshing and state transition), resulting in a similar flow shape.

The throughput for the RoHC algorithm and the OH algorithm are shown in Figure 6.10. A detailed data statistic is shown in Table 6.1. The link bit rate is set to 1 Gbps. The data starts recording when the packet arrives at the receiver port.

The throughput flow for the RoHC algorithm (shown in color blue) varies from 12.4 kB/s to 12.8 kB/s. The overall throughput decreases over time, indicating the RoHC algorithm is working to a better state. The lower bound of the speed should be 12.2 kB/s, which is still higher than the minimum speed, meaning the RoHC compression is not optimal. There are intentionally added error packets in the sending part, causing the state switch back in the state machine to test its self-reparation mechanism. In this case, it's reasonable to accept that the RoHC compression is not optimal.

The throughput flow for the OH algorithm (shown in color red) varies from 9.38 kB/s to 7.9 kB/s. The overall throughput decreases over time, which proves that the OH algorithm works properly for compression. The lower bound of the transmission speed should be 7.7 kB/s, as calculated in the last clause. However, the minimum speed is still slightly higher than the lower bound, indicating the algorithm sends more data than expected. It's because of the periodic refreshing and the error packets. Extra sending data than the optimal state is necessary for robustness.

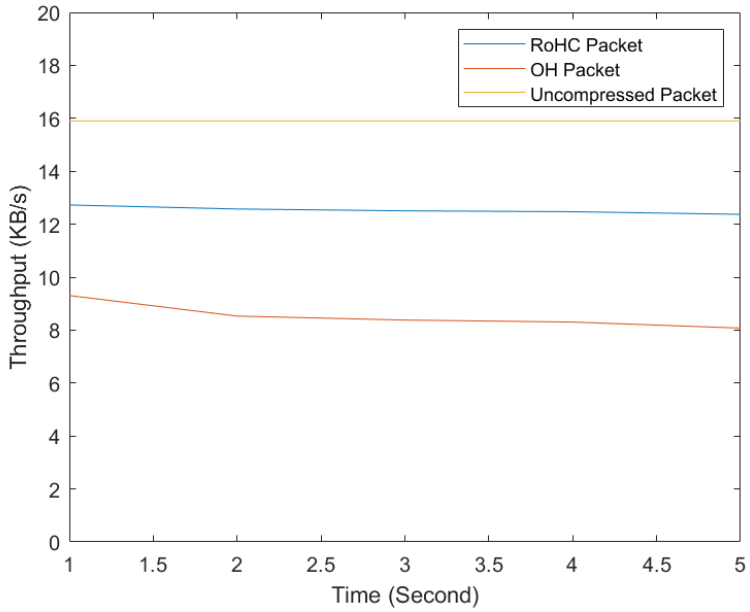


Figure 6.10: Throughput in Scenario 1

The Ethernet header is not considered in the calculation for the compression rate. Compared with the uncompressed speed, the average OH algorithm throughput decreases by 52.31% from 14.1 kB/s to 6.7 kB/s. The average RoHC algorithm throughput decreases by 22.09%. Therefore, the OH algorithm's throughput is decreased by 30.22% compared with the RoHC throughput.

The results indicate that the OH algorithm has successfully increased the transmission efficiency by compressing the size of the protocol headers and the overall throughput in this scenario. Besides, decreasing the transmission latency is predictable since a more flexible sending rate is feasible for congestion control because of lower bandwidth requirements.

Table 6.1: Simulation Results Statistics

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Uncompressed Packet	141	141	141	91
RoHC Packet	104	141	109.86	59.86
OH Packet	59	141	67.24	17.24

As shown in Table 6.1, the size of the Ethernet header is ignored here when calculating the header compression rate since it's the layer 2 header.

For the header compression rate, compared with the uncompressed, the RoHC compression rate is 34.22%, and the OH compression rate is 81.05%. The header compression rate is increased by 46.83% in the OH algorithm.

The execution time for the RoHC and the OH algorithm is illustrated in Figure 6.11. The x-axis means the sequence number of executing the algorithm. The algorithm is totally executed 500 times. The shape of the RoHC algorithm and the OH algorithm fluctuates over time, especially at 150 execution bins and 325 execution bins. When transmitting the packet, it's inevitable to have error packets (for instance, the error packet length), causing the state to switch back and extra header bytes to sending, requesting extra calculation resources and response time. The execution time is then jumping at 150 and 325 execution bins.

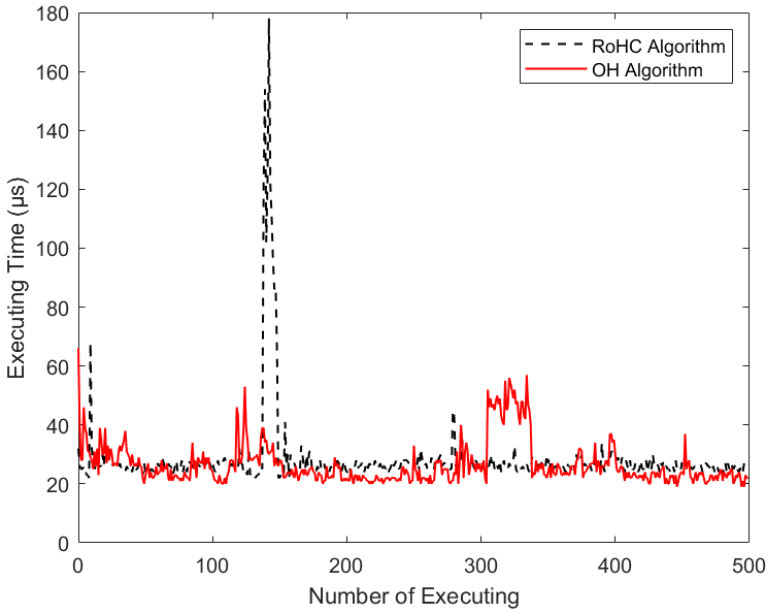


Figure 6.11: Execution Time for RoHC and OH Algorithm

In this simulation, the average execution time for the RoHC algorithm is 28.31 μs, and the average execution time for the OH algorithm is 26.70 μs. It's reasonable to accept that the OH algorithm will need less execution time (decreased by 5.7%) than the RoHC algorithm since fewer scanning and calculating processes are implemented because of the fewer header size in the OH algorithm. In real-life applications, less execution time means lower power consumption and latency in the OH algorithm.

The execution time covers the complete one-time execution of the algorithm, including compression and decompression, excluding the transmission latency. The fluctuation in the execution time is due to two reasons. Firstly, there are errors in packet flows, which will stop the state machine from working in the optimal state. Secondly, the congestion in the data buffer also influences the execution time. In this case, it's more feasible to focus on the average execution time of the algorithm instead of the execution time per execution bin.

It's important to focus on the algorithm execution latency, even if the latency is low in this simulation. The execution latency of the algorithm comes from the implementation of the algorithm and the loading of the data. This latency will be more apparent with more data executed per sec-

ond since the execution of the algorithm is serial, which will cause latency for the communication. When the throughput per second is assumed to be high as 1Gbps, the latency is possibly as high as 10 ms at the same time. Besides, the power consumption is also related to this latency, which is already discussed in Chapter 5.

6.2.2. MOH Algorithm for Two QoS Flows

In this scenario, two QoS flows (audio and video QoS flows) are simulated. The simulation result of packet size for the audio QoS flow is the same as shown in Figure 6.9. For packet size, only the video QoS flow simulation will be shown in Figure 6.12.

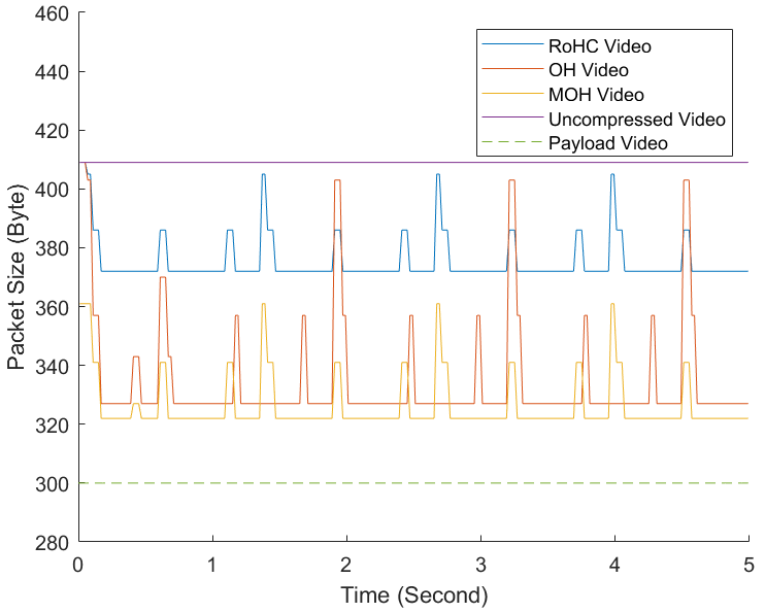


Figure 6.12: Flow of Video Packets in Scenario 2

The uncompressed video QoS flow is shown in purple. The video packet size is 409 bytes (18-byte Ethernet header + 91-byte compressible header + 300-byte payload). The simulation of QoS flow in the RoHC algorithm for video services is shown in color blue. The size of the video RoHC packet is 372 bytes (18-byte Ethernet + 51-byte compressible header + 3-byte RoHC header + 300-byte payload). The throughput of the RoHC packet is shown in Figure 6.13. The data statistic is illustrated in Table 6.2.

The simulation of video QoS flow in the OH algorithm is shown in color red. The size of the video OH packet is 327 bytes (18-byte Ethernet header + 9-byte OH header + 300-byte payload). The corresponding throughput is shown in Figure 6.13. A detailed data statistic is shown in Table 6.3.

The simulation video QoS flow using MOH algorithm is shown in color yellow. The size of the video OH packet is 322 bytes (18-byte Ethernet header + 4-byte OH header + 300-byte payload).

The corresponding throughput is shown in Figure 6.13. A detailed data statistic is shown in Table 6.4.

For the flow of uncompressed packets, the shape fluctuates over time because of the transmission of two types of packets repeatedly. In this case, the bytes change between 159 bytes (audio packet) and 409 bytes (video packet) per 10ms.

For the RoHC algorithm, the flow changes over time. The size varies from 372 bytes to 409 bytes (uncompressed video packet). Besides, the overall value is not as stable as the uncompressed flow. Periodic ups and downs of bytes are viewed. This is because the refreshing and reparation mechanisms work in the RoHC algorithm. When the state machine is in SO state (optimal state) and works for a while (for instance, sending 130 optimal packets), the state machine is required to change back to sub-optimal states (IR state or DYN state) for refreshing. Besides, transmission errors at the sender (for instance, an error packet length in the header) result in the error report at the transmitter. The state machine must switch back to a sub-optimal state to fix the STATIC or CHANGING field errors. In this case, the shape of the RoHC packet flow becomes as shown in Figure 6.12.

For the OH algorithm, the size of the sending packet fluctuates over time. The size varies from 327 bytes to 409 bytes (uncompressed video packet). The uncompressed packet for video is first transmitted to build the connection between two entities. The smaller packet, like the IR packet for initialization and DYN packet for CHANGING field update, is sent after the connection buildup. The state machine switches to the SO state if the STATIC and the CHANGING fields are correct. The state machine in the OH algorithm then starts to transmit the optimal packets (OH packets) for compression. Moreover, the periodic ups and downs are also viewed in Figure 6.12 for OH flow since the basic mechanism of the OH algorithm is similar to the RoHC algorithm.

For the MOH algorithm, the size of the sending packet changes similarly to the OH algorithm. Besides, there are several improvements. Firstly, the size varies from 322 to 361 bytes, decreasing both the packet size and the fluctuation range. This may help to decrease the throughput and increase the robustness. Secondly, the MOH algorithm decreases the optimal header size for the video packet from 327 to 322 bytes, helping to sync better between the audio and the video packet. Similar ups and downs are also viewed for MOH flow, as illustrated in RoHC and OH algorithms.

The throughput for the uncompressed audio and video flows are shown in Figure 6.13. For the uncompressed throughput, the speed should be 28.4 kB/s.

Figure 6.13 also shows the video and audio overall throughput for the RoHC packet over time. The speed changes from 25.3 kB/s to 24.75 kB/s, which decreases over time. The average speed is 25 kB/s over the simulation. The decrease in real-time speed indicates the working of the RoHC algorithm. The compression mechanism tries to decrease the real-time throughput by sending smaller packets. The lower bound of the speed should be 24.7 kB/s. The speed at

4s has approached this value, proving that the state tries to reach the optimal, and the RoHC algorithm seeks to compress more.

