FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Robust Header Compression over IEEE 802 Networks

Ana Raquel Silva Faria

Dissertação submetida para satisfação parcial dos
requisitos do grau de mestre
em
Redes e Serviços de Comunicação

Dissertação realizada sob a supervisão do
Professor Doutor Manuel Alberto Pereira Ricardo, do Departamento de
Engenharia Electrotécnica e de Computadores da Faculdade de
Engenharia da Universidade do Porto e com co-orientação do Mestre
António Pedro Freitas Fortuna dos Santos, do INESC Porto

Porto, Setembro 2009

Resumo

A utilização eficiente da largura de banda é um requisito dos sistemas de redes móveis e sem fios. Há uma série de factores que afectam a eficiência de transmissão de tráfego IP. Os cabeçalhos de pacotes IP podem levar ao consumo de uma parte significativa da largura de banda disponível, especialmente quando o tráfego IP transporta pacotes pequenos, tais como os pacotes de *Voz sobre IP* (VoIP). Uma forma de ultrapassar este problema é a utilização de sistemas de compressão de cabeçalhos como por exemplo o *Robust Header Compression* (RoHC).

Vários estudos foram realizados sobre o desempenho e ganho do RoHC U-mode aplicado ao tráfego VoIP. Estes estudos foram baseados em simulações do RoHC U-mode. Nesta dissertação, todos os testes efectuados tiveram por base uma implementação do RoHC U-mode. Para alcançar este objectivo foi desenvolvido um protótipo do RoHC U-mode desenvolvido para Linux, capaz de funcionar sobre as redes IEEE 802.3 e IEEE 802.11. Este protótipo, é focado no perfil 0x0001, para o envio de pacotes comprimidos de IP/ UDP / RTP. Os testes realizados com o protótipo RoHC mostraram que a aplicação de RoHC a tráfego VoIP, reduz *bit-rate* em 43% nas redes IEEE 802.3 e 72% nas redes IEEE 802.11.

Abstract

The efficient use of bandwidth is one of the main challenges of mobile and wireless networking systems. There is a number of factors that affect the transmission efficiency of IP traffic. The IP packet headers may consume a significant part of the available bandwidth, especially when the IP traffic transports small payloads such as Voice over IP (VoIP). In order to minimize this problem, IP *Header Compression* (HC) techniques were introduced. The *Robust Header Compression* (RoHC) protocol is the state of the art in IP HC.

Multiple studies were conducted on the performance and gain of using the RoHC U-mode applied to VoIP traffic. These studies were based on simulations of the RoHC U-mode. In this dissertation, all tests conducted were base on an implementation of the RoHC U-mode. To achieve this goal a prototype of the RoHC U-mode for Linux was developed, in user space, that supports the transport of RoHC packets over IEEE 802.3 and IEEE 802.11 networks. The prototype only supports the header compression of IP/UDP/RTP packets. The prototype was used to study the performance of RoHC applied to VoIP traffic over IEEE 802.3 and IEEE 802.11 networks. The tests performed showed that the application of RoHC reduced the VoIP bit-rate in 43% in IEEE 802.3 network and 72% in the IEEE 802.11 network.

Acknowledgments

There are a number of people that directly or indirect helped in the elaboration of the dissertation to whom I would like to express my gratitude.

First, I would like to, my supervisors, Prof. Manuel Ricardo and Pedro Fortuna for their guidance and patience throughout the course of this work.

Thanks also to the Department of Computer Science of the Instituto Superior de Engenharia do Porto and especially to its technical support team, Nuno Fonseca, Pedro Rocha, Jaime Neto, and also to Ana Costa, Joaquim Santos, and Bruno Silva for their incentive.

My final thanks goes to my family, especially to my sister Ana Paula Silva Faria for their support and understanding, without which this work could not be possible.

Table of Contents

Figures Index

Tables Index

Abbreviations and Acronyms

AVC - Advance Video Coding

CID - Context Identifier.

CBR – Constant Bit-rate

CRC - Cyclic Redundancy Check. Error detection mechanism.

CRTP - Compressed RTP.

cTCP - Compressed TCP. Also called VJ header compression.

DF - Don't Fragment bit

ESP - Encapsulating Security Payload

FC - Full Context state (*Decompressor*).

FO - First Order state (*Compressor*).

GPRS - General Packet Radio Service

GSM - Global System for Mobile Communications

HC - Header Compression.

IANA - Internet Assigned Numbers Authority

IETF - Internet Engineering Task Force

IHL - IP header length

IPHC - IP Header Compression.

IR - Initiation and Refresh state (Compressor). Also an IR packet.

IR-DYN- IR-DYN packet.

LSB - Least Significant Bits.

MRRU - Maximum Reconstructed Reception Unit.

MTU - Maximum Transmission Unit.

MSB - Most Significant Bits.

NAL –Network Abstraction Layer

NALU –Network Abstraction Layer unit

NBO - Flag indicating whether the IP-ID is in Network Byte Order.

NC - No Context state (*Decompressor*).

O-mode - Bidirectional Optimistic mode.

PDU – Protocol Data Unit

PHS - Payload Header Suppression

PPP - Point-to-Point Protocol.

R-mode - Bidirectional Reliable mode.

RND - Flag indicating whether the IP-ID behaves randomly.

RoHC - Robust Header Compression.

RTCP - Real-Time Control Protocol.

RTP - Real-Time Protocol.

RTT - Round Trip Time.

SC - Static Context state (*Decompressor*)

SDU - Service Data Unit.

SN - RTP Sequence Number.

SO - Second Order state (*Compressor*).

SSRC - Sending source. Field in RTP header.

CSRC - Contributing source. Optional list of CSRCs in RTP header.

TCP - Transmission Control Protocol

TOS - Type of Service

TTL - Time to Live

TS - RTP Timestamp.

RLC - Radio Link Control protocol

RoHC –Robust header Compression

UDP - User Datagram Protocol

U-mode - Unidirectional mode.

UMTS - Universal Mobile Telecommunications System

W-LSB - Window based LSB encoding.

VoIP – Voice over IP

WSN - Wireless Sensor Network

WMSN - Wireless Multimedia Sensor Networks.

3GPP - 3rd Generation Partnership Project

3GPP2 - 3rd Generation Partnership Project 2

1 Introduction

The efficient use of bandwidth is one of the challenges of mobile and wireless networking systems today. This aspect combined with the continuous expansion of mobile networks, paves the way to the development of mechanisms to help improve the wireless networks. Those mechanisms try to use efficiently the limited bandwidth available. The use of the IP protocol by these networks introduces a significant overhead. The IP packet headers may consume a significant part of the available bandwidth, thus reducing the transmission efficiency in the medium. In order to minimize this problem, IP *Header Compression* (HC) techniques were developed. HC consists mainly in the suppression of redundant information from the IP headers in order to save bandwidth. The use of HC can be beneficial because there is significant redundancy between the header fields within the same IP packet and between different IP packet headers of the same stream.

The inception of the use of HC schemes was in 1984 with the Thin-wire protocol [2]. This protocol has low compression rates. The 40 bytes IPv4/TCP headers were compressed to 13 bytes. Van Jacobson later on proposed *TCP header compression* cTCP [3], a protocol based on the header redundancy information as its predecessor, but with higher compression rates. The cTCP protocol compressed the 40 bytes of the TCP/IPv4 header to 3–6 bytes. There were other propositions for the different protocol headers based on redundancy, such as the cRTP. This header compression mechanism had a detailed specification for the RTP protocol. The HC protocol described above does not perform well over links with large and varying bit error rates and large round trip times [21].

The *Robust Header Compression* (RoHC) [1] was developed to address the limitations presented by the previous HC protocols and to improve the performance of transmissions. The RoHC [1] protocol is useful for flows with small size packets such as *Voice over IP* (VoIP) that can be responsible for large overheads. *Internet Engineering Task Force* (IETF) defined mechanisms for RoHC over *Point-to-Point Protocol* (PPP) and later 3GPP as integrated the use of RoHC in networks such as GPRS/UMTS. There was an initial attempt to draft a specification of RoHC over IEEE 802 networks, but it was given up. This dissertation work is centred on this subject.

1.1 Objective

The goal of this dissertation was to implement and study the performance of the RoHC U-mode when applied to VoIP traffic over IEEE 802.3 and IEEE 802.11 networks. A prototype was developed in Linux, in user-space, using Packet Sockets.

1.2 Results

The main contribution of this dissertation is a prototype of the RoHC U-mode developed for Linux, in user space, able to operate over IEEE 802.3 and IEEE 802.11 networks. This prototype, focus on the Profile 0x0001, for sending compressed IP/UDP/RTP packets.

1.3 Organization of the dissertation

This dissertation is organised as follows. Chapter 2 describes the state of the art in IP header Compression schemes, including a detailed description of the RoHC Protocol [1]. Chapter 3 addresses the related work that studies the performance of RoHC over a number of different network technologies. Chapter 4 describes the requirements, the software architecture and tools used to develop the RoHC U-mode implementation presented by this dissertation. Chapter 5 describes the functional tests used to validate the RoHC U-mode implementation, presents the performance tests and discusses the result obtained. Chapter 6 concludes the dissertation, describing the main results and contributions, and point out directions for future work.

2 Header Compression

The problem associated with the transmission of IP packets is the overhead generated by the packets headers, which may require a significant part of the available bandwidth, thus reducing the link efficiency. In order to minimize this problem, HC techniques were developed. HC consists mainly in the suppression of information from the headers in order to save bandwidth. The reason why the use of HC can be beneficial is that there is significant redundancy between the header fields within the same IP packet and between different IP packet headers of the same stream.

HC was first explored in 1984 by the Thinwire-II protocol [2] and to apply to the Internet connection of personal computers. This protocol has low compression rates. The 40 bytes IPv4/TCP headers were compressed to 13 bytes. In 1990, Van Jacobson proposed a *TCP header compression* scheme (cTCP) [3]. The cTCP is a scheme for compressing the headers of TCP traffic, which offers better compression rates than its predecessor. A 40 byte of an IPv4/TCP header is compressed to 3 bytes. Later the *IP Header Compression* (IPHC) [20] scheme was proposed. This scheme introduced some improvements on the cTCP and support for compressing other types of headers besides the IP headers, such as UDP and RTP headers. IPHC can compress the UDP and TCP headers to 4 -7 bytes. The *Compressed RTP* (cRTP) [19] was specified for the compression IP/UDP/RTP headers. The cRTP can compress a RTP header to a minimum of 2 bytes. In [21], it is shown that this HC scheme does not perform well over links with large round trip times and consecutive packets loss, making this protocol unsuitable for IP telephony over wireless links.

The HC protocols described above do not perform well over links with large and varying bit error rates and large round trip times. Moreover, they do not take in consideration that some applications may take advantage of packets that are delivered with errors, such as Video Streaming. In order to address the limitations of those HC protocols and to improve the header compression ratio of the transmissions, a new protocol was specified by IETF, the RoHC [1].

2.1 Early Header Compression schemes

The use of HC techniques were first explored in 1984 Thinwire-II protocol [2]. This technique explored the field similarities between the consecutive headers of a same TCP stream. The transmitting and receiving hosts, which in a HC scenario are known as *Compressor* and *Decompressor* respectively, cache information about packet stream. This information is stored locally in the host a *Context* per HC stream. After an initial *Context* setup, in which packets are sent with full headers, the *Compressor* is able to compress header packets based on the information kept in the stream's HC *Context*. The *Decompressor* reconstructs the original headers using the information of the *Context* stored locally. HC packets carry a *Context Identifier* (CID) that indicates the HC stream, and which *Context* to use. To guarantee a successful header decompression, both the *Compressor* and *Decompressor Contexts* need to be synchronised. The information stored in the *Context* is updated every time a new packet is sent. The Thinwire–II protocol offers low compression rates. An IPv4/TCP header with 40 bytes is usually compressed to 13 bytes.