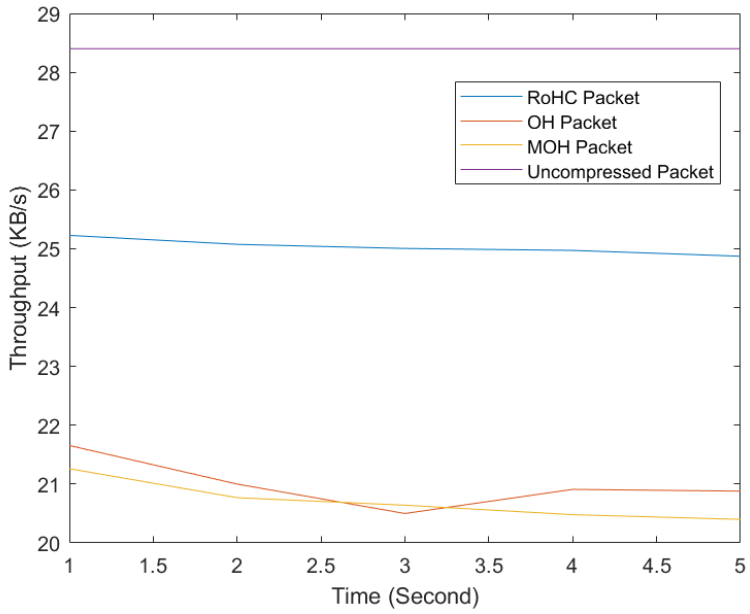


Figure 6.13: Throughput in Scenario 2

Figure 6.13 also illustrates the video and audio overall throughput for the OH packet over time. The speed ranges from 21.7 kB/s to 20.5 kB/s, decreases first, then increases, and finally decreases. It may be because of the transmission error or the two flow merge at the receiver. If the transmission error occurs, the speed is increased for more header transmission to repair the error. Besides, two QoS flow transmissions in the same channel may cause the data to merge at the receiver, resulting in an unstable reading. The lower bound speed is 20.2 kB/s, which is still higher than the real-time speed. The mechanism will work overtime to reach the optimal compression state later. The average speed is 20.9 kB/s

Figure 6.13 also shows the video and audio overall throughput for the MOH packets over time. The speed ranges from 21.3 kB/s to 20.2 kB/s. The shape continuously decreases over time, proving the compression mechanism works properly. The speed lower bound is 19.95 kB/s, as calculated in the last clause. The algorithm will work further to reach this value as much as possible. The average speed is 20.6 kB/s

Table 6.2: Simulation Results Statistics for Two RoHC Flows

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Audio Packet	104	141	107.07	57.07
Video Packet	354	391	357.19	57.19

Table 6.3: Simulation Results Statistics for Two OH Flows

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Audio Packet	59	141	66.41	16.41
Video Packet	309	391	316.25	16.25

Table 6.4: Simulation Results Statistics for MOH Flows

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Audio Packet	59	141	69.15	19.15
Video Packet	304	343	308.13	8.13

More detailed data is shown in Tables 6.2, 6.3, and 6.4. It's interesting to note that the maximum audio packet is the same for the three types of packets since the uncompressed audio packets are transmitted in all the algorithms.

For the RoHC algorithm, the header compression rate is 37.22%, the throughput compression rate is 12.73%. For the OH algorithm, the header compression rate is 82.05%, the throughput compression rate is 28.07%. For the MOH algorithm, the header compression rate is 85.01%, the throughput compression rate is 29.08%.

In comparison with the RoHC and the OH algorithm, the multi-stream optimized OH algorithm increases the header compression rate by 48% and 2% for the overall packet. However, the overall throughput does not change much, with only an average of 16% and 1%. Other advantages are discussed in Figure 6.14 about the execution time. The execution time for the RoHC algorithm, OH algorithm and the MOH algorithm is illustrated in Figure 6.14. The detailed data is shown in Table 6.5.

In Figure 6.14, the execution time for the three algorithms is unstable since two types of packets (audio and video) are transmitted simultaneously. Moreover, there are apparent ups in shape for three algorithms. This may contribute to the transmission error since more calculation resources are required when the error happens. The mechanism will work in the sub-optimal state for sending more data and need more response time.

The average execution time for the RoHC algorithm is 34.34 µs, the average execution time for the OH algorithm is 31.97 µs, and the average execution time for the optimized OH algorithm is 28.84 µs. The optimized OH algorithm decreases the execution time over RoHC and OH algorithms by 16.0% and 9.8%.

Besides, the execution time is different between the video stream and the voice stream. The execution time for the video is larger than the audio for all the algorithms. This is because the

video packet size is much larger than the voice packet, wasting more time on bytes reading and loading. This also proves the importance of compressing data packets from this part.

In this case, it's proved that the multi-stream optimized OH will improve the OH algorithm performance by decreasing the execution time, which leads to lower power consumption and lower latency. Moreover, the improvement of this optimization will be more apparent if more QoS flows are transmitted in the same PDU session.

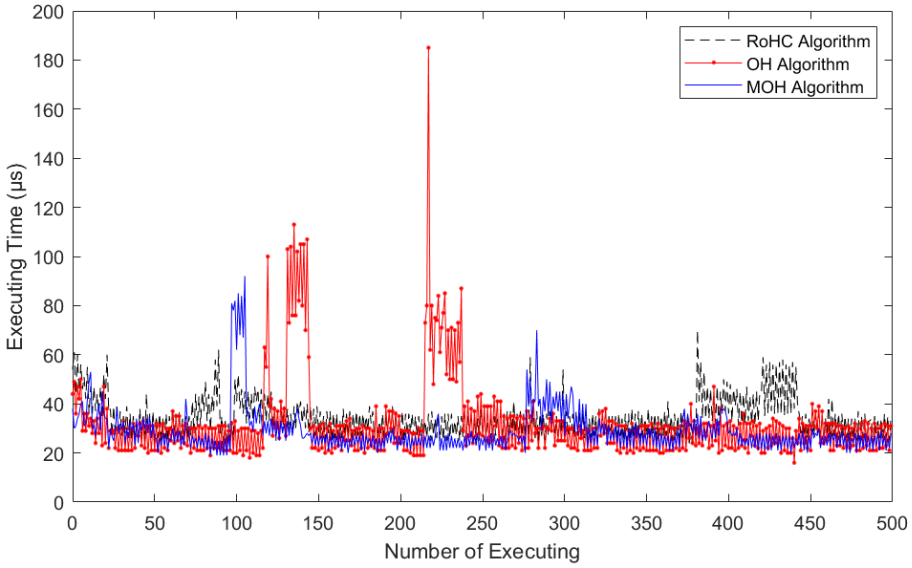


Figure 6.14: Execution Time for RoHC, OH and MOH Algorithms

Table 6.5: Execution Time for Each Stream in Three Scenarios

Stream Type	RoHC(μs)	OH(μs)	MOH(μs)
Voice Stream	28.90	26.23	25.98
Video Stream	39.79	37.71	31.70
Average	34.34	31.97	28.84

6.2.3. VOH Algorithm for Smartwatch Data Payload

For VOH compression, use the VOH packet shown in Table 5.10. For the smartwatch application mentioned in Figure 1.4, Bluetooth Low Energy (BLE) Technology is applied to build the connection between the smartwatch and the smartphone for real-time data transmission because of its low power consumption and high robustness. IP service transfers the packet to the cloud server from the mobile phone. The size of the payload in the BLE 4.2 version is 20 bytes. In this case, the payload size of the IP packet could be 60 bytes for three packets each time. The simulation results of flows in four types of packets are shown in Figure 6.15. Figure 6.16 shows the throughput

of four packet types. The statistical results of the simulation are illustrated in Table 6.6.

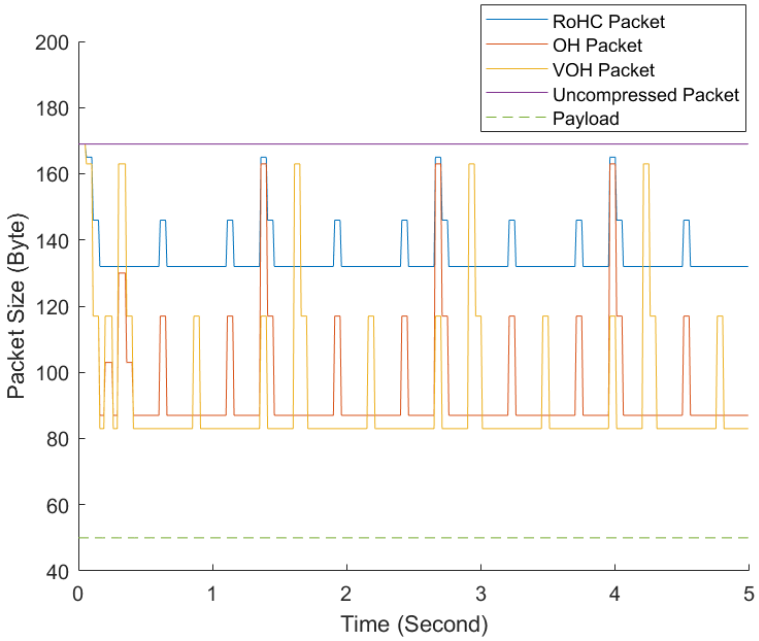


Figure 6.15: Flow of Packets in Scenario 3

Figure 6.15 indicate the transmitted packet size of each time bin. The size of the uncompressed packet is unchanged since the simulation environment is ideal.

The size of the RoHC packet varies from 114 bytes to 151 bytes (the size of the Ethernet header is excluded) because of the periodic refreshing. The optimal state (for example, SO state) working in the state machine must periodically switch back to the sub-optimal state (for example, FO state). A higher size shift means the state transition from SO state to IR state if the IR state countdown is zero or the STATIC field is incorrect. A comparatively lower size shift indicates the state transition from SO state to FO state if the FO state countdown is zero or the CHANGING field is incorrect. Besides, the stair shape means the transition from the IR state to the FO state if the STATIC field is synced successfully.

The size of the OH packet varies from 69 bytes to 151 bytes because of the same reason mentioned above. The minimum size of the OH packet is much smaller than the RoHC packet, attributed to a further header compression development in the OH algorithm. The specific decreased size is 45 bytes, which corresponds to the decreased size in the design (54-byte RoHC header - 9-byte OH header = 45 bytes), proving the consistency between the design and the simulation. The shape of the OH packet flow is similar to the RoHC packet since their basic working mechanism is similar. They use the same refreshing procedures and the same sending packets.

The size of the VOH packet varies from 65 bytes to 151 bytes. The maximum size of the packet is the same in these four types of packets since sending the uncompressed packet in the initial phase is compulsory no matter which type of algorithm is applied. According to the design shown

in Table 5.10, the size of the VOH packet should be 4 bytes smaller than the OH packet in the optimal state, which is also viewed in Figure 6.15. The flow shape of the VOH packet is similar to the OH packet since the same refreshing mechanism is applied to them.

The throughput of uncompressed packet, RoHC packet, OH packet and VOH packet are shown in Figure 6.16.

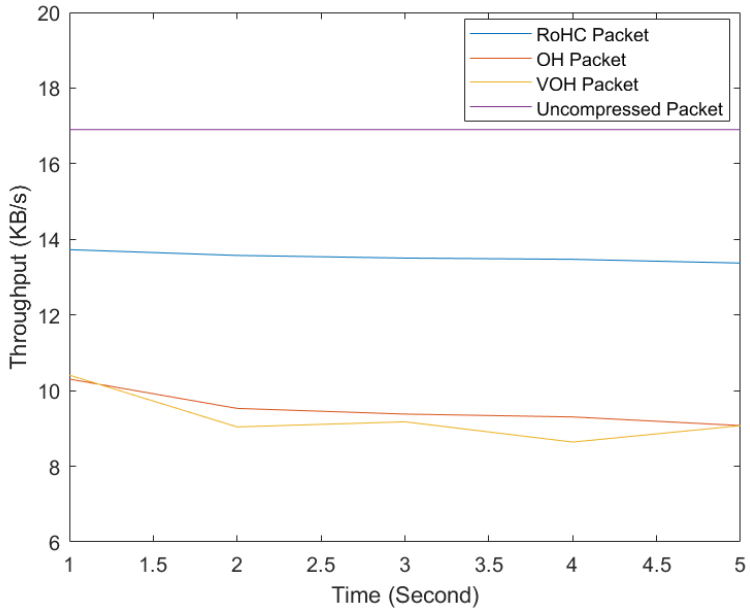


Figure 6.16: Throughput in Scenario 3

The throughput of the uncompressed packet is 16.9 kB/s all the time if the Ethernet header is included. Its straight line is reasonable since the environment is ideal and the uncompressed packet is the same size all the time.