Van Jacobson proposed cTCP [3] protocol in 1990. Although being similar to the Thinwire-II protocol, provides higher compression ratios. A 40 bytes IPv4/TCP header is usually compressed to 3 bytes. The cTCP encodes and transmits the difference between the consecutive headers. Uncompressed headers are reconstructed by applying the differences transmitted in a compressed header to the previous transmitted packet. However, if a packet is lost or corrupted, the following header could be reconstructed with errors. These errors can be detected by using the TCP checksum. The *Decompressor* discards all packets having incorrect TCP checksums. This protocol also introduces an error recovery mechanism in the form of TCP acknowledgments, which enables the *Compressor* to detect that a packet was not delivered. To synchronize the *Context* in the two hosts, the *Compressor* would send a packet with full headers.

The IPHC [20] protocol introduced some improvements when compared with cTCP [3] protocol. The IPHC can compress an UDP and TCP headers to 4 -7 bytes. This protocol introduces the possibility of compression of other type of headers besides the IP headers (IPv4 and IPv6), such as UDP, ESP and RTP [22]. When compressing non-TCP headers it does not send the difference between the packets. If there are changes in the field's headers the *Compressor* sends a packet with full headers, in order to create / update a *Context* on the *Decompressor* side. This helps preventing an incorrectly decompression due to an invalid *Context*. The *Compressor* sends periodically an uncompressed packet to update the *Context* created, enabling the *Decompressor* to recover from error situations. After a change to the *Context*, the *Compressor* uses a slow start mechanism in which a non-TCP full header packet is sent periodically, with an exponentially increasing interval. IPHC uses the same error recovery mechanisms of cTCP protocol and two additional mechanisms:

- A mechanism called TWICE, which enables the *Decompressor* to repair the *Context* information locally. This is achieved by estimating the changes to the *Context* if the previous packet headers have been reconstructed correctly. Assuming the headers change in a constant fashion, the changes are applied twice. If the TCP checksum is correct, then the reconstructed headers are considered correct.
- The *Decompressor* sends negative acknowledgment (NACK) when it fails to repair the *Context*, it requests full header packet from the *Compressor*.

The cRTP [19] was proposed to compress the headers of IP/UDP/RTP packets. The cRTP is able to compress an RTP header to a minimum of 2 bytes. Because of the use of UDP/RTP protocols, mechanisms of error detection must be different from the ones used in cTCP. Therefore, in cRTP the *Decompressor* uses signalling messages to inform the *Compressor* that its *Context* is not synchronized. To avoid packet loss due to unsynchronized *Context*, the *Decompressor* tries to correct the *Context* locally with the TWICE mechanism. Each cRTP packet transports a sequence number that is incremented every time a packet is send by cRTP. The sequence number allows the *Decompressor* to detect if a packet was lost. If the attempt to decompress the packet is successful, the packet is delivered to the application. The problems of using the TWICE mechanism can be divided in two major problems. Firstly, because of the use of UDP, checksum is mandatory; the minimal header size is four octets. Most part of the voice codecs does not tolerate errors. Those codec's activate UDP checksum, because they do not want the damage packets to be delivered. Secondly, losses on the RTP stream, that occurs prior to the point of compression, result irregular updates to header cRTP. The simples version of TWICE does not work very well, and the improved versions need a larger number of attempts. Later a study [21] showed that this HC scheme did not perform well over links with large round trip times and consecutive packets lost, making this protocol not suitable for IP telephony over wireless links.

2.2 RoHC

Robust Header Compression (RoHC) [1] is a header compression framework being specified by the IETF RoHC Working Group. The framework provides compression directives for the protocol headers to be compressed, protocol headers such as IP/UDP/RTP, IP/UDP, IP/ESP. These specifications are called *Profiles*.

The RoHC protocol is split over two instance, which compose a *Node*: the *RoHC Compressor*, responsible for the compression of the packets headers; and the *RoHC Decompressor*, responsible for the decompression of the packets headers.

Each *Node* is connected to another thru RoHC *Channels*, where the compress packets flow. It is possible to have several channels with different characteristics, such as, bandwidth, error rate connecting the same pair of *Nodes*.

The RoHC scheme has three modes of operation: the *Unidirectional Mode* (U-mode), the *Bidirectional Optimistic* (O-mode), and the *Bidirectional Reliable mode* (R-mode). The *Compressor* always starts in the U-mode and its transition to any bidirectional modes can happen as soon as a packet has reached the *Decompressor* and a feedback packet is sent back. To determine which mode to operate RoHC depends on the environment characteristics such as header size, feedback error probabilities.

Both the *Compressor* and the *Decompressor* can be regarded as states machines. The *Compressor* operates in three states: *Initialization and Refresh* (IR), *First Order* (FO), and *Second Order* (SO). The *Decompressor* also operates in three states: *No Context* (NO), S*tatic Context* (SC), and *Full Context* (FC). The *Compressor* starts in the IR state and it can change to the other states, when some of the RoHC parameters are meet, such as the number of packets sent in the IR state. The *Decompressor* starts in NO state and it can change to another state, as soon it is decompress successfully a compressed packet.

The *Compressor* starts by sending static information, and then suppresses this information. The *Decompressor* uses techniques like dependencies and predictability to discover the suppressed fields and reconstruct the original header. This way the compressed packets header size sent from the *Compressor* to the *Decompressor*, can be reduced significantly. All the relevant information of the past packets headers is kept in a header compression *Context*. This information contains the latest updates to the original header and the redundant information about the compressed stream. The *Context* is used to compress e decompress the packet headers, so it is kept both in the *Compressor* and in the *Decompressor*. Every time the fields of the headers of a stream change, the *Context* must be updated. *Context* is created for each header-compressed stream in a RoHC C*hannel*. The CID identifies a RoHC *Context* created.

2.2.1 RoHC Profiles

The header compression profile gives the definition of how to compress the protocol headers of a packet stream over a certain link. This specification by protocol allows the RoHC protocol to be robust and efficient. The profiles defined in the RoHC Working Group and listed in [24] are shown in Table I:

Profile Identifier	**Usage**
0x0000	ROHC uncompressed [24]
0xnn00	Reserved
0x0001	ROHC RTP [1]
0x0101	ROHCv2 RTP [23]
0xnn01	Reserved
0x0002	ROHC UDP [1]
0x0102	ROHCv2 UDP [23]
0xnn02	Reserved
0x0003	ROHC ESP [1]
0x0103	ROHCv2 ESP [23]
0xnn03	Reserved
0x0004	ROHC IP [25]
0x0104	ROHCv2 IP [23]
0xnn04	Reserved
0x0005	ROHC LLA [26]
0x0105	ROHC LLA with R-mode [27]
0xnn05	Reserved
0x0006	ROHC TCP [28]
0xnn06	Reserved
0x0007	ROHC RTP/UDP-Lite [29]
0x0107	ROHCv2 RTP/UDP-Lite [23]
0xnn07	Reserved
0x0008	ROHC UDP-Lite [29]
0x0108	ROHCv2 UDP-Lite [23]
0xnn08	Reserved
0x0009-0xnn7F	To be Assigned by the *Internet Assigned Numbers Authority* (IANA)
0xnn80-0xnnFE	To be Assigned by IANA
0xnnFF	Reserved

Table I – Profile Identifiers

2.2.2 Negotiation

Prior to the exchange of header-compressed packets, RoHC requires a negotiation phase. The *Compressor* and the *Decompressor* gain knowledge about the different characteristics of the link and about header compression parameters that will be use. With this negotiation is established the *RoHC Channel*.

The parameters establish in this phase and referred in [1] are:

- MAX_CID: Highest CID number used by the *Compressor*;
- LARGE_CIDS: if it has a false value, the short CID is used the CID value can go from 0 to 15 occupying from 0 to 1 byte). On the other hand if it has a true value, the embedded CID representation is used (the CID value can go from 0 to 16363 occupying from 1 to 2 bytes) ;
- PROFILES: Indicating the profile or profiles supported by the *Decompressor.* The *Compressor* cannot compresses the packets using a profile that is not indicated in the parameter;
- FEEDBACK_FOR: If provided, this parameter indicates that feedback sent over a certain channel;
- MRRU: Maximum reconstructed reception unit. This is the size of the largest reconstructed unit in octets that the *Decompressor* is expected to reassemble from segments.

2.2.3 Modes of operation

2.2.3.1 Unidirectional Mode

In the U-mode, RoHC packets are sent, in one direction only, from the *Compressor* to the *Decompressor*. There are no feedback packets in the opposite direction. This enables the use of RoHC over links where a return path from *Decompressor* to *Compressor* is unavailable or undesirable. In the U-mode, the transitions between the *Compressor's* states are performed when periodic timeouts expire, or when irregularities in the header fields compressed packet stream are detected. Due to the periodic refreshes that update the *Context* in the *Decompressor*, and the lack of feedback for initiation of error recovery, the compression in the U-mode will be less efficient due to the higher probability of loss propagation, when compared to the bidirectional modes. Compression with RoHC must start in the U-mode. Transition to any of the bidirectional modes can occur as soon as a packet has reached the *Decompressor* and it has replied with a feedback packet indicating that a mode transition is desired [1].

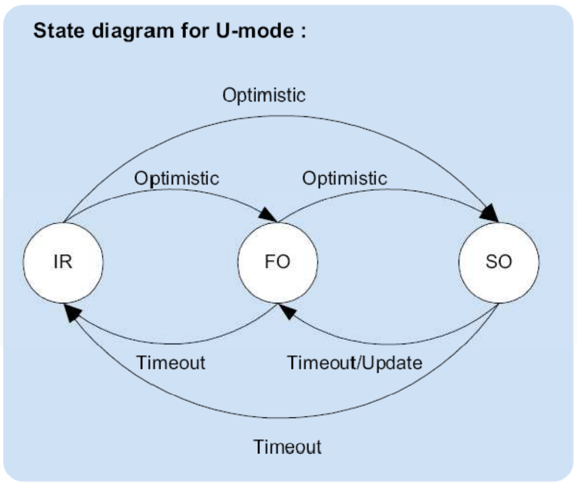


Figure 1- State diagram for U-mode

As shown in Figure 1, the *Compressor* starts in the IR state and uses an optimistic approach to change to other states. It sends a number of IR packets until it is confident that the *Decompressor* has established a *Context* for that stream. It then changes to the FO state and it uses the same approach to change to the SO state. The *Context* updates are sent periodically to ensure the *Compressor* and the *Decompressor Contexts* are synchronized. A periodic time-out mechanism is used to transit to the states of FO and IR. They are two timeouts, one for each state. When they are reached, they force the state machine to go back to FO or IR state. This allows the *Context* to be updated or to recover from errors.

2.2.3.2 Bidirectional Mode

The O-mode is similar to the U-mode. The main difference is that a feedback channel is used to send error recovery requests (NACKs) and, optionally, acknowledgments of significant *Context* updates from the *Decompressor* to the *Compressor*. In the O-mode, periodic refreshes are not needed. The O-mode aims to maximize compression efficiency with sparse usage of the feedback channel. It minimizes the number of damaged headers delivered to the upper layers, due to residual errors or *Context* invalidation [1].

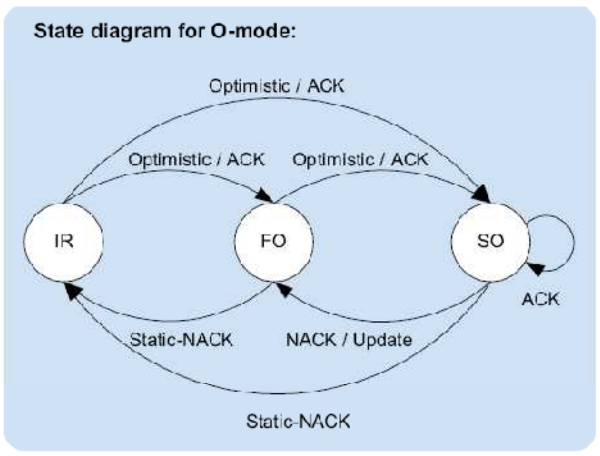


Figure 2- State diagram for O-mode

As shown in Figure 2, the *Compressor* may use an optimistic approach or receive positive acknowledgments (ACKs) from the *Decompressor* to change to FO or SO states. The *Decompressor* sends ACKs for IR packets. For other types of packets, it is optional to send ACKs. To recover from error conditions, it sends NACKs or static NACKs (depending on its state, indicate that the static context is invalid). The state changes of the *Compressor* enter in the optimistic mode upon receiving feedback from the *Decompressor* and using the optimistic approach.