The average throughput of the RoHC packet is 13.5 kB/s if the Ethernet header is included (as shown in Figure 6.16), which decreases by 20% compared with the uncompressed throughput. The shape of RoHC throughput is decreasing all the time, proving its success in compressing the packet. The lower bound of the transmission speed calculated in the last clause is 13.2 kB/s, which is the same speed at 4s in the RoHC simulation, proving that the RoHC algorithm has been stable in the optimal state of the state machine.

The average throughput of the OH packet is 9.5 kB/s if the Ethernet header is included (as shown in Figure 6.16), which is smaller than the uncompressed flow and the RoHC flow by 43.8% and 29.6%, respectively. The OH throughput is decreased by compressing the size of the protocol headers with the help of the OH algorithm, saving the transmission resources and decreasing the latency. The lower bound of the OH throughput should be 8.7 kB/s, as calculated in the last clause. The shape of this throughput is decreasing all the time to 9.1 kB/s at 4s, indicating that the OH algorithm has been in the optimal state but not stable. More or fewer transmission distractions

stop the OH algorithm from performing perfectly.

The average throughput of the VOH packet is 9.27 kB/s, which decreases the size by 2%. The throughput improvement is unapparent since the minimum size of the packet decreased by only 4 bytes. The advantages of this type of packet will be discussed later. The shape of the VOH throughput fluctuates all the time. The lower bound of the speed should be 8.3 kB/s. The vertical optimized OH algorithm does not reach the stable optimal state in this simulation result since the overall throughput increases after 3 seconds. From the analysis of Figure 6.16, the VOH algorithm is more sensitive than the OH algorithm because of its compact algorithm design. In this case, the robustness of the VOH algorithm is worse than the OH algorithm.

Table 6.6: Simulation Results Statistics for Vertical Optimization

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Uncompressed Packet	151	151	151	91
RoHC Packet	114	151	117.3	57.3
OH Packet	69	151	77.2	17.2
VOH Packet	65	151	73.2	13.2

For the RoHC algorithm, the header compression rate is 37.03%, the throughput compression rate is 22.32%. For the OH algorithm, the header compression rate is 81.10%, the throughput compression rate is 48.87%. For the VOH algorithm, the header compression rate is 85.49%, the throughput compression rate is 51.52%.

Compared with the RoHC and the OH algorithm, the VOH algorithm increases the header compression rate by 48.46% and 4.39%. The VOH algorithm increases the packet compression rate by 29.20% and 2.65%.

The execution time for the RoHC algorithm, OH algorithm, and Vertical Optimization OH algorithm is shown in Figure 6.17. The average time for the RoHC algorithm is 29.32 µs, the average time for the OH algorithm is 27.55 µs, and the average time for the Vertical OH algorithm is 17.91 µs. The Vertical-optimized OH algorithm decreases the execution time over the RoHC algorithm and the OH algorithm by 38.9% and 34.9%.

The shape of the execution time for the three types of algorithm is stable except for the first 200 bins. Firstly, the initial phase of the three algorithms should transmit the larger packets (IR packet or DYN packet) for shifting to the optimal state, which requires more calculation resources and response time. Secondly, there are several interferences before 200 execution bins (for instance, STATIC and CHANGING fields error), causing the frequent switch from one state to another in the state machine and extra calculation resources. Therefore, three algorithms have a sudden execution time increase during the first 200 execution bins.

Overall, the Vertical-Optimized OH algorithm improves the performance of the OH algorithm by decreasing the execution time and the size of the overall throughput. However, this new design also brings about a new trade-off of lower robustness. In this case, the robustness improvement should be the next research orientation in the future.

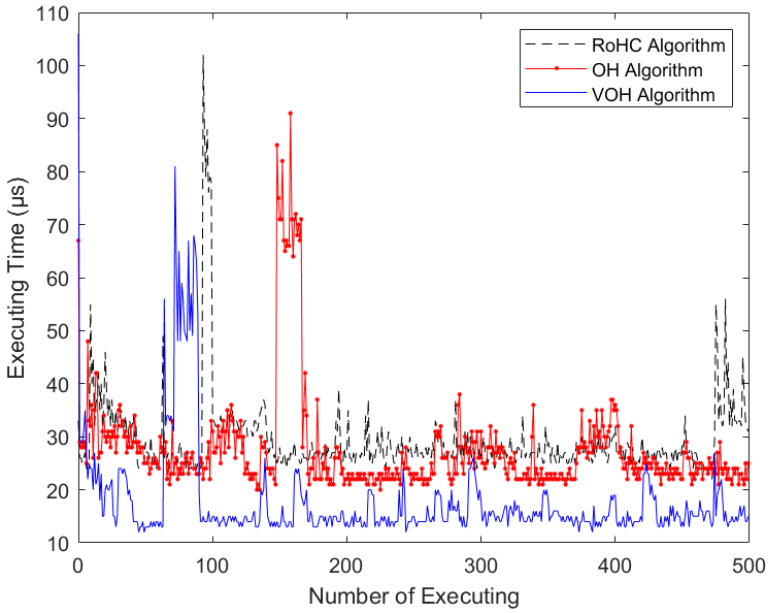


Figure 6.17: Execution Time for RoHC, OH and VOH Algorithm

6.3. Simulation Results over N3 Reference Point

There are some similarities between the simulation of F1-U and N3 reference points. The N3 reference point communication possesses the inner layer RTP/UDP/IP protocols, and outer layer GTP/UDP/IP protocols, the same as illustrated in F1-U reference points. They have a similar working mechanism, leading to similar simulation results.

There are also some differences between the simulation of F1-U and N3 reference points. Firstly, the RoHC algorithm is not enabled in the N3 reference point, resulting in a better compression performance for the OH algorithm. Secondly, the PDCP/SDAP layer is not enabled in the N3 reference point.

Even if there are minor differences between the two reference points, the majority of the simulation should be the same. The N3 reference point simulation analysis is not discussed specifically in this case. A brief simulation analysis and discussion will be given for the packet flow, the throughput, and the execution time in this clause. There are three simulation scenarios, the same as mentioned in F1-U reference point simulation:

Firstly, the Opus coding audio VoIP packet with 50-byte payload packet is transmitted. The uncompressed packet and the OH packet are applied in this case.

Secondly, the Opus coding VoIP audio packet with 50-byte payload packet and the H.264 coding VoIP video packet with 300-byte payload are sent. The uncompressed packet and the OH packet are applied in this scenario. Moreover, the multi-stream optimized OH algorithm is also applied for further header compression toward the video packet.

Thirdly, the BLE packet is transmitted from the smartwatch for telemedicine service. The payload of the BLE 4.2 packet is 20 bytes. Three packets of 60 bytes are combined to transmit through IP service. This packet can be viewed as the IP packet with a 60-byte payload. Furthermore, the vertical optimized OH algorithm is implemented to compress the overall header in a vertical manner.

The simulation results of data flow and transmitted bits per second in three scenarios are shown in Figure 6.18.

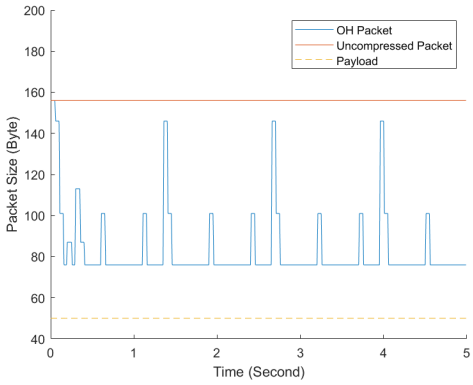
It's noted that the shapes of these data flows are roughly the same as discussed above for the F1-U reference point. In this case, only a brief discussion is presented here.

For the uncompressed packet in the first scenario shown in Figure 6.18a, the size is 156 bytes (18-byte Ethernet header + 88-byte compressible header + 50-byte payload). The shape is a straight line because of the consistent packet sending and the ideal environment.

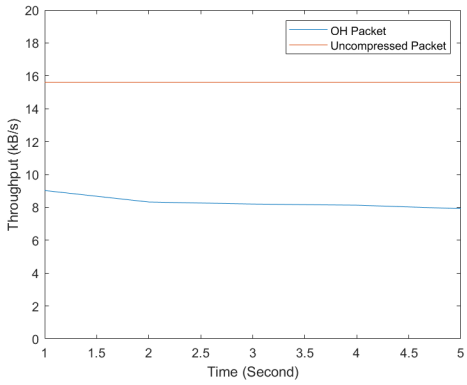
For the OH packet in the first scenario shown in Figure 6.18a, the size varies from 156 bytes to 76 bytes (18-byte Ethernet header + 8-byte OH header + 50-byte payload). The 76-byte OH packet is one byte smaller over N3 than that over F1-U because of the shortage of the PDCP protocol in N3 reference point. The average packet size is 83.20 bytes. The periodic ups and downs have already been discussed above.

For the uncompressed packet in the second scenario shown in Figure 6.18c, the packet is the 406-byte video packet (18-byte Ethernet header + 88-byte compressible header + 300-byte payload). The audio packet in this scenario is the same as shown in Figure 6.18a.

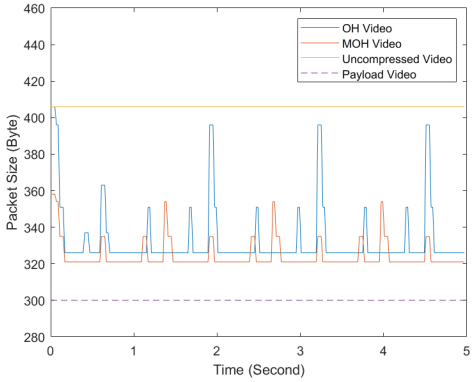
For the OH video flow in the second scenario shown in Figure 6.18c, the packet size varies from 326-byte OH video packets to 406-byte uncompressed ones. The OH audio flow is the same as the OH flow shown in Figure 6.18a. Sending the uncompressed packet is necessary for the OH algorithm for building up communications. The average packet (including the audio and video packets) is 207.66 bytes. For the MOH video packet in the same scenario illustrated in Figure 6.18c, the size varies from 321 to 358 bytes. The audio packet flow in the MOH algorithm is the same as the OH packet flow shown in Figure 6.18a. The average packet size (including the audio and video packets) is 205.03 bytes. The video packet size decreases from 326 to 321 bytes.



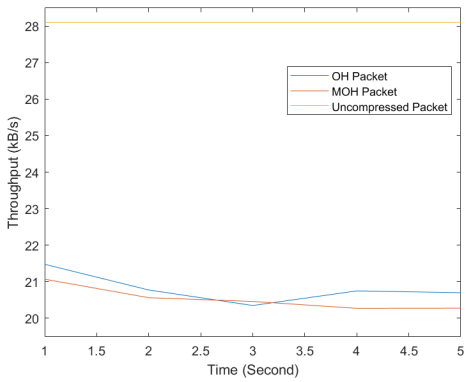
(a) Packet Size in Scenario 1 for N3 Reference Point



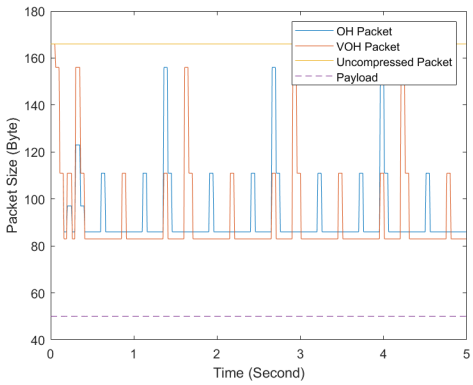
(b) Throughput in Scenario 1 for N3 Reference Point



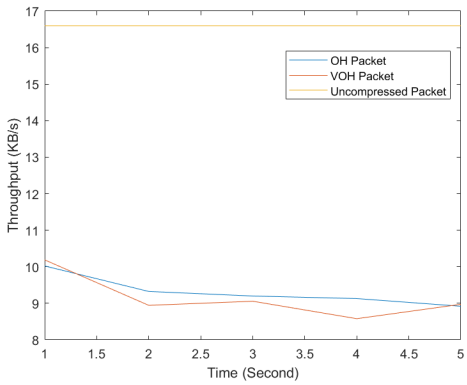
(c) Video Packet Size in Scenario 2 for N3 Reference Point



(d) Overall Throughput in Scenario 2 for N3 Reference Point



(e) Packet Size in Scenario 3 for N3 Reference Point



(f) Throughput in Scenario 3 for N3 Reference Point

Figure 6.18: Data Flow of sending Packets and Throughput in Three Scenarios

For the uncompressed packet flow in the third scenario shown in Figure 6.18e, the packet size is 166 bytes. It's 10 bytes higher than the packet in Figure 6.18a because of the payload size.

For the OH packet flow in the third scenario shown in Figure 6.18e, the packet size varies from 86 to 166 bytes, with an average of 93.2 bytes.