2.2.3.3 Bidirectional Reliable Mode

The R-mode differs in many ways from the U-mode and the O-mode. The most important differences are an intensive usage of the feedback channel and a stricter logic at both the *Compressor* and the *Decompressor* that minimizes the probability of *Context* invalidation. Feedback is sent to acknowledge all *Context* updates, including updates of the SN field. However, not every packet updates the *Context* in R-mode. It may have a lower probability of *Context* invalidation than O-mode, but a larger number of damaged headers may be delivered when the *Context* actually is invalidated. [1]

The frequency of *Context* invalidation may be higher than the R-mode, in particular when long loss/error bursts occur

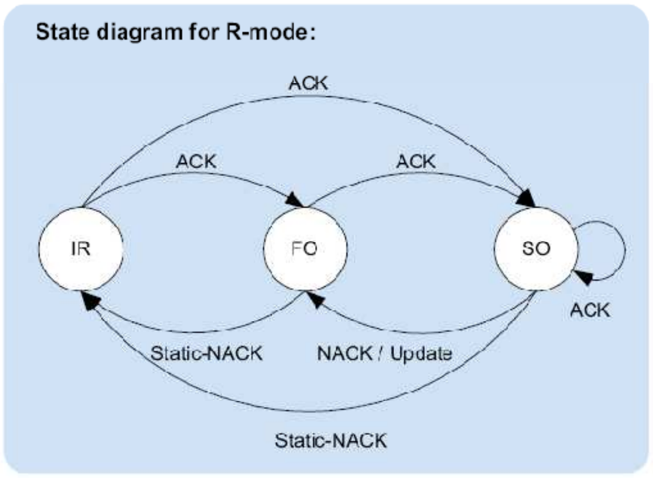


Figure 3- State diagram for R-mode

Figure 3 represents the state diagram for R-mode. Instead of using an optimistic mode, the *Compressor* transitions rely only on the ACKs sent by the *Decompressor*. This will maximize the probability of the *Contexts* ensure that the *Context* staying synchronized. The downward state transitions are very similar to the O-mode. To recover from errors or from unsynchronized *Contexts*, the *Decompressor* sends NACKs or static NACKs (depending on its state).

2.2.4 States

2.2.4.1 Compression states

The *Compressor* has three states: *Initialization and Refresh* (IR), *First Order* (FO), and *Second Order* (SO). If possible, the *Compressor* works in the higher state.

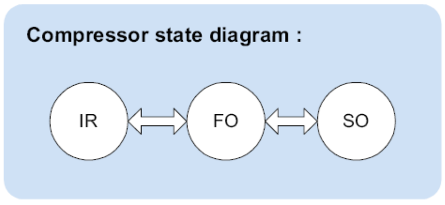


Figure 4- Compression states

The IR state initializes the static parts of the *Context* in the *Decompressor* and recovers from failure by updating the static part of the *Context* in the *Decompressor*. In this state, the *Compressor* sends full header packet. This includes all static and non-static fields, in uncompressed form. The *Compressor* will remain in the IR state until it is confident that the *Decompressor* has received the static information correctly. The RoHC standard [1] does not estipulate values for the level of confidence.

The FO state purpose is to initialize the dynamic part of the *Context* and to communicate irregularities in the packet stream. When operating in this state, the *Compressor* rarely sends all static fields, and the information sent, is usually compressed, at least partially. Only a few static fields in *Decompressor Context* can be updated. The *Compressor* enters in FO state from the IR state or from the SO states, in this last case the headers of the packet stream do not match their previous saved pattern, so the *Compressor* tries to update its *Context* by going down to a prior state. It stays in the FO state until it is confident that the *Decompressor* has acquired all the parameters of the new pattern. Changes in fields that are irregular are communicated in all packets. It is very important to detect corruption of such packets to avoid erroneous updates and inconsistencies in the *Context*. [1]

The SO state is the state where compression is optimal. The *Compressor* enters the SO state when the header to be compressed is completely predictable, and the *Compressor* is sufficiently confident that the *Decompressor* has acquired all parameters of the functions from RTP Sequence Number (SN) to other fields. Correct decompression of packets sent in the SO state only hinges on correct decompression of the SN. However, successful decompression also requires that the *Decompressor* has efficiently received the information sent in the preceding FO and IR state packets. The *Compressor* leaves this state and goes back to the FO state when the header no longer conforms to the uniform pattern and cannot be independently compressed based on previous *Context* information. In addition, if the *Compressor* has reached the limit of one of the timeouts, it forces to go to the previous state [1]. The *Compressor* decides which state will transit to base on the variations in the packet headers, the feedback from the *Decompressor* and periodic timeouts. Periodic timeouts are used when RoHC is operating in a U-mode or when feedback is not present.

2.2.4.2 Decompression states

There are three states in the *Decompressor*: *No Context* (NO), *Static Context* (SC), and *Full Context* (FC).

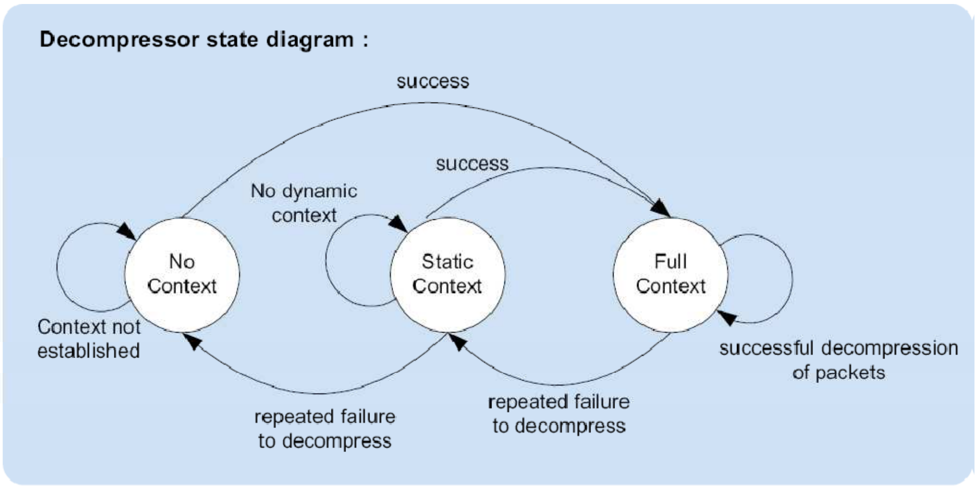

Decompressor state diagram :
success
success
No dynamic context
No Context
Static Context
Full Context
Context not established
successful decompression of packets
repeated failure to decompress
repeated failure to decompress

Figure 5- Compression states

The NO state the *Decompressor* has no *Context*, so in order to build one it can only receive IR-Packets. All other packets types are discarded, this is due to the fact, that the *Decompressor* needs information about the compressed packets *in* order to decompress them.

The SC state sends the dynamic information, which can be updated in the *Context*, helping the *Decompressor* to recover from unsynchronized *Context*.

In the FC state, the *Decompressor* contains the full *Context* and it is able to decompress the packets correctly.

The *Decompressor* starts in the lower state and gradually advances to the following states, unless a decompressing error is detected. It starts by working in NO, as soon as the first packet is decompressed successfully the *Decompressor* changes to the FC state. When the *Decompressor* fails to decompress a packet in the FC state, it changes to the SC state. If it is still not able to decompress the RoHC packet successfully, it falls back to the NO state.

2.2.5 Packet Types

In the RoHC standard [1], the header fields are classified based on often their changes in a packet stream. There are five categories of header fields:

- **INFERRED:** fields that are never sent, since they can be inferred by using other fields in the header that are actually transmitted.
- **STATIC-DEF:** fields that are sent initially and identify the stream.
- **STATIC KNOWN:** fields that are never sent as their values are well-known.
- **CHANGING:** fields that may change unpredictably and therefore are always sent.
- **STATIC:** fields that are sent once, as their values never change during the stream.

Using this classification, RoHC divides the fields in static and dynamic parts of the original header. Table II lists the fields of a typical IP/UDP/RTP packet headers and their classification.

IP Packet		
Description	**Bits**	**Type**
Version	4	STATIC-DEF
IHL	4	INFERRED
TOS	8	CHANGING
Total length	16	INFERRED
Identification	16	CHANGING
Flags	3	CHANGING
Fragment offset	13	INFERRED
TTL	8	CHANGING
Protocol	8	STATIC-DEF
Header checksum	16	CHANGING
Source IP address	32	STATIC-DEF
Destination IP address	32	STATIC-DEF
UDP Packet		
Source Port	16	STATIC-DEF
Destination Port	16	STATIC-DEF
Length	16	INFERRED
Checksum	16	CHANGING
RTP Packet		
Version	2	STATIC-KNOWN
Padding	1	CHANGING
Extension	1	CHANGING
CRSC count	4	CHANGING
Marker	1	CHANGING
Payload Type	7	CHANGING
Sequence Number	16	CHANGING
Timestamp	32	CHANGING
Synchronization Source	32	STATIC-DEF
Contributing Source	0-480	CHANGING

Table II – Classified header fields

2.2.5.1 RoHC packets and packet types

A RoHC packet has the general format illustrated in Figure 6.

Padding	*variable length*
Feedback	*0 or more feedback elements*
Header	*variable, with CID information*
Payload	

Figure 6- RoHC packet general format

The *Padding* is any number (zero or more) of padding octets. Either the *Feedback* or the *Header* must be present in the packet.

The *Feedback* elements can be or not present in the packet and they always start with a packet type indication. *Feedback* elements contain the CID and feedback information (e.g. NACK) from the *Decompressor* to the *Compressor*.

There are three types of *Feedback* supported by RoHC:

- **ACK -** Acknowledges a successful decompression of a packet;
- **NACK -** Indicates that the dynamic part of the *Context* of the *Decompressor* is invalid;
- **STATIC-NACK -** Indicates that the static *Context* of the *Decompressor* is invalid or has not been established.

In addition to the type of *Feedback*, other information may be included in profile-specific *Feedback* information. *Feedback* sent on a RoHC *Channel* consists of one or more concatenated *Feedback* elements, where each *Feedback* element has the following format:

0	1	2	3	4	5	6	7	
1	***1***	***1***	***1***	***0***	***Code***			*feedback type octet*
Size								*if Code = 0*
feedback data								*variable length*

Figure 7- Feedback Format

If the *Code* field is 0, it indicates that the field *Size* is present in the packet. If the *Code* field contains a value between 1 and 7 indicates the size of the *feedback data* field in octets. The *Size* field is optional and indicates the size of the F*eedback data* field in octets. The *Feedback data* is profile-specific feedback information and also includes CID information.

The *Header* section has a variable length and it contains CID information. *Header* is either a profile-specific header or a specific RoHC packet type. Finally, the *Payload* field that corresponds to the payload of the original packet completes the RoHC packet.

2.2.5.1.1 Basic structure of the IR packet

The IR packet type transports the static part of the *Context* and in some situations, the dynamic part of the *Context.* The basic structure of an IR packet is represented in Figure 8.

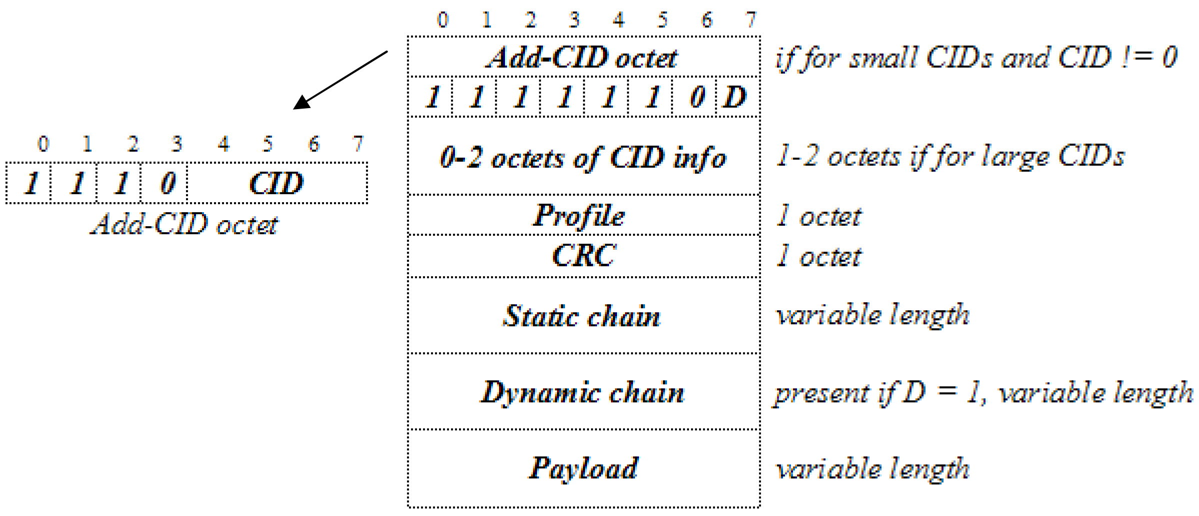


Figure 8- IR Packet Format

The *Add-CID* is the first field of the IR packet and this octet contains 4 bits for padding and 4 bits for the CID that identifies the *Context*. During the negotiation phase is establish which type of CID will be use. The CID space can be small, in which case the CIDs can take values form 0 through 15. In the other hand if the CID is large, the CIDs can take values between 0 and $2^{14} - 1 = 16383$. The use of large CID will imply the use of an extra field of CID info. The *D* field is a flag that indicates if the Dynamic *chain* is present in the packet. The profile field contains the profile identifier this defines the protocol it is going to be compressed. The *Cyclic Redundancy Check* (CRC) field is the result of the integrity of the header. The CRC in the IR and IR-DYN packet type is calculated using the entire packet except the payload, padding. It includes the *CID* or any *Add-CID* octet, except for the *Add-CID* octet for *CID* 0. This field is used to detect errors ant to prevent or reduce damage propagation. Following the CRC field there are two other fields of variable length, they are the *Static Chain* and the *Dynamic Chain*. The *Static chain* contains information about the static parts of the different protocol headers that compose a packet. The static part contains all the fields that never change during the stream. This section will describe, in detail, the static chain of RTP packet headers, which is represented in Figure 9. The RTP packet has the following protocol headers: IP, UDP, and RTP. The static part of an IPv4 header contains the Version, Protocol, Source Address, Destination Address fields. The source port represents the port number of the sender and the Destination port represents the port that the packet is addressed to. These fields are the static part of the UDP packet. The static part of an RTP header is the *Synchronization source* (SSRC) field. The value of this field is chosen randomly by the source host and identifies RTP session. Other sessions will have different SSRCs. Although the probability of multiple sources choosing the same identifier is low, all RTP implementations must be prepared to detect and resolve SSRC collisions. If a source changes its source transport address, it must also choose a new SSRC to avoid being interpreted as a looped source.

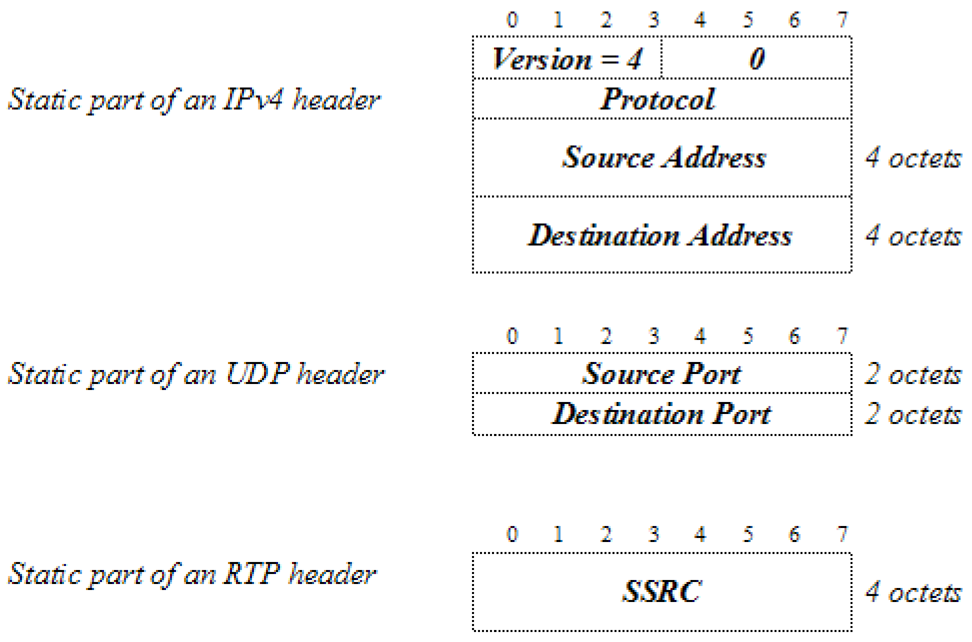


Figure 9- Static part of a RTP

Dynamic chain contains the dynamic parts of the different headers that compose the RTP packet: IP, UDP, and RTP. When dynamic information is not present is inferred from the *Static chain* information. The *Dynamic part* of the IP header is composed by the following fields: Type of Service, Time to Live, Identification, *Don't Fragment bit* (DF), RND, *Network Byte Order* (NBO) and the extension header list. *The Dynamic part* of the UDP header is composed by the UDP checksum. The fields *P, X, CC, PT, M, SN, TS, TS stride, CSRC* identifiers, compose the *Dynamic part* of RTP header. In Figure 10 represents dynamic parts of the different headers of the RTP packet headers.

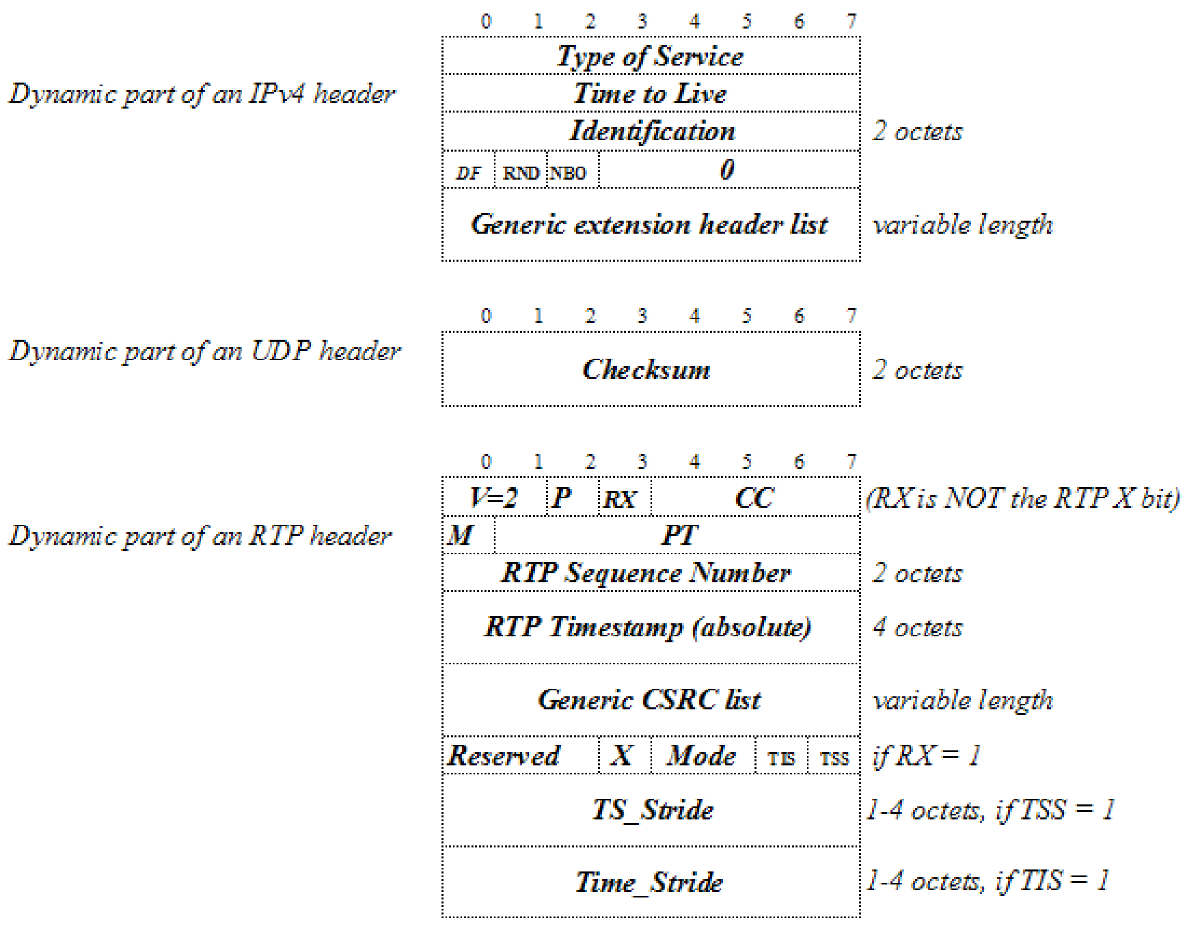


Figure 10- Dynamic part of the RTP packet headers

Finally, the IR packet last field is the *Payload* field that corresponds to the payload of the original packet. The presence of a *Payload* can be inferred from the packet length.

2.2.5.1.2 Basic structure of the IR-DYN packet

The IR-DYN packet is very similar to the IR packet. The only differences are the packet type field and the absence of the *Static Chain* sub header.

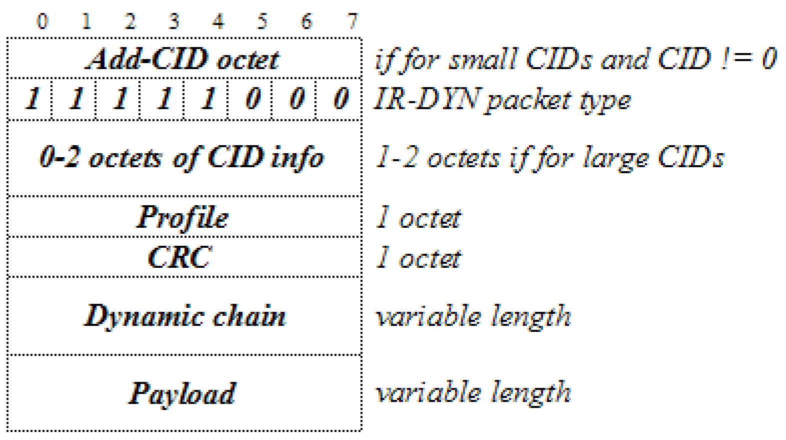


Figure 11- IR-DYN Packet Format

2.2.5.1.3 General format of a compressed RTP header

After the *Context* initialization, other types of packets can begin to be sent. The Figure 12 represents a general format of a compressed RTP header. RoHC RTP uses three packet types to identify compressed headers: packet type 0, 1and 2. The packet type fields are sent in the *first octet of base header* field and the remaining bits are sent in the *remainder of base header* as shown in Figure 12. The *Extension* field is optional and it is use to send additional bits of information. The orders of the fields following the optional Extension field if they exist have the same order as the uncompressed packet. The presence of the *IP-ID of outer IPv4 header* and the *IP-ID of inner IPv4 header* depend of the value of the RND2 and the RND respectively. The value of RND2 and RND must equal one. In addition, the presence of *GRE Checksum* field depend if the value of the GRE flag C equals one. The presence of the *UDP Checksum* in the packet is controlled by its value in the *Context*. If the value of the *UDP Checksum* is zero it indicates that the *UDP Checksum* is disable. In this case, the value of the *UDP Checksum* it is not sent in the packet. In the other hand if the value of the *UDP Checksum* is different from zero, indicates that the *UDP Checksum* is enable and it is send in each packet. The value of the *UDP Checksum* in the *Context* can only be update by IR and IR DYN headers and not by any of the packets types 0, 1 and 2.