For the VOH flow in the third scenario stated in Figure 6.18e, the packet size varies from 83 to 166 bytes. The header is optimized from 8 to 5 bytes. The main advantage of this optimization is

the decrease in execution time, which will be discussed later.

The shapes of all the throughput figures are similar to that discussed above in the F1-U reference point. A brief discussion is given here:

1. In the first scenario:

For the uncompressed flow with 50 bytes payload, the throughput is shown in Figure 6.18b. The speed is 15.6 kB/s all the time because of the ideal simulation environment and the stable transmission.

For the OH packet flow in the first scenario, throughput is shown in Figure 6.18b. The speed ranges from 9.1 kB/s to 7.8 kB/s. The lower bound speed is 7.6 KB/s, as calculated before. The shape decreases over time, proving the OH algorithm works properly. The average speed is 8.32 kB/s. Compared with the uncompressed flow, the throughput decreases by 46.7%.

2. In the second scenario:

For the uncompressed packet flow in the second scenario with two types of packet, the illustration is shown in Figure 6.18d. The average speed is 28.1 kB/s, the same as before, indicating the correctness of this simulation.

For the OH packet flow in the second scenario, the simulation results are shown in Figure 6.18d. The shape decreases first, then increases, and finally decreases. The possible reasons are discussed above in F1-U. The speed ranges from 21.50 kB/s to 20.30 KB/s. The average speed is 20.78 kB/s. Compared with the uncompressed flow, the throughput decreases by 26%.

For the MOH algorithm, the throughput is illustrated in Figure 6.18d. The shape of this throughput decreases over time, proving the multi-stream optimized OH algorithm works in a good manner for compression. The speed ranges from 21.22 kB/s to 20.23 kB/s. The average speed is 20.50 kB/s. Compared with the uncompressed flow, the throughput decreases by 27%.

3. In the third scenario:

For the uncompressed flow in the third scenario, the illustration is shown in Figure 6.18f. The speed is 16.6 kB/s. For the OH packet flow in the same scenario, the speed ranges from 11.1 KB/s to 8.9 kB/s, with an average of 9.3 KB/s. The lower bound speed should be 8.6 kB/s. Compared with the uncompressed flow, the throughput decreases by 44.0%

Figure 6.18f shows the throughput for the MOH packet flow. The illustration of this shape has been discussed above. The speed ranges from 10.27 kB/s to 8.58 kB/s. The average speed is 9.15 kB/s. Compared with the uncompressed flow, the throughput decreases by 44.9%.

Detailed analysis data are shown in Table 6.7. In the first scenario, the OH algorithm compresses the header size by 82.7%. In the second scenario, the OH algorithm compresses the header

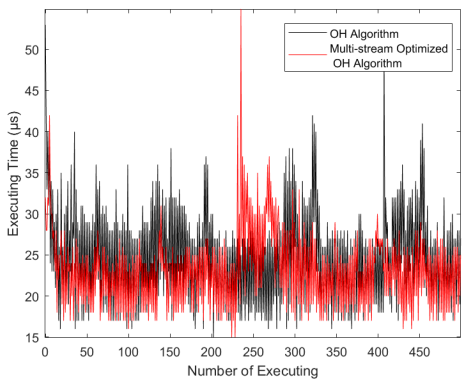
size by 83.2%. The optimized OH algorithm compresses the header size by 86.3%. In the third scenario, the OH algorithm compresses the header size by 82.7%. The optimized OH algorithm compresses the header size by 84.7%.

The execution time for the N3 reference point communication in the second and the third scenario is shown in Figure 6.19a and 6.19b, respectively.

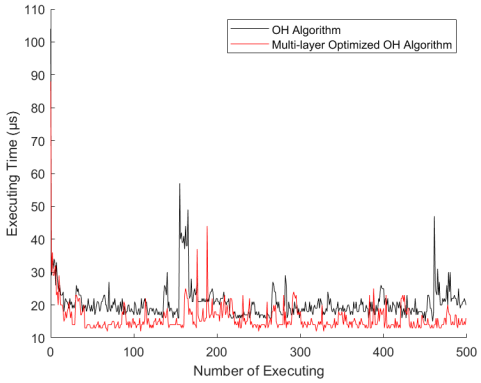
In the second scenario, the average execution time for video packets is decreased from 29.88 µs to 26.38 µs by 11.7%. The overall execution time for both packets decreased from 24.66 µs to 23.12 µs by 6.2%. In the third scenario, the average execution time for the smartwatch packet is decreased from 20.69 µs to 16.04 µs by 22.5%. The results indicate that the optimization toward the OH algorithm in two scenarios has apparently decreased the algorithm execution time, resulting in decreasing the transmission latency.

Table 6.7: Simulation Results Statistics for N3

Packet Type	Min (byte)	Max (byte)	Average (byte)	Average Header (byte)
Uncompressed in Scenario 1	138	138	138	88
OH in Secnario 1	58	138	65.2	15.2
Uncompressed in Scenario 2	92	388	262.7	87.7
OH in Scenario 2	58	388	189.7	14.7
MOH in Scenario 2	58	340	187	12
Uncompressed in Scenario 3	148	148	148	88
OH in Scenario 3	68	148	75.2	15.2
VOH in Scenario 3	65	148	73.5	13.5



(a) Execution Time for N3 in Scenario 2



(b) Execution Time for N3 in Scenario 3

Figure 6.19: Execution Time for N3 Reference Point

6.4. Overall Results Analysis

The overall simulation results for the F1-U and N3 reference points are shown in Table 6.8 and 6.9. The compression rate is calculated by comparing it with the uncompressed packet. The multi-stream optimized OH algorithm is enabled in scenario 2, and the multi-layer optimized OH algorithm is enabled in scenario 3. The RoHC algorithm is only enabled in F1-U simulation.

Table 6.8: Simulation Results Statistics for F1-U

Scenario	Evaluation Metric	RoHC Algorithm	OH Algorithm	MOH Algorithm	VOH Algorithm
Scenario 1	Average Header Compression Rate (%)	34.22	81.05	-	-
	Average Throughput Compression Rate (%)	22.09	52.31	-	-
	Average Algorithm Execution Time(μs)	28.31	26.70	-	-
Scenario 2	Average Header Compression Rate (%)	37.22	82.05	85.01	-
	Average Throughput Compression Rate (%)	12.73	28.07	29.08	-
	Average Algorithm Execution Time(μs)	34.34	31.97	28.84	-
Scenario 3	Average Header Compression Rate (%)	37.03	81.10	-	85.49
	Average Throughput Compression Rate (%)	22.32	48.87	-	51.52
	Average Algorithm Execution Time(μs)	29.32	27.55	-	17.91

In the F1-U reference point simulation, the OH algorithm is capable of improving the header compression rate by 45% in all cases compared with the RoHC algorithm. The decrease in throughput may vary in different scenarios, by 30% in voice-over IP service (Scenario 1), by 15% in voice-video combined service (Scenario 2), and by 26% in smartwatch telemedicine IP service (Scenario 3). The OH algorithm compresses the algorithm execution time by an average of 2 μs (10%) in all cases. Moreover, the optimization toward the OH algorithm in both Scenario 2 and Scenario 3 improves the header compression rate by 3%, compared with the OH algorithm. The optimization toward the OH algorithm decreases the execution time by 3 μs (10%) in Scenario 2 and by 9 μs (35%) in Scenario 3 compared with the OH algorithm, which indicates that the latency and the power consumption will be further decreased when applying the MOH and VOH algorithm in some specific scenarios.

Table 6.9: Simulation Results Statistics for N3

Scenario	Evaluation Metric	RoHC Algorithm	OH Algorithm	MOH Algorithm	VOH Algorithm
Scenario 1	Average Header Compression Rate (%)	-	82.73	-	-
	Average Throughput Compression Rate (%)	-	52.75	-	-
	Average Algorithm Execution Time(μs)	-	21.88	-	-
Scenario 2	Average Header Compression Rate (%)	-	83.24	86.32	-
	Average Throughput Compression Rate (%)	-	27.79	28.82	-
	Average Algorithm Execution Time(μs)	-	29.88	26.38	-
Scenario 3	Average Header Compression Rate (%)	-	82.73	-	84.66
	Average Throughput Compression Rate (%)	-	49.19	-	50.34
	Average Algorithm Execution Time(μs)	-	20.69	-	16.04

The improvement in the N3 reference point simulation is better since the RoHC algorithm is not applied here. The OH algorithm achieves an average of 82% header compression rate in all cases. The OH algorithm decreases the throughput by 50% in Scenario 1 and Scenario 3, by 28% in Scenario 2. Moreover, the optimization toward the OH algorithm improves the header compression rate by 2% in both Scenario 2 and Scenario 3, compared with the OH algorithm. The optimization toward the OH algorithm in Scenario 2 and 3 both decreases the execution time by 4 µs (15%).

To sum up, the OH algorithm is capable of improving the header compression rate by at least 45% in all cases, largely decreasing the bandwidth requirements and latency. Furthermore, the optimization toward the OH algorithm in some specific services enhances the performance by improving the compression rate and decreasing the algorithm execution latency even more.

7

Conclusion

In this thesis, the Optimized Header Compression Algorithm based on the RoHC algorithm is proposed to compress the redundant header information over the 5G protocol stack for real-time communication in 5G networks.

This thesis introduces the development of wireless communication, basic concepts and present applications, which elicits the motivation and essence of this research study. Besides, a literature review is provided to introduce the status development of 5G communication and header compression technique, emphasizing the importance of developing a new header compression algorithm for the next generation of communication. Moreover, the overall user plane protocol stack in 5G communication is illustrated to specify the compression targets in the OH algorithm.

Furthermore, the overall design of the OH algorithm is introduced in detail. It discusses the profiles of classification, working mechanisms for compressing and decompressing, the structure of packets, evaluation metrics, and theoretical results of optimization.

The final results illustrate that the OH algorithm increases the average header compression rate by 45% and decreases average algorithm execution latency by 10% than the RoHC algorithm in all proposed cases. Moreover, the MOH and VOH algorithms optimize the OH algorithm further to decrease the average algorithm execution latency by 20% in each case by applying simpler algorithm structures. These results show that the OH algorithm has successfully optimized the protocol headers over the 5G protocol stack, leading to higher transmission efficiency, lower latency and lower power consumption.

Future research will prioritize enhancing the reliability of the OH algorithm in dynamic network environments, focusing on strengthening the algorithm's maintenance mechanism to improve op-

erational dependability and network stability in random networks.

Bibliography

[1] Joyce Ayoola Adebusola, Adebiyi Ayodele Ariyo, Okeyinka Aderemi Elisha, Adebiyi Marion Olubunmi, and Okesola Olatunji Julius. “An Overview of 5G Technology”. In: *2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS)*. 2020, pp. 1–4. DOI: 10.1109/ICMCECS47690.2020.240853.

[2] Anju Uttam Gawas. “An Overview on Evolution of Mobile Wireless Communication Networks: 1G-6G”. In: *International Journal on Recent and Innovation Trends in Computing and Communication*. Vol. 3. 5. 2015.

[3] “IMT Vision –Framework and overall objectives of the future development of IMT for 2020 and beyond”. In: *M Series: Mobile, radio determination, amateur and related satellite services*. 2015. URL: https://standards.globalspec.com/std/9964221/ITU-R%20M.2083#:~:text=IMT%20Vision%20%E2%80%93%20Framework%20and%20overall%20objectives%20of,in%20light%20of%20the%20roles%20that%20IMT%20could.

[4] Haesik Kim. “Enhanced Mobile Broadband Communication Systems*”. In: *Design and Optimization for 5G Wireless Communications*. 2020, pp. 239–302. DOI: 10.1002/9781119494492.ch7.

[5] Naser Hossein Motlagh, Ibrahim Afolabi, Matteo Pozza, Miloud Bagaa, Tarik Taleb, Sasu Tarkoma, and Hannu Flinck. “mMTC Deployment over Sliceable Infrastructure: The Megasense Scenario”. In: *IEEE Network* 35.6 (2021), pp. 247–254. DOI: 10.1109/MNET.111.2100093.

[6] Rashid Ali, Yousaf Bin Zikria, Ali Kashif Bashir, Sahil Garg, and Hyung Seok Kim. “URLLC for 5G and Beyond: Requirements, Enabling Incumbent Technologies and Network Intelligence”. In: *IEEE Access* 9 (2021), pp. 67064–67095. DOI: 10.1109/ACCESS.2021.3073806.

[7] Mauro Conti, Denis Donadel, and Federico Turrin. “A Survey on Industrial Control System Testbeds and Datasets for Security Research”. In: *IEEE Communications Surveys Tutorials* 23.4 (2021), pp. 2248–2294. DOI: 10.1109/COMST.2021.3094360.

[8] Shan Lu, Anzhi Wang, Shenqi Jing, Tao Shan, Xin Zhang, Yongan Guo, and Yun Liu. “A study on service-oriented smart medical systems combined with key algorithms in the IoT environment”. In: *China Communications* 16.9 (2019), pp. 235–249. DOI: 10.23919/JCC.2019.09.018.

[9] Wrya Monnet. “IoT, IoE and Tactile Internet”. In: *The Tactile Internet*. 2021, pp. 65–93. DOI: 10.1002/9781119881087.ch5.

[10] Máté Tömösközi, Martin Reisslein, and Frank H. P. Fitzek. “Packet Header Compression: A Principle-Based Survey of Standards and Recent Research Studies”. In: *IEEE Communications Surveys Tutorials* 24.1 (2022), pp. 698–740. DOI: 10.1109/COMST.2022.3144473.

[11] Puneet Jain. “A Release 18 Update”. In: 2022. URL: https://www.3gpp.org/news-events/3gpp-news/rel18-sa2.

[12] Alain Sultan. “5G System Overview”. In: 2022. URL: https://www.3gpp.org/technologies/5g-system-overview.

[13] 3GPP. “Release 18”. In: 2023. URL: https://www.3gpp.org/specifications-technologies/releases/release-18.

[14] Gnana Jayanthi J and Albert Rabara S. “IPv4 addressing architecture in IPv6 network”. In: *2010 2nd International Conference on Advanced Computer Control*. Vol. 3. 2010, pp. 282–287. DOI: 10.1109/ICACC.2010.5486617.

[15] Raja Kumar Murugesan, Sureswaran Ramadass, and Rahmat Budiarto. “Increased performance of IPv6 packet transmission over ethernet”. In: *2009 2nd IEEE International Conference on Computer Science and Information Technology*. 2009, pp. 171–175. DOI: 10.1109/ICCSIT.2009.5234738.

[16] Coe E, Stepanek J, and Raghavendra C. “Analysis of UDP performance enhancement proxies for mobile users”. In: *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*. Vol. 4. 2003, 2708–2711 Vol.4. DOI: 10.1109/VETECF.2003.1286059.

[17] Haohang Li, Yufeng Zhao, and Runzhi Wu. “Optimal Design of UDP Protocol in Embedded Real-Time OS”. In: *2021 13th International Conference on Advanced Infocomm Technology (ICAIT)*. 2021, pp. 180–184. DOI: 10.1109/ICAIT52638.2021.9702066.

[18] “11th IEEE International Conference on Advanced Thermal Processing of Semiconductors. RTP 2003”. In: 2003. DOI: 10.1109/RTP.2003.1249116.

[19] Adriana Lipovac, Anamaria Bjelopera, Ivan Grbavac, Ines Obradović, and Tomo Sjekavica. “Practical Cross-Layer Testing of HARQ-Induced Delay Variation on IP/RTP QoS and VoLTE QoE”. In: *2019 European Conference on Networks and Communications (EuCNC)*. 2019, pp. 262–266. DOI: 10.1109/EuCNC.2019.8801978.

[20] Share Technote. *IP/Network*. URL: https://www.sharetechnote.com/html/IP_Network_OSI7Layer.html.

[21] CSDN. *OSI seven-layer model, TCP/IP four-layer model*. 2020. URL: https://blog.csdn.net/wwy0324/article/details/109310658.

[22] Juniper Networks. “GTP Versions and GPRS Interfaces Overview”. In: URL: https://www.juniper.net/documentation/en_US/junos-mobility11.2/topics/concept/gtp-mobility-versions-overview.html.

[23] Tejmani Sinam, Irengbam Tilokchan Singh, Pradeep Lamabam, Ngasham Nandarani Devi, and Sukumar Nandi. “A technique for classification of VoIP flows in UDP media streams using VoIP signalling traffic”. In: *2014 IEEE International Advance Computing Conference (IACC)*. 2014, pp. 354–359. DOI: 10.1109/IAdCC.2014.6779348.

[24] Indrajati Haryono, Dodi Wisaksono Sudiharto, and Aji Gautama Putrada. “QoS Improvement Analysis of VoIP Service Which Uses Overlay Network. Case Study: Calling AWS VoIP Gateway From Bandung, Indonesia”. In: *2018 International Seminar on Application for Technology of Information and Communication*. 2018, pp. 381–387. DOI: 10.1109/ISEMANTIC.2018.8549807.

[25] Share Technote. *5G/NR - NAS*. URL: https://www.sharetechnote.com/html/5G/5G_PDUSessionEstablishment.html.

[26] Internet Engineering Task Force (IETF). “RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed”. In: 2001. URL: https://datatracker.ietf.org/doc/html/rfc3095.

[27] Share TechNote. “4G/LTE - Core Network”. In: URL: https://www.sharetechnote.com/html/Handbook_LTE_GTP.html.

[28] Ana Carolina Minaburo. “ROHC Interaction in the 3GPP Architecture”. In: 2002. URL: https://www.3g4g.co.uk/Other/ROHC/Presentations/MINABURO.pdf (visited on 08/2002).

[29] The OpenSource ROHC library. In: 2019. URL: https://github.com/didier-barvaux/rohc (visited on 09/24/2019).

[30] 3GPP. “Release 15”. In: 2019. URL: https://www.3gpp.org/specifications-technologies/releases/release-15.

[31] 3GPP. “NR; NR and NG-RAN Overall description; Stage-2”. In: 2023. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3191.

[32] 3GPP. “NG-RAN; Architecture description”. In: 2023. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3219.

[33] 3GPP. “General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)”. In: 2022. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=1699.

[34] 3GPP. “NG-RAN; F1 data transport”. In: 2022. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3261.

[35] Internet Engineering Task Force (IETF). “RObust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed”. In: 2008. URL: https://datatracker.ietf.org/doc/html/rfc5225.

[36] Effnet. “Effnet ROHCv2 Saves bandwidth and improves QoS”. In: URL: http://www.effnet.com/pdf/Effnet_ROHCv2_Datasheet.pdf.

[37] Internet Engineering Task Force (IETF). “The RObust Header Compression (ROHC) Framework”. In: 2010. URL: https://datatracker.ietf.org/doc/html/rfc5795.

[38] Devishree Naidu and Rakhi Tapadiya. “Implementation of Header Compression in 3GPP LTE”. In: *2009 Sixth International Conference on Information Technology: New Generations*. 2009, pp. 570–574. DOI: 10.1109/ITNG.2009.257.

[39] Xiuliang Chen, Fuliang Guo, Ping Dang, and Libing Wu. “A survey of ROHC header compression schemes”. In: *Proceedings of 2012 2nd International Conference on Computer Science and Network Technology*. 2012, pp. 331–335. DOI: 10.1109/ICCSNT.2012.6525949.

[40] Whai-En Chen, Wei-Che Chien, Chin-Feng Lai, and Han-Chieh Chao. “Promising Framework of Ethernet Header Compression in Industrial Internet of Things”. In: *2019 International Conference on Intelligent Computing and its Emerging Applications (ICEA)*. 2019, pp. 134–139. DOI: 10.1109/ICEA.2019.8858285.

[41] Mate Toemoeskoezi, Ievgenii Tsokalo, Sreekrishna Pandi, Frank H.P. Fitzek, and Peter Ekler. “Header Compression in Opportunistic Routing”. In: *European Wireless 2018; 24th European Wireless Conference*. 2018, pp. 1–6.

[42] Jianan Sun, Ping Dong, Yajuan Qin, Tao Zheng, Xiaoyun Yan, and Yuyang Zhang. “Improving bandwidth utilization by compressing small-payload traffic for vehicular networks”. In: vol. 15. 4. 2019, p. 1550147719843050. DOI: 10.1177/1550147719843050.

[43] Woosuk Kwon, Jaeho Hwang, Hyun-Koo Yang, Sunghee Hwang, Kazuyuki Takahashi, and Lachlan Michael. “The ATSC Link-layer Protocol (ALP): Design and Efficiency Evaluation”. In: *IEEE Transactions on Broadcasting* 62.1 (2016), pp. 316–327. DOI: 10.1109/TBC.2015.2506983.

[44] Manlin Fang, Dong Li, Han Zhang, Lisheng Fan, and Imene Trigui. “Performance analysis of short-packet communications with incremental relaying”. In: *Computer Communications* 177 (2021), pp. 51–56. ISSN: 0140-3664. DOI: https://doi.org/10.1016/j.comcom.2021.06.007. URL: https://www.sciencedirect.com/science/article/pii/S0140366421002310.

[45] 3GPP TS 29.281 version 13.3.0 Release 13. *Universal Mobile Telecommunications System (UMTS); LTE; General Packet Radio System (GPRS) Tunnelling Protocol User Plane (GTPv1-U)*. 2017. URL: https://www.etsi.org/deliver/etsi_ts/129200_129299/129281/13.03.00_60/ts_129281v130300p.pdf.

[46] 3GPP. “System architecture for the 5G System (5GS)”. In: 2022. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3144.

[47] 3GPP. “Study of separation of NR Control Plane (CP) and User Plane (UP) for split option 2”. In: 2018. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3307.

[48] Darshan Srinivas. *Optimised user plane routing in a 5G mobile communications network*. 2022. URL: http://resolver.tudelft.nl/uuid:0171df3e-9f12-4662-8527-7f9991c87b40.

[49] 3GPP. “NG-RAN; NG Application Protocol (NGAP)”. In: 2023. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3223.

[50] Internet Engineering Task Force (IETF). “Stream Control Transmission Protocol”. In: 2007. URL: https://datatracker.ietf.org/doc/html/rfc4960.

[51] 3GPP. “Non-Access-Stratum (NAS) protocol for 5G System (5GS); Stage 3”. In: 2023. URL: https://portal.3gpp.org/desktopmodules/Specifications/SpecificationDetails.aspx?specificationId=3370.

[52] Thomas Pfeiffer. “Next Generation Mobile Fronthaul Architectures”. In: *Optical Fiber Communication Conference*. Optica Publishing Group, 2015, M2J.7. DOI: 10.1364/OFC.2015.M2J.7. URL: https://opg.optica.org/abstract.cfm?URI=OFC-2015-M2J.7.

[53] Mai Hassan, Syed Hassan Raza Naqvi, and Pin-Han Ho. “On CPRI based Front-hauling over Residential Power Lines”. In: *2020 International Conference on Networking and Network Applications (NaNA)*. 2020, pp. 47–52. DOI: 10.1109/NaNA51271.2020.00016.

[54] Larry. *CPRI vs eCPRI: What Are Their Differences and Meanings to 5G?* 2021. URL: https://community.fs.com/blog/cpri-vs-ecpri-differences-and-meanings-to-5g.html.

[55] Maté Tömösközi, Patrick Seeling, and Frank H. P. Fitzek. “Performance evaluation and comparison of RObust Header Compression (ROHC) ROHCv1 and ROHCv2 for multimedia delivery”. In: *2013 IEEE Globecom Workshops (GC Wkshps)*. 2013, pp. 544–549. DOI: 10.1109/GLOCOMW.2013.6825044.

[56] “AppScope: Application energy metering framework for Android smartphones using kernel activity monitoring”. In: *Proceedings of the 2012 USENIX Annual Technical Conference, USENIX ATC 2012*. Proceedings of the 2012 USENIX Annual Technical Conference, USENIX ATC 2012. USENIX Association, 2019, pp. 387–400.

[57] Matlab-Mathworks. 2023. URL: https://nl.mathworks.com/?s_tid=gn_logo.

[58] Shingchern D. You and Po-Yueh Lai. “On the Efficiency Difference Between Range and Huffman Coding on CELT Layer of Opus Audio Coder”. In: *2018 IEEE International Conference on Consumer Electronics-Taiwan (ICCE-TW)*. 2018, pp. 1–2. DOI: 10.1109/ICCE-China.2018.8448721.

[59] NS-3 Network Simulator. 2023. URL: https://www.nsnam.org/.

[60] Wireshark Software. 2023. URL: https://www.wireshark.org/.

[61] Mostafa El-Khamy, Jungwon Lee, and Inyup Kang. “Detection Analysis of CRC-Assisted Decoding”. In: *IEEE Communications Letters* 19.3 (2015), pp. 483–486. DOI: 10.1109/LCOMM.2014.2388229.

[62] Jeounglak Ha and Young-Il Choi. “Support of a Multi-access Session in 5G Mobile Network”. In: *2019 25th Asia-Pacific Conference on Communications (APCC)*. 2019, pp. 378–383. DOI: 10.1109/APCC47188.2019.9026455.

A

Calculation for Optimized Header Compression Algorithm

There are multiple calculation requirements in OH compression, especially for the CRC check and INFERRED fields. The CRC check is used for OH packet verification. The INFERRED field is usually used for uncompressed packet verification. Both of them will be discussed in this appendix.