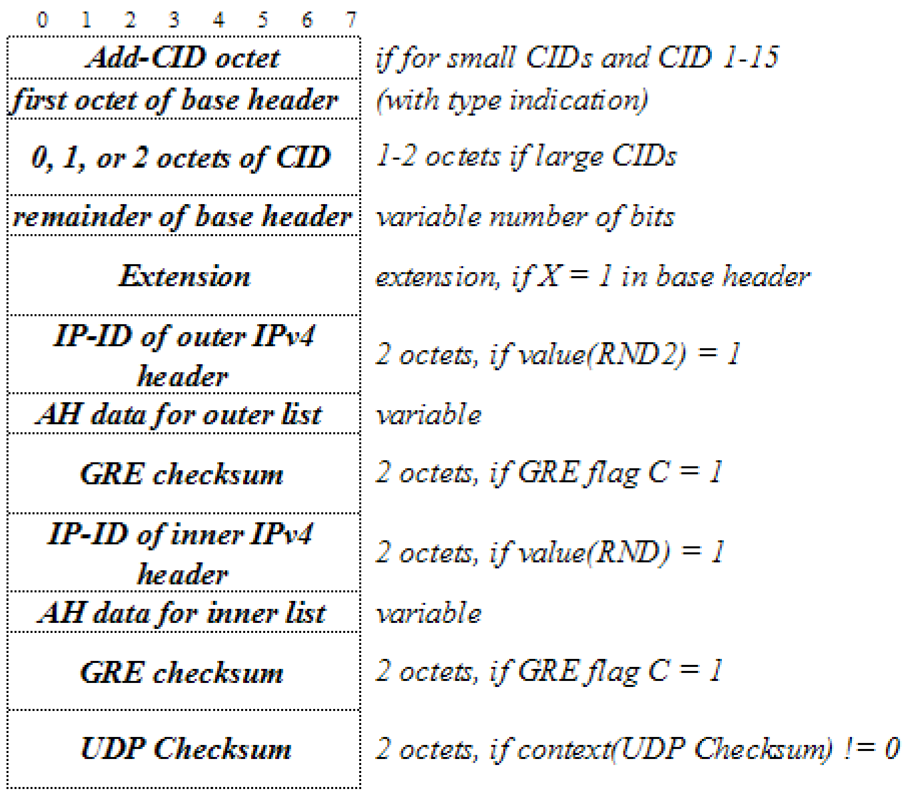

0 1 2 3 4 5 6 7
Add-CID octet
if for small CIDs and CID 1-15
first octet of base header
(with type indication)
0, 1, or 2 octets of CID
1-2 octets if large CIDs
remainder of base header
variable number of bits
Extension
extension, if X = 1 in base header
IP-ID of outer IPv4 header
2 octets, if value(RND2) = 1
AH data for outer list
variable
GRE checksum
2 octets, if GRE flag C = 1
IP-ID of inner IPv4 header
2 octets, if value(RND) = 1
AH data for inner list
variable
GRE checksum
2 octets, if GRE flag C = 1
UDP Checksum
2 octets, if context(UDP Checksum) != 0

Figure 12- General format of a compressed RTP header

2.2.5.1.3.1 Packet types

Besides, of the two types of packets for initialization/refresh, the RoHC scheme has three packet types to identify compressed headers. The format of a compressed packet can depend on the RoHC mode. Therefore, the standard adopted the following naming scheme for the packet types: *[Mode format is used in] - [Packet type number]-[Some property].*

Packet type 0

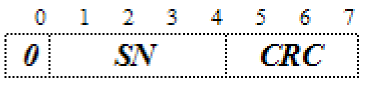


Figure 13- Packet type 0: UO-0

The first bit being 0 indicates packet type 0. UO-0 is the minimal packet used when the *Decompressor* knows all of the SN-functions. A small CRC is present in the UO-0 packet. It is 3-bit CRC calculated using the original IP and UDP headers, and it is used for integrity checks.

Packet type 1

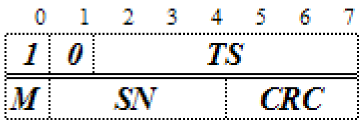


Figure 14- Packet type UO-1

If the first two bits have the value 10 indicate that is a packet type 1. This packet type is used when the number of bits needed for the SN exceeds those available in packet type zero, or when the parameters of the SN-functions for RTP TS or IP-ID change. Only the values of RTP SN, RTP TS and IP-ID can be used as references for future compression. A 3-bit CRC is also present in this packet type.

Packet type UO-1-IP-ID

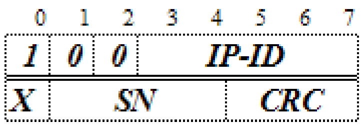


Figure 15- Packet type UO-1-ID

If the first two bits are 10, it indicates that it is a packet type 1. If the following bit is 0, it indicates format UO-1-ID. This packet type cannot be used if there is no IPv4 header in the context or if the value RND and RND2 equals 1. The X field can be used to indicate the presence of an extension to the header. Values of SN, TS, or IP-ID fields, transported in header extensions, are use to update the *Context* in the *Decompressor*. A small CRC is included in this packet.

Packet type UO-1-TS

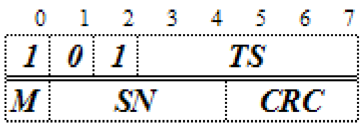


Figure 16- Packet type UO-1-TS

If the first two bits of this packet have the value 10, it indicates that it is a packet type 1 and if the following bit is 1, which indicates the format UO-1-TS. This type of packet cannot be used if there is no IPv4 header in the context or if the value RND and RND2 equal 1. This packet updates the following fields of the *Context* in the *Decompressor*: RTP

SN, and the TS. A small CRC is present in this packet

Packet type 2

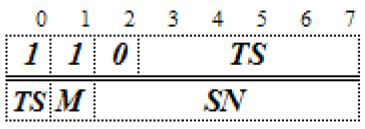


Figure 17- Packet type UOR-2

If the first three bits have the value 110, it indicates that it is a packet type 2. This type of packet cannot be used if there is an IPv4 header in the *Context* where the RND field equals zero. This disambiguates it from the packet-types UOR-2-ID and UOR-2-TS, which also starts with 110 bit sequence. This packet type can be used to change the parameters of any SN-function.

Packet type UOR-2-ID

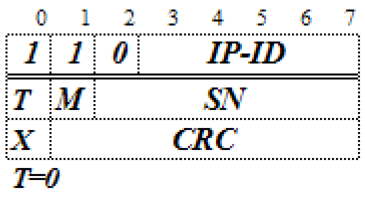


Figure 18- Packet type UOR-2-ID

If the first three bits have the value 110, it indicates that it is a packet type 2 and that the T field equals zero. This type of packet cannot be used if there is no IPv4 header in the *Context* or if value RND and value RND2 are both one. There is also a field X that indicates or not the presence of an extension to the packet. The CRC is 7-bit CRC calculated over the original IP headers and UDP headers, this server as an integrity check.

Packet type UOR-2-TS

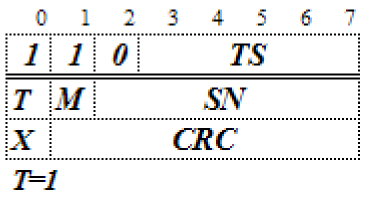


Figure 19- Packet type UOR-2-TS

The first bits being 110 indicate packet type 2 and the T field equals one. This type of packet cannot be used if there is no IPv4 header in the context or if fields RND and RND2 contain the value one. All values provided in UOR-2 packets update the context. The X field X can be used to indicate the presence of an extension to the header. The CRC is present in this packet.

2.2.5.1.4 Extension formats

Extensions are used to transport extra bits of information of the SN, TS and IP-ID fields. In the packet type 2 that may transport extensions. If they have the T field equal zero that will indicate that the +T field in the extensions below represents the IP-ID value and the –T field will represents the TS value. On the other hand if the T field equal one that will indicate that the +T field in the extensions below represents the TS value and the –T field will represents the IP-ID value.

Extension 0:

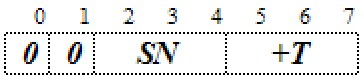


Figure 20- Extension 0

This extension is used when the number of bits of SN is less than or equal to 8 and the number of bits of the IP-ID is less than or equal to 3

Extension 1:

This extension is used when the number of bits of SN is less than or equal to 8 and the number of bits of the IP-ID is less than or equal to 11.

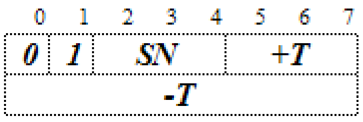


Figure 21- Extension 1

Extension 2:

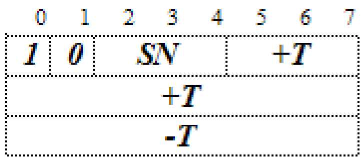


Figure 22- Extension 2

This extension is used when the number of IP header is higher than one, the number of bits of SN is less than or equal to 3 and the number of bits of the IP-ID is less than or equal to 11. In addition, the number of bits of the IP-ID of the second IP header is less than or equal to 8.

Extension 3:

The extension 3 represented in Figure 23, is more complete. It transports values for fields other than the SN, TS, and the IP-ID. It is used when the *Compressor* has to send the static and dynamic part of the *Context* or when the number of bits of SN is higher than 8 and the number of bits of the IP-ID is higher than 11. Three optional flag octets indicate changes to the IP header and the RTP header. The S, R-TS, I, ip, rtp, ip2, are flag fields that indicate the presence of fields: SN, TS, Inner IP header fields, IP-ID, Outer IP header fields and RTP Header flags fields respectively. The field TSC indicates if the TS field is scaled or not.

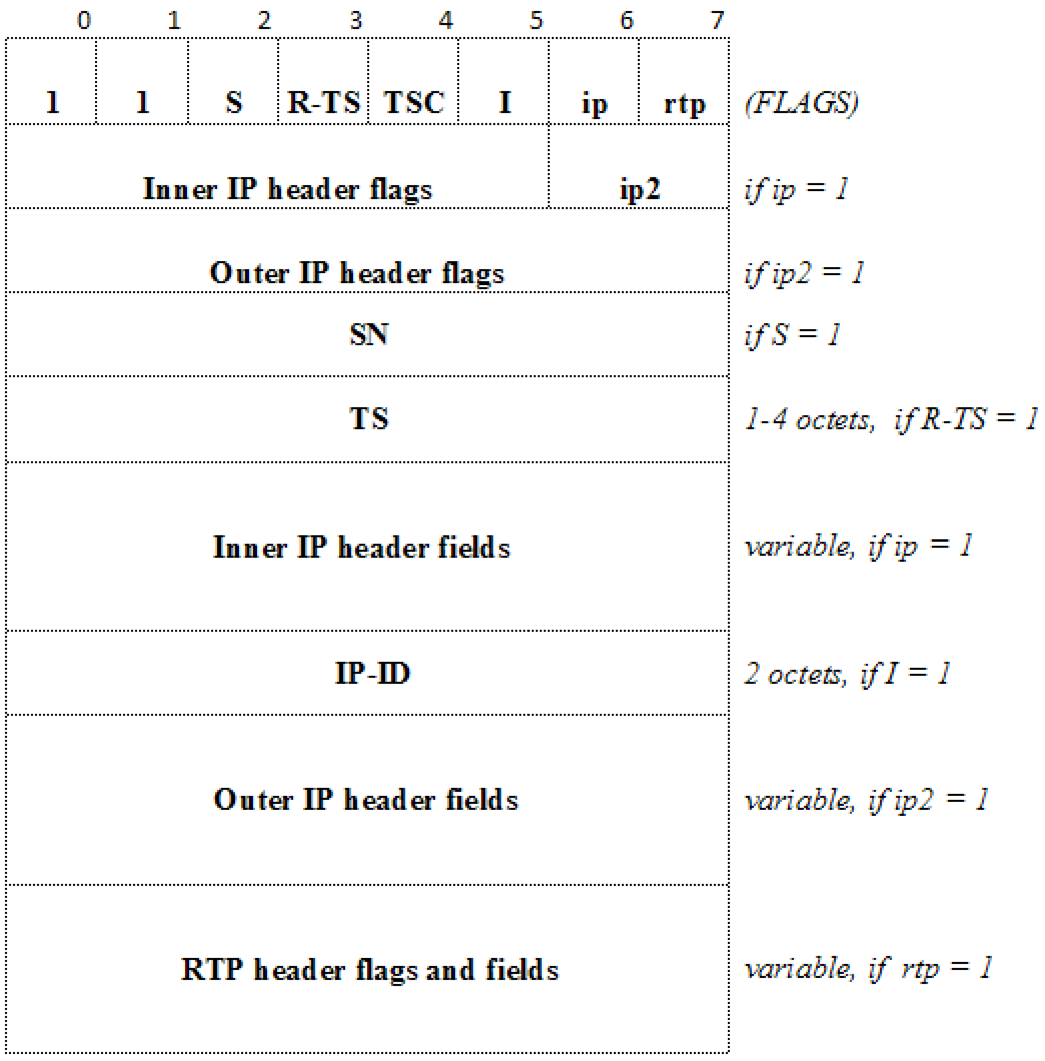


Figure 23- Extension 3

Inner IP header flags

The *inner IP header flags* indicate the presence of the fields TOS, TTL, PR, and IPX in the compressed header. The DF field is the same field that is present in the IP header. The NBO field indicates whether the octets of the header of this IP header are swapped before compression and after decompression, NBO indicates whether the octet is in network byte order or not. The RND flag indicates whether the IP-ID is not to be compress and is sent as is in the compressed header. The IP2 indicates the presence of the Outer IP header fields. Unless the static *Context* contains two IP headers, the IP2 field is always zero.