A.1. CRC Check

The Cyclic Redundancy Check is referred to as CRC. It is a type of error-detecting code that is frequently used in digital communication systems to find errors that could happen while transmitting data. A checksum is generated by CRC based on the data to be transferred. The data is then transferred with this checksum applied to it. The identical checksum computation is made on the received data at the receiving end, and the outcome is compared to the checksum that was communicated. The data is presumed to be transmitted correctly if the two values line up [61].

A data transmission error occurs if the calculated checksum at the receiving end does not match the sent checksum. To confirm the accuracy of the data received, the receiver might then request a retransmission of the data or take other suitable actions. The CRC error detection mechanism, popular in many communication protocols like Ethernet, USB, and Bluetooth, is straightforward and effective [61].

A.1.1. Types of CRC Check in Optimized Header Compression

There are three types of CRC used in OH compression:

1. 8-bit CRC check: $C(x) = 1 + x + x^2 + x^8$
2. 4-bit CRC check: $C(x) = 1 + x + x^4$
3. 3-bit CRC check: $C(x) = 1 + x + x^3$

where $C(x)$ means the CRC polynomial, and x indicates the digits of the polynomial.

The 8 bytes CRC check is applied for the IR and the DYNAMIC packets. The 4 and 3 bytes CRC check are applied for the OH packets. The polynomial of 8-bit CRC is 0x07, and the polynomial of 4-bit CRC and 3-bit CRC is 0x03.

A.1.2. Calculation Preparation

The entire calculation is performed under the binary. There are several definitions required before calculating the CRC:

1. Dividend: It's composed of two parts. The first part is the parameter that waits for CRC calculation. The second part consists of a few zeros that concatenate the first part. The number of zeros is the same as the number of bytes in the CRC check. For instance, if an 8 bytes CRC check is calculated, eight zeros will be concatenated after the parameters.
2. Divider: The divider is defined by the polynomial of the CRC. The polynomial of 3 bytes CRC defines the divider as binary 0011, the polynomial of 7 bytes CRC defines the divider as binary 01001111, and the polynomial of 8 bytes CRC defines the divider as binary 00000111.
3. Input Data Inversion: It's optional to inverse the input data since the order of the input data would be different.
4. Output Data Inversion: It's recommended to inverse the output data if the input data is inversed.

A.1.3. CRC Calculation

Use the binary number corresponding to the generator polynomial to perform modulo 2 division on the dividend (high bit alignment), equivalent to a bitwise XOR. The CRC result is the remainder.

For instance, in the IR packet, the first byte is 10000100, and the value corresponding to 8 bytes CRC polynormal is 00000111, the remainder of the modulo 2 division of 10000100,00000000/00000111 is 10010101. The remainder, 10010101, will be the next divider to calculate the CRC for the next byte in the IR packet.

A.2. INFERRED Fields Calculation

There are two types of INFERRED fields in the OH compression. The first is the packet length in the IP and the UDP packets, and the second is the header checksum in the IP and the UDP packets.

A.2.1. Packet Length Calculation

The packet length field is used to calculate the total bytes of the packets, including the headers. The packet length field will be recalculated in the receiver to review the length of the packets in case of losing part of the packet. The checksum field is used to calculate the sum of the headers except for the checksum itself. It's also recalculated in the receiver to verify the correctness of the header fields.

For packet length calculation, it's performed as follows:

1. Determine the size of the IP or the UDP header. The IP header is a fixed-size header that is 20 bytes for IPv4. The UDP header is a fixed-size header that is 8 bytes.
2. Determine the payload size (data) that will be included in the IP or the UDP packet.
3. Add the size of the IP or the UDP header and the payload to calculate the total packet length.

For example, if you have an IPv4 packet with a header size of 20 bytes and a payload of 100 bytes, the packet length would be 120 bytes. Similarly, if you have a UDP packet with a header size of 8 bytes and a payload of 200 bytes, the packet length would be 208 bytes.

A.2.2. Checksum Calculation

The IP (UDP) packet checksum ensures data integrity and is calculated over the entire IP (UDP) header. Here are the steps to calculate the IP (UDP) checksum:

1. Split the IP (UDP) header into 16-bit words (2 bytes); the checksum should be set to zero first. If the packet's total length is an odd number, add a padding byte of zero to the end.
2. Sum all the 16-bit words of the header together using one's complement arithmetic. The calculation for the IP protocol checksum covers the entire header of the IP protocol. The calculation for the UDP checksum covers the UDP header, IP header, and IP message.
3. The resulting value is the IP (UDP) checksum, which should be placed in the IP (UDP) header.

B

Versions of VoIP

Real-time media can be transmitted over an IP network using Voice over IP and Video over IP. VoIP is the technology that permits voice communication over IP-based networks, including the Internet. This technology converts analog voice signal to digital form and compresses it into tiny packets that may be sent via IP networks. These packets are received and transformed back into analog signal. VoIP has many uses, including phone conversations, video conferencing, and instant messaging. It is frequently used in households and businesses as a low-cost substitute for conventional phone systems [23].

Similar to VoIP, Video over IP sends video signals over IP networks. Video signal is converted to video calling, compressed into packets, and sent over the IP network. Applications for this technology include live streaming, video-on-demand services, and video conferencing. VoIP technologies require a high-quality network connection with enough bandwidth and low latency to achieve high-quality media transmission. They also rely on the caliber of the codecs employed for the compression and decompression of media signals [24].

B.1. Voice over IP

In VoIP, voice data is compressed into small packets and transferred across IP networks using a variety of coding (or compression) algorithms. These consist of the following [23, 24]:

1. G.711: Voice data is compressed using this industry-standard method into 64 kbps digital streams. It has a reputation for being high caliber yet uses much bandwidth. It samples audio at a rate of 8 bits/ sample, and the payload size in a packet will be 20-160 bytes depending on the packetization time.

2. G.729: This low-bandwidth technique turns audio data into digital streams at a rate of 8 kbps. Due to its ability to transmit with little quality loss over low bandwidth connections, it is frequently utilized in VoIP applications. The payload size in a packet will depend on the packetization time.
3. G.723: This algorithm creates 6.3 kbps or 5.3 kbps digital streams from voice data. For connections with low bandwidth, it is frequently utilized in VoIP applications. The payload size in a packet will be either 40 bytes or 20 bytes
4. G.726: This algorithm converts voice data into digital streams at 16, 24, 32, or 40 kbps. For digital circuits, VoIP applications frequently use it. The payload size in a packet will be either 40 bytes or 20 bytes.
5. Opus: Opus is an open-source codec that can turn audio data into digital streams ranging in bitrate from 6 kbps to 512 kbps. It can adapt to different network conditions and is made to be used in various applications. Depending on the frame size and bitrate used, the payload size in a packet varies from a few to 120 bytes. The packetization time is commonly set to 20 ms or 40 ms for VoIP applications like WhatsApp, and the bitrate is constantly adjusted based on the network conditions. The Opus payload size of a packet at a frame size of 20 ms can range from 12 bytes for very low bitrates to 120 bytes for very high bitrates. The Opus payload in a packet can be between 24 and 240 bytes in size at a frame size of 40 ms.

B.2. Video over IP

In Video over IP (VoIP), video data is also compressed into small packets and transferred across IP networks using a variety of coding (or compression) algorithms. These consist of the following [23, 24]:

1. H.264/AVC (Advanced Video Coding): This is a widely accepted standard for video compression for video streaming and conferencing. It offers high-quality video at low bitrates and is compatible with most platforms and devices. When H.264/AVC is utilized for a video conversation, the resolution, frame rate, and particular configuration options used to encode the video stream can all affect the packet's payload size. Though frequently between 100 to 1500 bytes, the payload size for H.264/AVC video packets is typically between 100 and 1200 bytes.
2. H.265/HEVC (High-Efficiency Video Coding): Compared to H.264, this recent video compression standard offers improved quality video at lower bitrates. It is made for streaming high-resolution video and is gaining popularity for video over IP. The payload size in a packet, if H.265/HEVC is used for a video call, can also vary depending on the resolution and frame rate of the video, as well as the specific configuration settings used for encoding the video stream. However, the payload size for H.265/HEVC video packets is typically between 100 and 1200 bytes.
3. VP9: This Google-developed video compression standard offers high-quality video at lower

bitrates than H.264. Most contemporary web browsers support it, created to stream video over the internet. The payload size in a packet, if VP9 is used for a video call, can also vary depending on the resolution and frame rate of the video, as well as the specific configuration settings used for encoding the video stream. However, the payload size for VP9 video packets is typically between 100 and 1200 bytes, similar to H.265/HEVC.

4. AV1: Compared to H.264 and VP9, this is a relatively new video compression standard created by the Alliance for Open Media (AOMedia). It offers high-quality video at reduced bitrates. It is designed for video over IP and is gaining popularity. The payload size for AV1 video packets is typically between 100 and 1200 bytes, similar to H.265/HEVC and VP9. The exact size is also determined by the network's maximum transmission unit (MTU).

C

Basic Concepts of 5G Communications and RoHC

C.1. PDU Session

To enable the transport of user data in 5G, a PDU session is a logical connection made between the UE and the 5GC. The PDU session offers a specialized and optimized data stream for particular services or applications, allowing for better user experience and effective use of network resources. Each PDU session has a unique identifier (PDU Session ID) that links it to a particular service or application. The PDU session between the UE and the 5GC can be formed either initially during network registration or dynamically throughout the runtime of the UE [62].

A PDU session can support several QoS flows. UE employs the default QoS flow to exchange data to notify other DN services, such as a voice server, to configure extra QoS flows to manage more demanding traffic. Each PDU session only supports one type of session (IPv4, IPv6, IPv4v6, ethernet, unstructured) [62].

C.1.1. PDU Session in CP/UP Split 5G-RAN Architecture

For the CP/UP split 5G-RAN architecture, Figure 3.14 represents the overall protocol stack in the UP architecture. In 5G Multi-access Edge Computing (MEC), GTP-U messages are transmitted across the N3 interface between UPF and gNodeB-CU-UP and across the F1-U interface between gNodeB-DU and gNodeB-CU-UP [62].

F1-U tunnel refers to a pair of unidirectional GTP-U tunnels used to connect gNodeB-DU to

gNodeB-CU-UP for the UL/DL connection [62]:

- The F1-U UL tunnel transmits UE's UL data from gNodeB-DU to gNodeB-CU-UP over the UL tunnel, which links to gNodeB-CU-UP. It is identified by F1-U UL tunnel information consisting of F1-U UL GTP-U TEID and gNodeB-CU-UP IP address [62].
- The F1-U DL tunnel transmits UE's DL data from gNodeB-CU-UP to gNodeB-DU through the DL tunnel connecting to gNodeB-DU. It is identified by F1-U DL tunnel information comprising F1-U DL GTP-U TEID and gNodeB-DU IP address [62].

In addition, another pair of unidirectional GTP-U tunnels connecting gNodeB-CU-UP to UPF for UL/DL connection is referred to as the N3 tunnel and is described as follows:

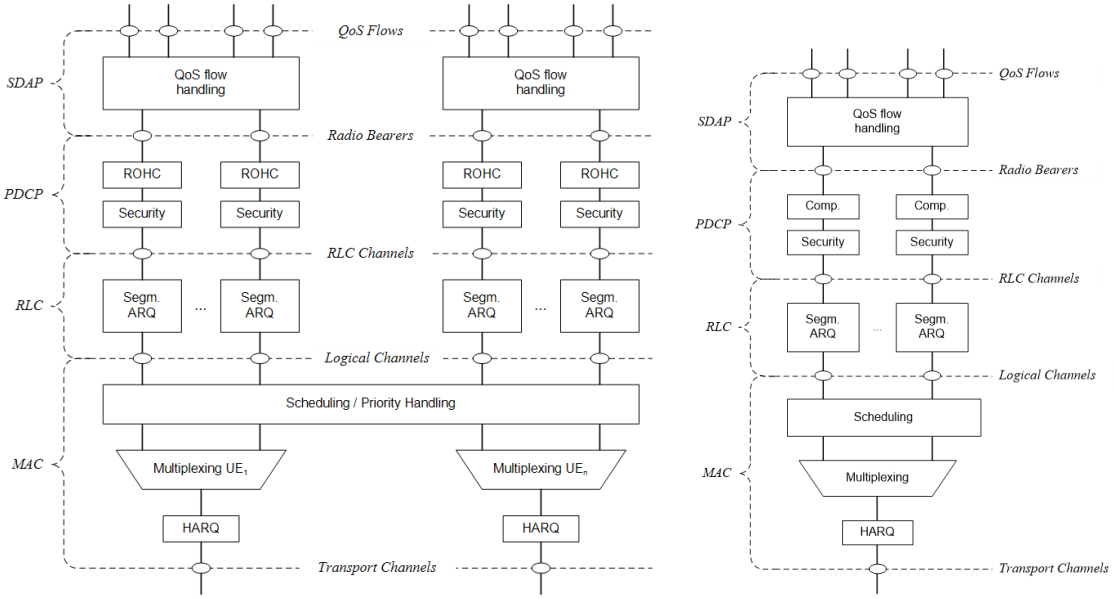
- The N3 UL tunnel (also known as the CN tunnel) is the UL tunnel that links to the UPF and is used to forward UE's UL data from gNodeB-CU-UP to UPF. It is identified by N3 UL tunnel data consisting of N3 UL GTP-U TEID and UPF's IP address [31].
- The N3 DL tunnel (also known as the AN tunnel) is the DL tunnel linking to gNodeB-CU-UP and forward UE's DL data from UPF to gNodeB-CU-UP. It is identifiable by N3 DL tunnel data, which contains N3 DL GTP-U TEID and gNodeB-CU-IP UP's address [31].

C.1.2. QoS Flow

The 5G quality of service paradigm is based on the quality of service Flows. It supports guaranteed flow bit rate quality of service Flows (GBR quality of service Flows) and non-GBR quality of service Flows. Quality of service flow is the finest granularity of quality of service differentiation in a PDU session at the NAS level. Quality of service flow can be identified in a PDU session by its quality of service Flow ID (QFI), carried in the encapsulation header over NG-U [31].

C.1.3. Downlink and Uplink Layer2 Structure

Figure C.1a and Figure C.1b depict the Layer 2 architecture for downlink and uplink QoS flow. Radio bearers are categorized into two groups: DRB for user plane data and SRB for control plane data.



(a) Downlink Layer 2 Structure

(b) Uplink Layer 2 Structure

Figure C.1: Layer 2 Structure [31]

C.1.4. Layer 2 Data Flow

An example of the Layer 2 Data Flow is depicted in Figure C.2. In the downlink, Layer 2 protocols process IP packets received from the application. The packets received by any layer are referred to as SDUs; the receiving layer processes the SDUs, adds the layer's header, and sends them out as PDUs. The IP Packets are initially processed by the SDAP layer, which adds an SDAP header and creates an SDAP PDU. The PDCP layer takes the SDAP PDU, analyses it, and adds a PDCP header, transforming it into PDCP PDUs. RLC and MAC PDUs are also created [31].

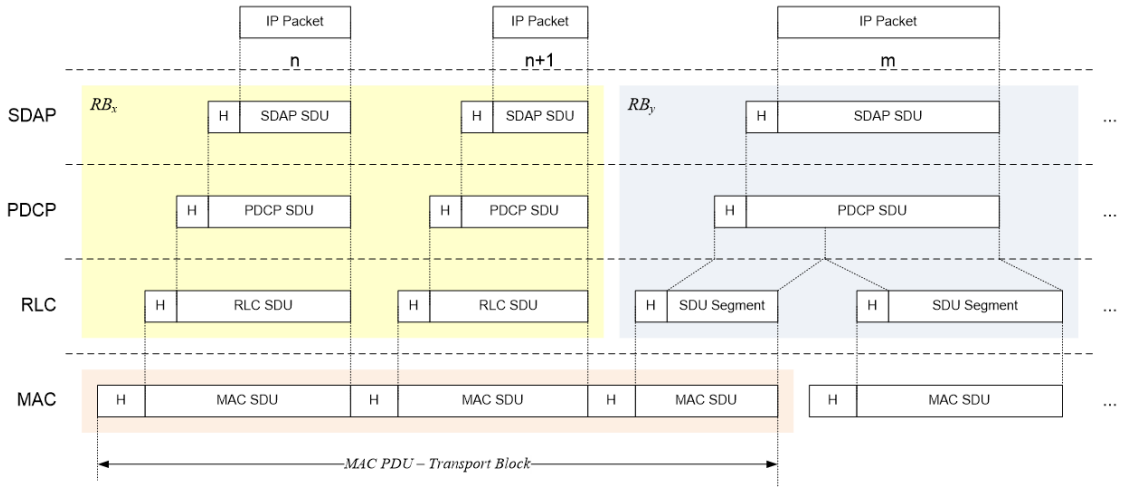


Figure C.2: Data Flow Example [31]

Uplink works similarly, where MAC generates a transport block by concatenating two RLC PDUs from RBx and one from RBy. The two RLC PDUs from RBx (n and n+1) are equivalent to one IP packet, while the RLC PDU from RBy (m) is equivalent to a segment of an IP packet. The letter H represents the headers and subheaders [31].

C.1.5. QoS Architerture

The QoS architecture in NG-RAN, both for NR connected to 5GC and for E-UTRA connected to 5GC, is depicted in Figure C.3.

- For each UE, 5GC establishes one or more PDU sessions at 5G attach.
- For each UE, the NG-RAN establishes at least one DRB concurrently with the PDU session, and additional DRB(s) for QoS flow(s) of that PDU session can be established subsequently (it is up to NG-RAN when to do so).
- The NG-RAN maps packets belonging to different PDU sessions to different DRBs.
- NAS level packet filters in the UE and the 5GC associate UL and DL packets with QoS Flows.

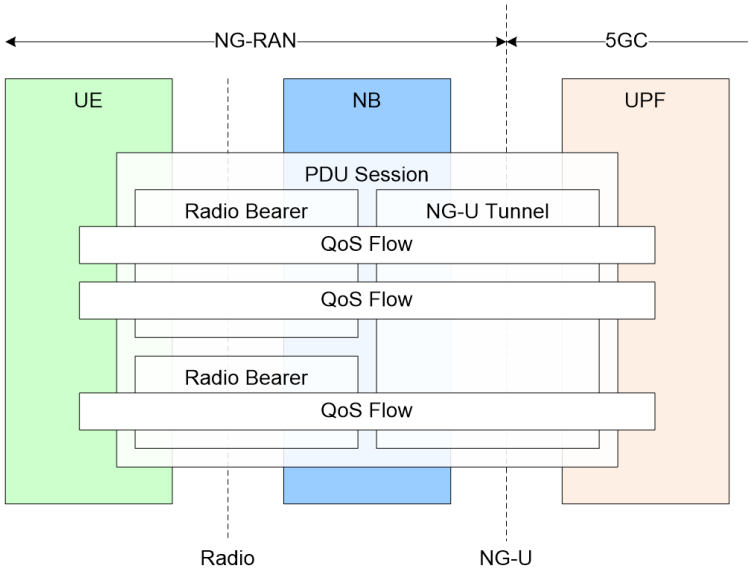


Figure C.3: QoS Architecture and Data Flow [31]

By mapping packets to appropriate QoS flows and DRBs, NG-RAN, and 5GC ensure the quality of service (e.g., reliability and target latency). IP flows are mapped to QoS flows (NAS) and DRBs (Access Stratum) in two steps [31].

A QoS flow is characterized at the NAS level by a QoS profile provided by 5GC to the NG-RAN and QoS rule(s) issued by 5GC to the UE. NG-RAN uses the QoS profile to define the treatment on the radio interface, while the QoS rules determine the mapping between uplink user plane traffic

and QoS flows to the UE. Depending on the profile of a QoS flow, it may be GBR or non-GBR. A QoS flow's QoS profile contains QoS parameters, for instance, the 5G QoS Identifier (5QI), the Allocation and Retention Priority (ARP), the Guaranteed Flow Bit Rate (GFBR), etc. [31].

The NG-RAN maps QoS flows to DRBs based on the QFI and the related QoS profiles, which include QoS parameters and characteristics. DRBs are responsible for defining the treatment of packets on the radio interface. NG-RAN determines whether separate DRBs or multiplexing within the same DRB are required for distinct packet forwarding treatment. This mapping ensures that QoS flows receive the appropriate packet handling and forwarding within the 5G network architecture [31, 32].

The illustration of NAS and AS mentioned above is shown in Figure C.4 and Figure C.5. The PDU session Resource service is offered from the Service Access Point (SAP) to SAP by the Access Stratum. It also shows the protocols on the Uu and NG interfaces linked to provide this PDU session Resource service.

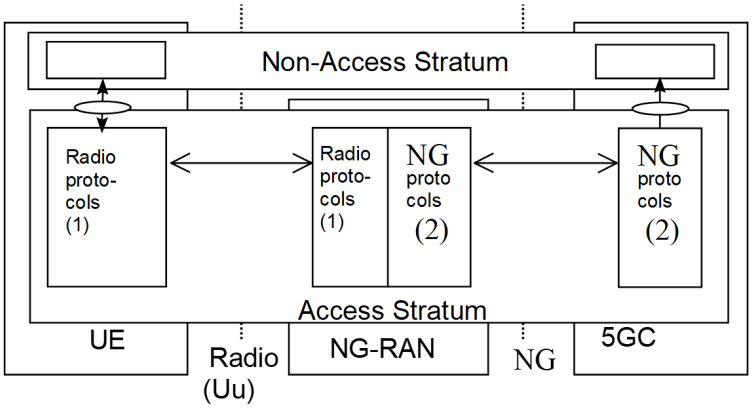


Figure C.4: Non-Access Stratum and Access Stratum Architecture of User Plane [32]

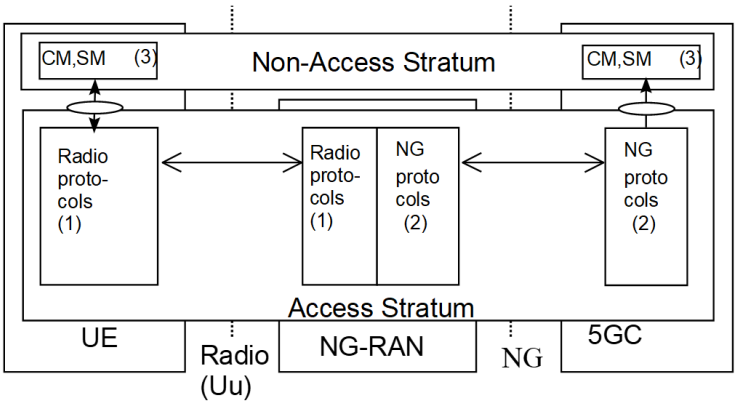


Figure C.5: Non-Access Stratum and Access Stratum Architecture of Control Plane [32]

C.2. RoHC Packets

The overall content in this section is quoted from IETF RFC 3095 [26]. The design of RoHC packets is illustrated in Figure C.6. The padding is optional, and its length is flexible. Either Feedback or a Header must be present. Feedback components always start with a sign of the packet type and carry information about the internal CID. The Header field is loaded with an IR/ DYNAMIC or profile-specific header [26].

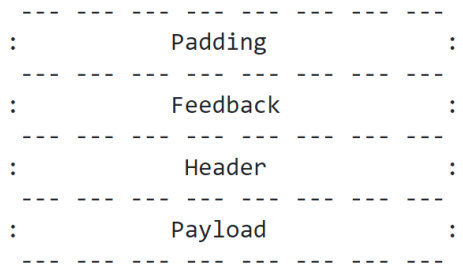


Figure C.6: Structure of RoHC Packets [26]

The header has three options [26]:

1. Does not carry CID information (indicating CID zero);
2. Includes one Add-CID Octet ;
3. Contains embedded CID information of length one or two octets.

Option 1 and 2 only apply to compressed headers in channels with a small CID space. Option 3 only applies to compressed channel headers with a large CID space. The padding octet and the Add-CID Octet (placed before the original CID Octet for large CID requirements) are shown in Figure C.7 and Figure C.8, respectively.

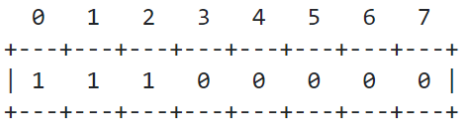


Figure C.7: Padding Octet [26]

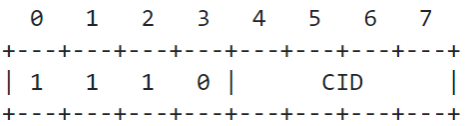


Figure C.8: Add-CID Octet [26]

The Padding Octet can also be viewed as an Add-CID octet for CID 0. The general Header is shown in Figure C.9. The header field begins with a packet-type indicator or follows an Add-CID

Octet with a packet-type indication. (In the diagram, colons ":" indicate that the part is optional, and slashes "/" indicate variable length) [26].