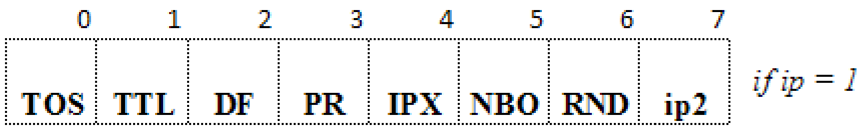


Figure 24- Inner IP header flags

Inner IP header fields

The *Inner IP header fields* are represented in Figure 25. They are the Type of Service/Traffic Class, Time to Live/Hop Limit and Protocol/Next Header fields correspond to the fields of the same name in the uncompressed IP header.

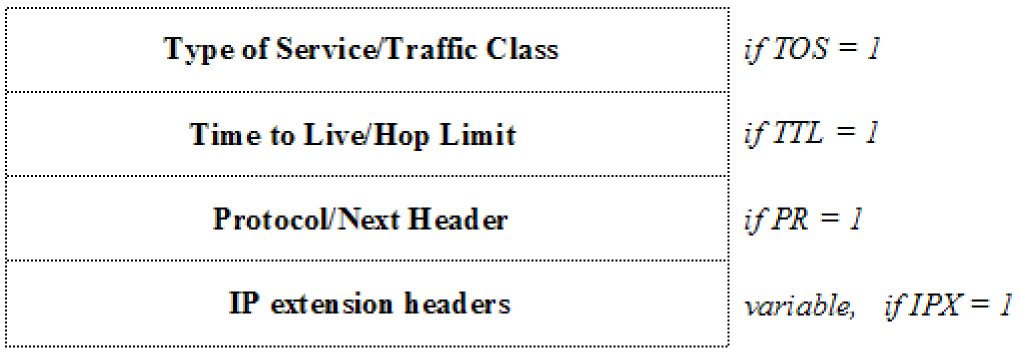


Figure 25- Inner IP header fields

Outer IP header flags

The fields represent in Figure 26 are part of the Extension 3 header refer to the *Outermost IP header*. These flags are the same as the *Inner IP header flags*, but refer to the *outer IP header*. The flag *I2*, however, has no counterpart in the *Inner IP header flags*. The *I2* flag indicates the presence of the IP-ID field in the compressed header

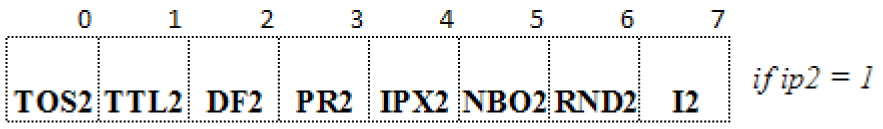


Figure 26- Outer IP header flags

Outer IP header fields

The fields in this part of Extension 3 refer to the *outer IP header* fields. The IP-ID field, however, has no counterpart in the *Inner IP header* fields. If the inner header is an IPv6 header, the IP-ID field of the *outer IP header* is never used and I2 flag is always zero.

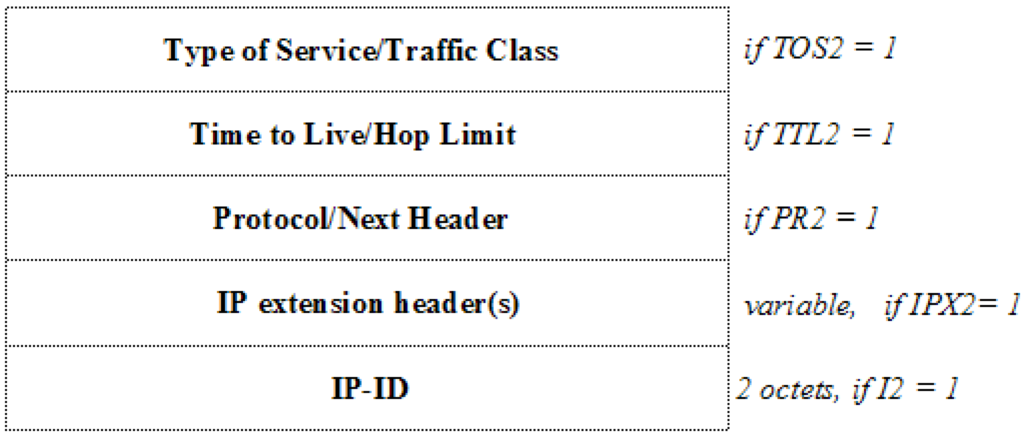


Figure 27- Outer IP header fields

RTP header flags and fields

The RTP header flags are composed by the *Mode* that indicates the compression mode. It can have the following values: 0 = Reserved, 1 = Unidirectional, 2 = Bidirectional Optimistic, 3 = Bidirectional Reliable. The rest of the flags R-PT, CSRC, TSS, TIS indicate the presence of fields in the extension. The R-P flag is the RTP padding bit, it can be presumed zero if absent. The R-X flag is the RTP extension bit that indicates if an extension is present. The TS_STRIDE predicts increment/decrement of the RTP TS field. The TIME_STRIDE calculates time interval in milliseconds between changes in the RTP Timestamp. It also indicates that the *Compressor* desires to perform timer-based compression of the RTP Timestamp field.

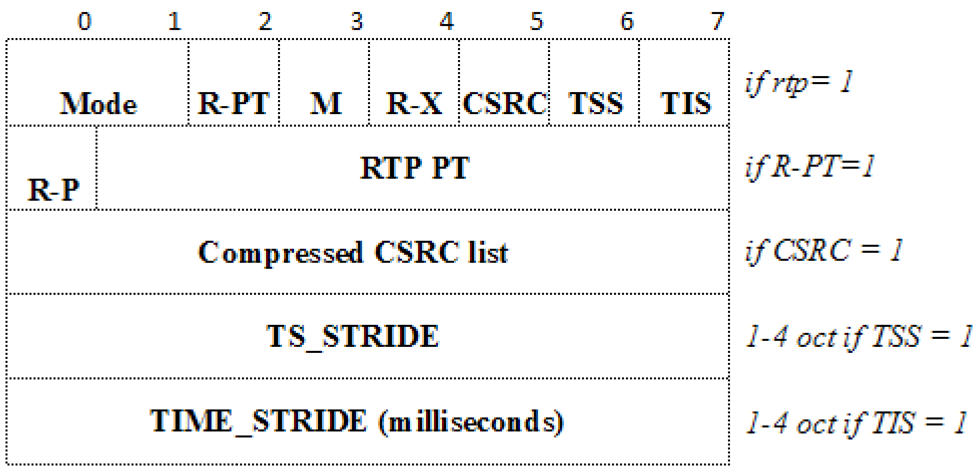


Figure 28- RTP header flags and fields

2.2.5.2 Packet types for each state

The decision of which packet to send depends of several facts: the compression state, the characteristics of the stream at the time of the arrival of the packet and other characteristics of the packet. The characteristics of the packet change when the number of bits of the SN, IP-ID and TS change or if the stream had a considerable change in the static or dynamic parts. In Table III is represented the packet types sent in each state.

When the *Compressor* starts, he always starts in IR state. In this state, all packets sent to the *Decompressor*, are IR packets. IR packets are used to initializes or to refresh the static and dynamic parts of the *Context* in the *Decompressor*. In FO state, the *Compressor* can send two types of packets IR-DYN or UOR-2 depending on characteristics of the stream. The packets send in this state serve to communicate irregularities in the packet stream therefore the big majority of information send corresponds to dynamic part of the *Context* and only a few static fields are updated. The SO state is the higher state of the compression, in this state the *Compressor* can send tree type of different packets UO-0, UO-1, UO-2. In this state as in the previous states, the decision between the packets depends on the characteristics of the stream. If the packet types send, are UO-1-IP-ID, UO-2-IP-ID or UO-2-TS, this packets can take extensions. These extensions take the extra bits from the SN, TS and IP-ID fields. The decision of witch type of extension is send depends in what field or fields as the extra bits of information to transport and what type of information it is needed to send.

Compression State	**IR**	**FO**	**SO**
Header Format packet used	IR packet	IR-DYN or UOR-2	UO-0, UO-1, UOR-2

Table III – Type of packet for each state

2.2.6 Encoding methods

This section describes the encoding methods used in the RoHC standard [1]. These methods can reduce a significant number of bits of a field by encoding the fields that have large values and small constant changes. The methods described are *Least Significant Bits* (LSB) encoding and *Window-based LSB* (W-LSB) encoding.

With LSB encoding only the k, least significant bits are transmitted. The *Decompressor* receives the k value and using the previously received reference store in the *Context*, v_{ref} it can derive the original value. Both the *Compressor* and *Decompressor* work with an interpretation interval that can be described as:

$$f\left(v_{ref}, k\right) = [v_{ref} - p, v_{ref} + \left(2^k - 1\right) - p]$$

The function f defines for every value of k a single and indentified value in the interpretation interval. The variable p is an integer used to change the interval with respect to the v_{ref} depending on the field characteristic. For example, if the value of the field is expected to stay the same or increase the p will assume the value zero. Therefore, the *Compressor* to compress the value v, it as to find a minimum value k, such that v falls into the interval $f(v_{ref,}k)$. The *Decompressor* is able to obtain the original value v from the k bits received.

The W-LSB encoding provides for robust LSB encoding. The *Compressor* is unable to determine the exact reference value, v_{ref}, used by the *Decompressor*, for a value v. Therefore, the *Compressor* maintains a sliding window of possible values sent to the *Decompressor*. The Compressor stars with an empty sliding window. Every time a value v is compressed, this value is added to the sliding window. For each value being compressed, the *Compressor* determines the minimum number of bits k necessary to compress the value, such as $k = max(g(v\_min, v), g(v\_max, v))$. The v_min and the v_max variables correspond to the minimum and maximum values in the sliding window, and g is the function defined previous for LSB encoding. When the *Compressor* is receives confirmation that the Decompressor will not use a certain value v and all values older than v as reference, the window is shift by removing those values, including the *v* value.

2.2.6.1 Encoding in the Compressor

The calculation of LSB for fields such SN, TS or IP-ID, in the *Compressor,* goes thru a series of steps: 1) k calculation, which is achieved using the expression. $k = FLOOR\left(log_2(v_{ref}\ XOR\ v)\right) + 1$; 2) To create a mask is use this expression $mask = 2^k - 1$ in which the k is value achieved in step 1; 3) To calculate the value of LSBs the following expression is use $LSBs = v\ \&\ mask$.