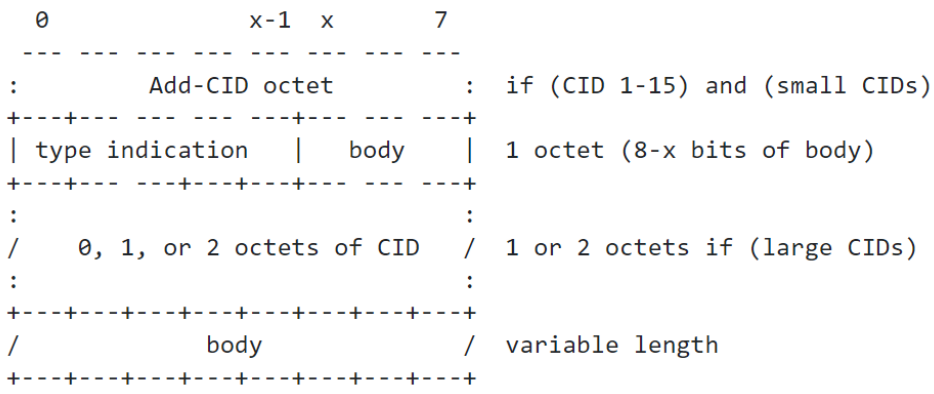


Figure C.9: General Header [26]

C.2.1. RoHC Feedback Type

Feedback transmits data from the decompressor to the compressor. The following primary feedback types are supported: Besides the type of feedback, the profile-specific feedback information may contain additional information [26].

- ACK: It confirms the successful decompression of a packet, which indicates that the context of the depressor is correct.
- NACK: It indicates that the decompressor's dynamic context is out of sync. Generated when multiple consecutive packets cannot be decompressed correctly.
- STATIC-NACK: It indicates that the decompressor's static context is invalid or has not been set up.

C.2.2. RoHC Feedback Format

Feedback provided over a ROHC channel consists of one or more concatenated feedback items, with the following syntax in Figure C.10a for each element:

Where:

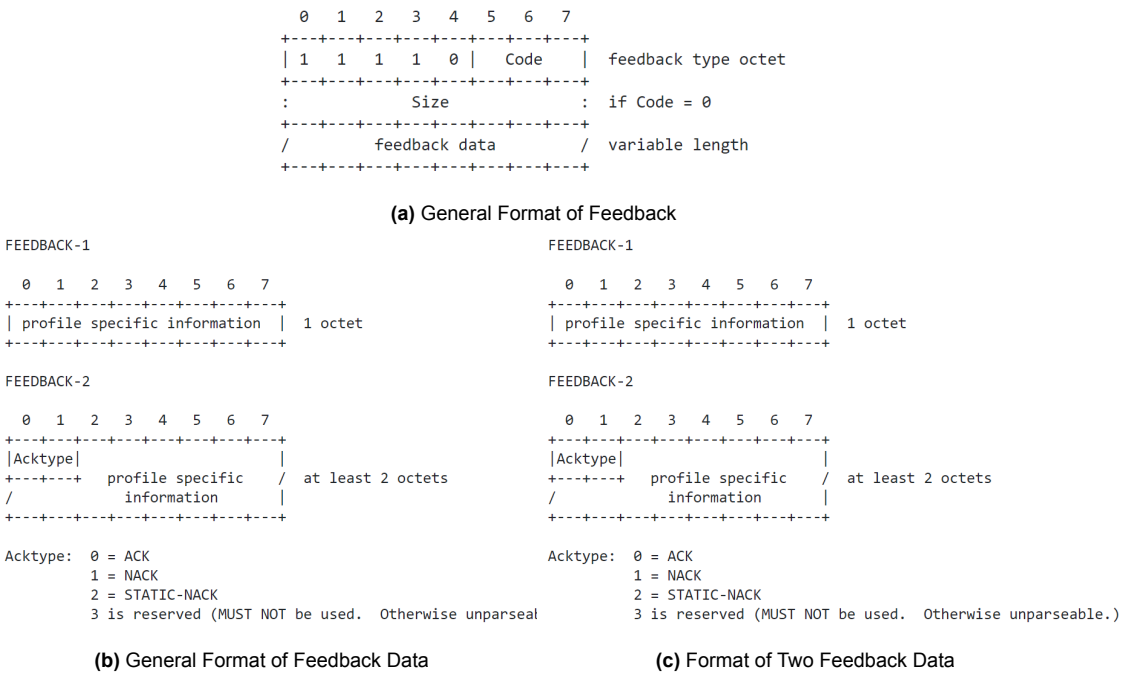
- Code: 0 denotes the presence of a Size octet. 1-7 specifies the feedback data field's size in octets.
- Size: Octet that indicates, in octets, the size of the feedback data field.
- Feedback Data: Specific profile-based feedback information containing CID information

Upon reception by the decompressor, the entire feedback data field is determined by examining

the code field and perhaps the size field. This exact information length enables piggybacking and transmitting several feedback elements per packet [26].

Once the decompressor has calculated the size of the feedback data field, it eliminates the feedback type octet and the size field (if present). It passes the remaining data and a size indicator to the same-side associated compressor. The structure of the feedback data received by the compressor is as follows in Figure C.10b.

The feedback field has two following formats, shown in Figure C.10c. The compressor can parse the feedback field using the following logic:



(a) General Format of Feedback

(b) General Format of Feedback Data

(c) Format of Two Feedback Data

Figure C.10: Format of Feedback [26]

1. The feedback for large CIDs will always begin with a CID encoded. If the first bit is 0, the CID consists of a single octet. Whenever the initial bit is 1, a CID consists of two octets;
2. If the size of a small CID is one octet, the feedback is FEEDBACK-1;
3. Whenever the size of a small CID is greater than one octet and the feedback begins with the bits 11, the feedback begins with an Add-CID octet. If the size is 2, the next value is FEEDBACK-1. If the size is greater than 2, FEEDBACK-2 follows Add-CID [26];
4. Otherwise, the feedback begins with a FEEDBACK-2, and there is no Add-CID octet.

C.2.3. Packet Formats

The following notation is employed in this section [26]:

- bits(X): The number of bits present in the compressed header for field X. (including exten-

sion).

- field(X): The value of field X in the compressed header.
- context(X): The value of field X as established in the context.
- value(X) = field(X) if X is present in the compressed header; = context(X) otherwise.
- hdr(X) = the value of field X in the uncompressed or decompressed header.
- Updating Properties: Lists the context fields immediately modified by processing the compressed header. There may be dependent fields that are implicitly changed as well (for example, an update to context(SN) often changes context(TS)).
- SN: The compressed RTP Sequence Number.
- IP-ID: The compressed IP-ID field for the innermost IP header or outer IP header.
- TS: The compressed RTP Timestamp value.
- CRC: The CRC over the uncompressed, original header.
- M: RTP Marker bit.

Figure C.11 is the usual format for a compressed RTP header. Note that the order of the fields following the extension is identical to those in an uncompressed header.

The value of the UDP Checksum in the context determines whether or not the UDP Checksum field is present. The UDP Checksum is enabled and delivered with each packet if this value is not zero. If 0 is returned, the UDP Checksum is deactivated and not sent. The header cannot be compressed when context (UDP Checksum) is zero and hdr (UDP Checksum) is nonzero. It must be transmitted uncompressed, or the context must be reset via an IR packet. Context(UDP Checksum) is only modified by IR or IR-DYN headers, never by type 2, 1, or 0 headers [26].

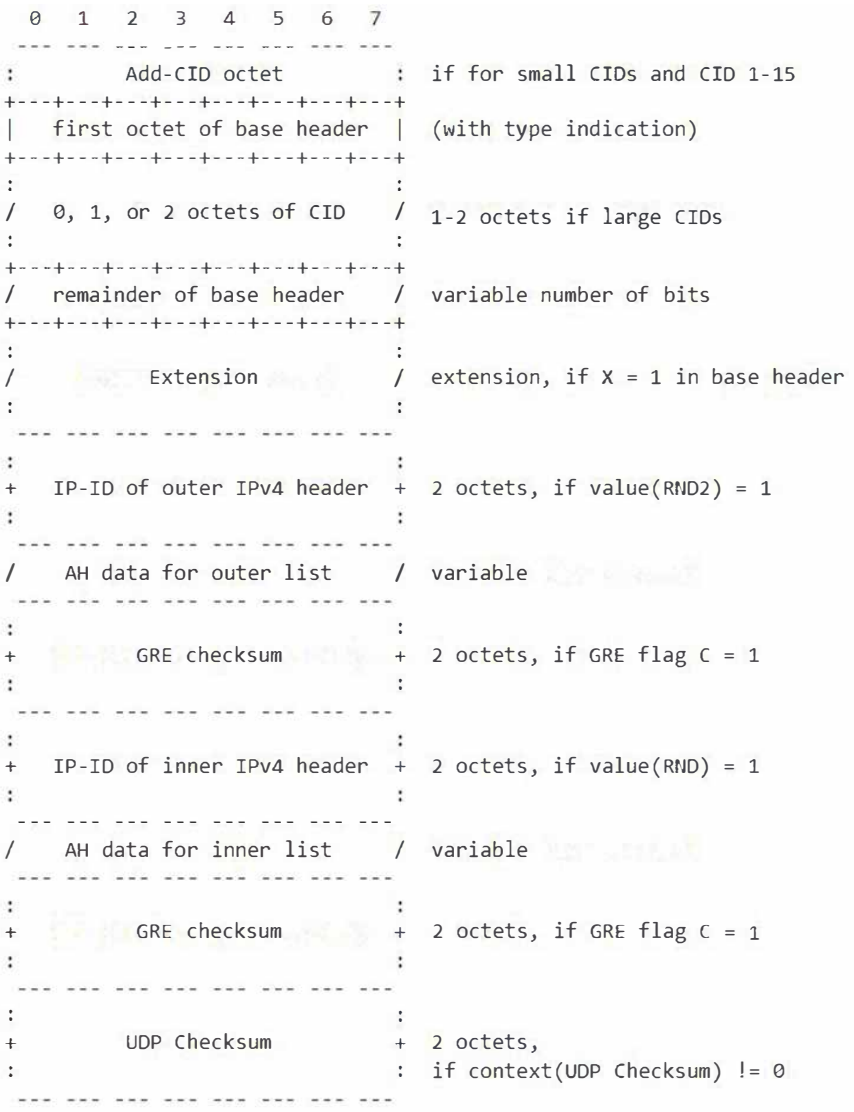
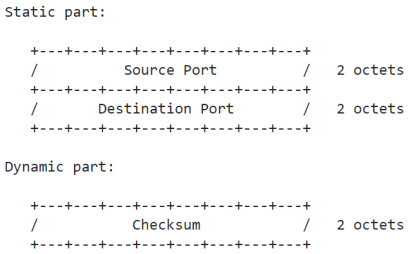


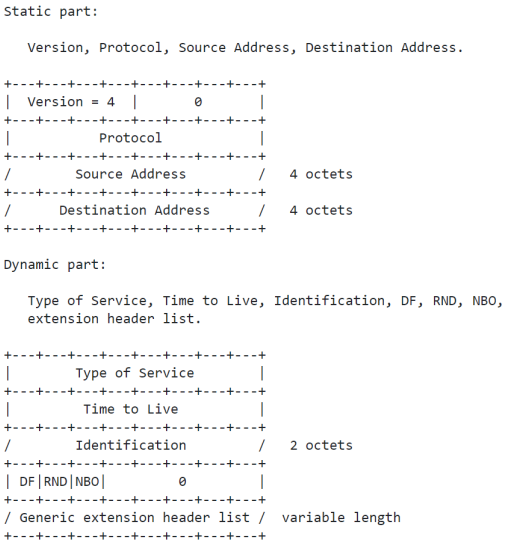
Figure C.11: The General Format For a Compressed RTP Header [26]

C.2.4. Initialization of IPv4 Header

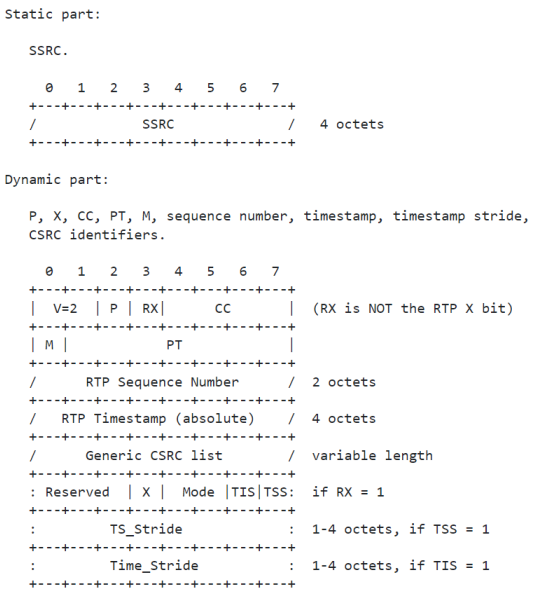
The static and dynamic part of the IPv4 header in compressor/ decompressor context is shown in Figure C.12b. The corresponding UDP header and RTP header in context are shown in Figure C.12a and Figure C.12c.



(a) UDP in Context



(b) IPv4 in Context



(c) RTP in Context

Figure C.12: UDP, IPv4, RTP in Compressor/ Decompressor Context [26]