2.2.6.2 Decoding for the Decompressor

To decode the LSB receive in the *Decompressor* the following steps are taken: 1) the value of k is self-containing. On receiving the value of the LSBs, the *Decompressor* can derive the k using the expression $k = FLOOR\big(log_2(LSBs)\big) + 1$; 2) Form a mask for the value that it is being decoded (i.e. SN (16 bits)) using the expression $mask = (2^{16} - 1)(2^k - 1)$; 3) to calculate the original value v the following expression is used $v = (ref\ \&\ mask)|LSBs$.

2.2.6.3 Scaled RTP Timestamp Encoding

The RTP TS usually increases by a fixed time interval. Therefore, when using Scaled RTP TS Encoding the TS is reduce by a factor of TS_STRIDE before compression. This saves *floor(log2(TS_STRIDE))* bits for each compressed TS. The TS may be derived by the linear function below, where the TS STRIDE is the fixed time interval, the TS SCALED is the integral multiple, and the TS OFFSET is the linear offset;

$$TS = TS_{SCALED} * TS_{STRIDE} + TS_{OFFSET}$$

The TS STRIDE is explicitly communicated from the *Compressor* to the *Decompressor*. TS OFFSET is implicitly communicated by sending several uncompressed TS values from which the *Decompressor* can extract its value. The TS OFFSET is updated locally at TS wraparound via special interpretation interval rules. After the initialization, only the TS SCALED is communicated between the *Compressor* and the *Decompressor* using W-LSB encoding.

2.3 Conclusions

This chapter starts by analysing the problem associated with the transmission of IP packets and the large protocol header overhead they create, which consumes great part of the bandwidth available. This overhead provides key motivation for using HC schemes. This chapter has presented the state of the art of HC schemes, giving a brief description of the different schemes that preceded the development of the RoHC solution. The HC schemes have been studied since 1984 with the Thin-wire protocol [2] proposed for connecting personal computers to the Internet. Van Jacobson late proposed cTCP [3], a protocol based on the header redundancy information as is predecessor. This protocol compressed the 40 bytes of the TCP/IPv4 header to 3–6 bytes. There are other proposals for the different protocol headers based on redundancy, such as the cRTP, which is a header compression mechanism with a detailed specification for RTP protocol for real-time data streams.

This chapter also addresses the fundamentals of the RoHC framework. All the specifications detailed in this chapter are described in the RoHC standard [1] and, particularly in, the chapter focused in the Profile 0x0001 for compressing RTP/UDP/IP packets. The RoHC protocol is composed by two instances, which compose a *Node*: the *RoHC Compressor*, responsible for the compression of the packets headers; and the *RoHC Decompressor*, responsible for the decompression of the packets headers. The RoHC scheme has three modes of operation: the U-mode, the O-mode, and the R-mode. The *Compressor* always starts in the U-mode and its transition to any bidirectional modes can happen as soon as a packet has reached the *Decompressor* and a feedback packet is sent back. In order to determine in which mode to operate, RoHC depends on environment characteristics such as header size, feedback and error probabilities. Both the *Compressor* and the *Decompressor* can be regarded as states machines. The *Compressor* operates in three states: IR, FO, and SO. The *Decompressor* also operates in three states: NO, SC, and FC. The *Compressor* starts in the IR state and it can change to the other states, when some of the RoHC parameters are meet, such as the number of packets sent in the IR state. The *Decompressor* starts in NO state and it can change to another state, as soon it is decompress successfully a compressed packet. Described also in this chapter are the type of packets exchanged between the *Compressor* and *Decompressor* and the optional extensions. The chapter also describes the use of encoding methods, which save significant number of bits in the compressed packets, such as LSB and W-LSB and Scaled RTP TS Encoding.

3 RoHC Performance over Heterogeneous Networks

The third and the fourth generation of wireless systems are being developed to support a wide range of services, including audio and video applications. The flexibility of services offered by these generations of wireless systems is introduced by the use of the *Internet Protocol* (IP). The major problem of using IP based protocol architectures is the large overhead, which affects the limited bandwidth of wireless channels. Applications that generate small payloads, such as *Voice over IP* (VoIP) can be responsible for large overheads. In certain cases, this overhead can be about three times the size of the payload or 75% of the total packet size [5]. Assuming that all the *Global System for Mobile Communications* (GSM) traffic migrates to 4G packets switched networks, the amount of overhead created by the VoIP traffic will be considerable. Therefore, using header compression techniques such as RoHC can be desired for wireless heterogeneous networks.

This chapter describes the use of HC schemes and in particularly the RoHC scheme, over different networks. Other possible solutions presented to minimize the problem of large overhead over low bandwidth links are also addressed.

3.1 Related work over Heterogeneous Networks

3.1.1 GPRS

General Packet Radio Service (GPRS) [33] is an end–to-end mobile packet radio system that extends the existing GSM network developed by 3GPP.

3.1.1.1 GPRS Architecture

The GSM networks are composed by several functional elements. The *Base Station Subsystem* (BSS) or *Base Transmission Stations* (BTS) contains the equipment for transmitting and receiving of radio signals from the *Mobile Stations* (MS), and equipment for encrypting and decrypting communications with the *Base Station Controller* (BSC). The BSC controls one or more BTS, and each BTS may serve one or more cell areas. The *Mobile Switching Centres* (MSC) is the central component of the *Network Security Services* (NSS) and connects BSC to, the *Home Location Register* (HLR), the *Visited Location Register* (VLR), and the *Authentication Centre* (AUC).

The elements that support the use of GPRS are *Serving GPRS Support Node* (SGSN) and the *Gateway GPRS Support Node* (GGSN). The SGSN is responsible for routing the packet switched data to and from the MS within its cell area. The SGSN is responsible for packet routing and transfer, mobile attach and detach procedure, location management, assigning channels and time slots, authentication and charging for calls. It stores the location information of the user and user profile of registered users in its location register. The GGSN acts as interface between the GPRS backbone and the external *Packet Data Network* (PDN). It converts the GPRS packet coming from the SGSN into proper *Packet Data Protocol* (PDP) (i.e. IP, X.25) format before sending to the outside data network. The Figure 29 represents a simplified view of the GPRS architecture.

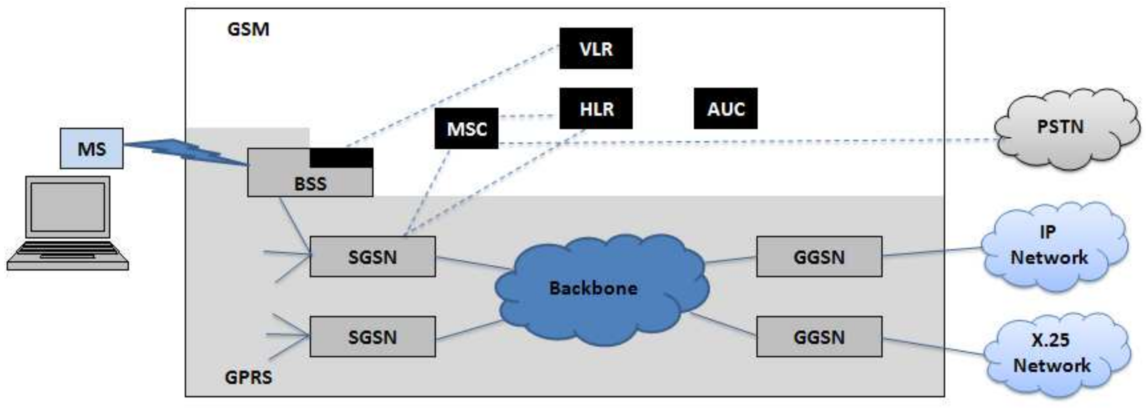


Figure 29- GPRS Architecture

When a GPRS device is turns on, it will scan for a local GPRS channel. If a channel is detected, the device will attempt to attach to the network. The SGSN receives the attach request, fetches subscriber profile information from the subscribers HLR node, and authenticates the user. The SGSN uses the profile information to determine which GGSN will be used. The selected gateway may perform a *Remote Authentication Dial-In User Service* (RADIUS) authentication and allocate a dynamic IP address to the user before setting up connections to outside networks. This process is called the packet data profile context activation and the setup may vary from one carrier to the next. It may include additional functions like *Quality of Service* (QoS) management and V*irtual Private Network* (VPN) tunnel management.

When the GPRS device is powered off or it moves out of a GPRS coverage area, its context is deactivated and the device is detached from the network. When the mobile user sends data, the SGSN routes the packets to the appropriate GGSN. The GGSN then routes the data according to the current context established for the session. On the other hand, packets destined for the user are routed to the GGSN associated with the users IP address. The GGSN checks the received packets against the current context, identifies the SGSN that is serving the user and routes the traffic accordingly. The SGSN then forwards the packets to the BSS where the subscriber is located. Each dedicated channel is divided into eight time slots, with each time slot supporting a maximum data transmission speed.

3.1.1.2 Header Compression in GPRS

The HC is performed in the *Sub Network Dependent Convergence Protocol* (SNDCP). SNDCP is responsible for the transparent transfer of data between MS and the SGSN as is showed in Figure 30. The header compression mechanisms currently supported by SNDCP protocol are cTCP [3], IPHC [20] and later the 3GPP has recommended the use of RoHC [1].

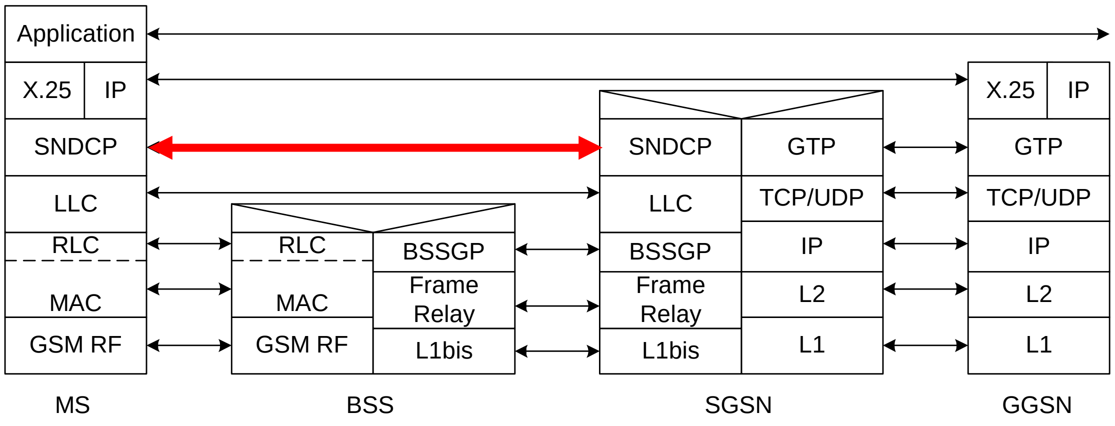


Figure 30- GPRS Transmission Plane

3.1.2 UMTS

The *Universal Mobile Telecommunications System* (UMTS) [37] is the 3G mobile network developed by 3GPP. The UMTS networks support greater numbers of voice and data customers, especially in urban areas. Also, include features such as enhanced multimedia, wide-area wireless voice telephony, video calls, and broadband wireless data, all in a mobile environment. It allows the transmission of 384 Kbit/s for mobile systems and 2 Mb/s for stationary systems. The 3G users have greater capacity and better spectrum efficiency, which allows them to access global roaming between different 3G networks.

3.1.2.1 UMTS Architecture

The UMTS networks consist of three interacting elements: *Core Network* (CN), *UMTS Terrestrial Radio Access Network* (UTRAN) and UMTS *User Equipment* (UE). The

CN architecture for UMTS is based on GSM network with GPRS. All equipment has to be modified for UMTS operation and services. The main function of the CN is to provide switching, routing and transit for user traffic. The CN also contains the databases and network management functions The UTRAN provides the air interface access method for UE. The BTS is referred as Node-B and the control equipment for Node-B's is called *Radio Network Controller* (RNC). The Node-B functions include air interface transmission and reception, error handing and closed loop power control. The RNC functions are: radio resource control, admission control, channel allocation, handover control and open loop power control. UE terminals work as an air interface with the Node-B. The Figure 31 illustrates the UMTS network architecture.

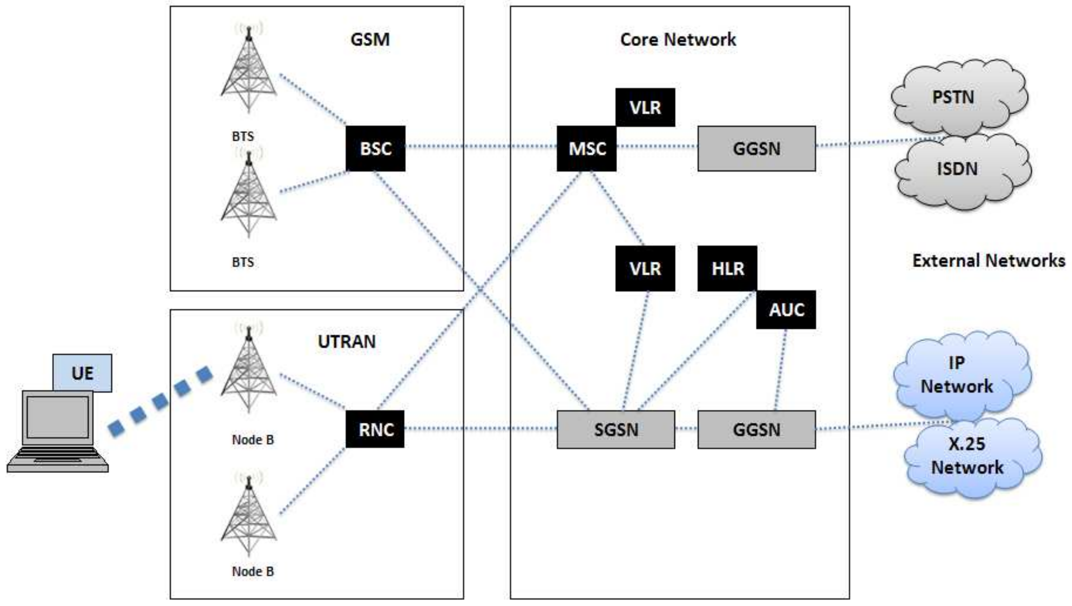


Figure 31- UMTS Architecture

3.1.2.2 Header compression over UMTS

The UMTS and it use of IP based services that generate large amounts of overheads that consume great part bandwidth available in radio links. The header compression protocols are supported in the *Packet Data Convergence Protocol* (PDCP) [38] layer of the UMTS. This protocol replaces the SNDCP protocol in the stack of protocols. Figure 32 presents the UMTS stack of protocols.

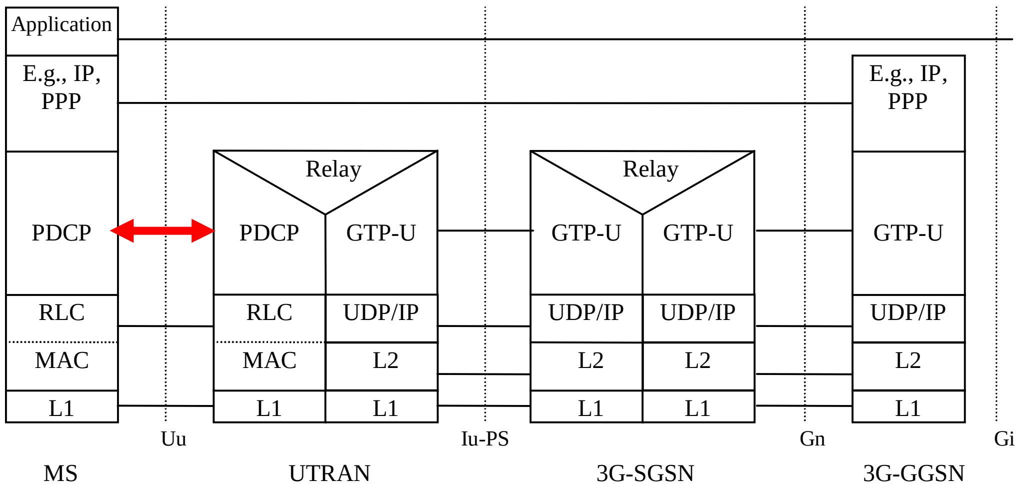


Figure 32- UMTS stack of protocols

The PDCP can be configured with *No Compression* in which case it will send the IP Packets without compression. It can be also configure to compress the packets according to its configuration of the upper layer and attach a PDCP header. The PDCP header consists of two fields: PID and PDU TYPE. The PDU TYPE field indicates whether the PDU is Data PDU or Sequence Number PDU. The PID field value indicates header compression type used and packet type or CID. The PDCP supports IPHC [20] and RoHC [1] as header compression protocols.

The compressed header packets over a network such as UMTS can be confronted with a variable environment that could lead to deficient compression. The use of correct compression parameters adapted to these changes will give better robustness and efficiency of RoHC. As always to maximize RoHC gain the parameters, IR_TIMEOUT and FO_TIMEOUT timeouts, and L_VAR parameter must be configure although they are no values defined in the RoHC standard [1] some values have been proposed in same publications [6][7] for RoHC over UMTS. Those values propose are: IR_TIMEOUT values between 200 and 400 packets if a value of FO_TIMEOUT is around 90. If the value of FO_TIMEOUT is around 30 packets the value for IR_TIMEOUT must be higher in order to maintain robustness. These studies used as metric the *Average Compress Header Length* (ACL) and the *Additional Packet Loss propagation* (APL) to quantify the RoHC performance over UMTS. There is a direct relation between the ACL and the RoHC variables. For instance, in U-mode the ACL is proportional to the L_VAR [7]. These studies also conclude that the high robustness and efficiency of RoHC highly improves the communication in a real situation.

3.1.3 Ultra Mobile Broadband

Ultra Mobile Broadband (UMB) [31] a standard developed by the *3rd Generation Partnership Project 2* (3GPP2). The UMB is in an evolution of CDMA2000 that significantly increased the efficiency of the wireless communications when compared to the later. The UMB uses IP and a next generation radio system. Simulations [32] performed on the subject showed that the UMB system improved significantly the VoIP capacity when compared to his predecessor. A 5 MHz UMB network can support up to 320 users simultaneously and support an upstream data throughput up to 2.26 Mbps. To provide compatibility with other systems, UMB system, supports handoffs with other technologies including existing CDMA2000 1X and 1xEV-DO systems. The UMB network encodes the digitized voice speech before transmission. The combine size of the header protocols of a VoIP packet is approximately 320 bits. To transmit a 320 bits header in a 16-17 bit frame is undesirable, so before transmission the headers are compressed using RoHC [1], to 16 bits [32].

3.1.4 IEEE 802.11

The IEEE 802.11 [36] wireless logical architecture contains several main components: *Station* (STA), wireless *Access Point* (AP), *Independent Basic Service Set* (IBSS), *Basic Service Set* (BSS), *Distribution System* (DS), and *Extended Service Set* (ESS). An IEEE 802.11 wireless LAN is based on a cellular architecture. Each cell is a BSS controlled by AP and the serving stations. An AP has with two interfaces: one on the wireless LAN and another wired interface. Its function is to serve as bridge between the STA and the DS. An IBSS is a wireless network, consisting of at least two STAs, used where no access to a DS is available. An IBSS is also sometimes referred to as an ad hoc wireless network. An ESS is a set of two or more BSS.

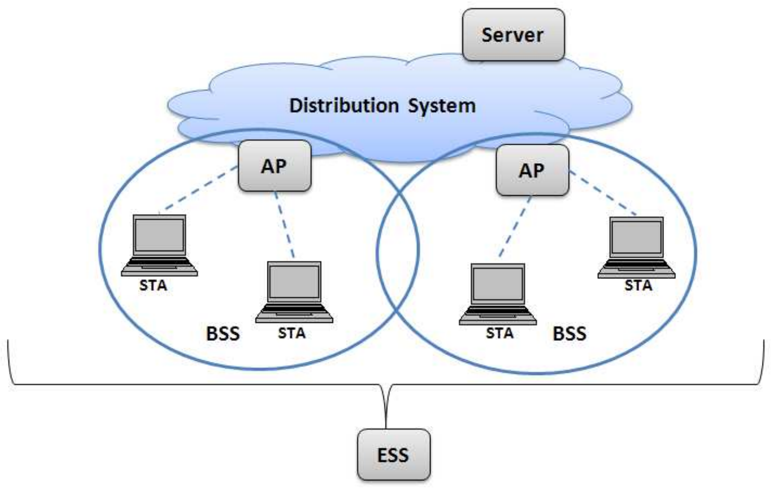


Figure 33- IEEE 802.11 Architecture

The use of IP headers that contribute a large overhead; for example, for VoIP, a packet has a total IP/UDP/RTP header size of 40 octets in IPv4. The size of the payload may be as low as 15-20 octets. Then the need to reduce header size for efficiency is obvious, especially for wireless links. Several methods have been proposed to reduce the header size. However, wireless links have characteristics that make header compression less efficient. They have to be robust enough to perform well in an environment with high bit error and packet loss rates. RoHC was proposed to save bandwidth in the narrow radio spectrum and to reduce the packet loss rate over unreliable wireless links. The standardization of RoHC over wireless LANs such as the IEEE 802.11 can provide better transmission efficiency to several applications such as VoIP. The IEEE 802.11 is a technology, which will play an important role in 4G networks. The 4G networks are expected to be IP-based networks with QoS support.

Some RoHC performance studies over 4G networks showed results that RoHC saves bandwidth, which is very scarce in wireless networks. The gathered results showed that packet compression ratios go up to 71.5% [35]. In addition, this study indicates that RoHC does not significantly deteriorate the delay jitter in the voice signal. In overall, it even improves voice quality [35]. RoHC study also been done to examine impact of RoHC in wireless video transmissions, such experiments revealed that has small impact on the quality of video received. RoHC reduces the bandwidth requirement by about 10% for intermediate quality video. For lower quality video, the bandwidth reductions are significantly larger up to 40% [12].

3.1.5 Measuring Header Compression Gain

The *RoHC-Gain* is a compression metric, introduced to measure the performance of RoHC over 802.11 networks [35].

> *RoHC-Gain is characterized as the relative amount of extra payload (data) bytes that becomes available for other flows after applying RoHC [35]*

The *RoHC-Gain* is based on the payload bytes received by all the streams, with and without the application of RoHC. It can only be quantified when there is additional traffic present that can occupy the bandwidth freed by the RoHC compression.

The RoHC Gain can be calculated by using the following equation:

$$RoHCGain = \frac{TotalPayloadRcvdUsingRoHC}{TotalPayloadRcvdNotUsingRoHC} - 1$$

All of the parameters used to characterization of RoHC are represented as a function of the percentage of uncompressed traffic received in the 802.11 network. This does not affect the amount of the payload received only the header lengths. For example, the following equation gives percentage of uncompressed VoIP traffic received in the 802.11 network [35].

$$\mathrm{VoIPTrafficReceived} = \frac{\mathrm{VoIPPayloadRcvd}}{\mathrm{TotalPayloadRcvdNotUsingRoHC}}$$

Study [35] in this area showed that the use of RoHC could be advantageous, when the flow transports other flows that can take advantage of the freed bandwidth. Results indicate a maximum *RoHC-Gain* of 23%, for a medium voice quality.

3.1.6 Sensor Networks

Sensor Networks were developed initially for military applications such as battlefield surveillance. Later on the use of *Sensor Networks* spread to other applications such as healthcare applications, home automation, and traffic control.

There are several types of *Sensor Networks*. One of the most common is the *Wireless Sensor network* (WSN). WSNs are formed by distributed autonomous devices using sensors that together monitor an environment or other physical conditions such as sound, temperature or motion. Another type of *Sensor Network* is the *Wireless Multimedia Sensor Networks* (WMSN). These networks contain sensor nodes equipped with cameras, microphones, and other sensors producing multimedia content. A *Sensor Network* is usually a wireless ad-hoc network, where each sensor supports a multi-hop routing algorithm. One of the problems that *Sensor Networks* face is that node resources, such as memory, computation power and energy are very limited. Wireless transmissions usually consume a lot of power, so the transmission power must be reduce to spare energy. With this in mind and taking as example the IP-based *Sensor Networks*, the use of HC schemes, can be desirable, since the nodes will transmit less bytes thus saving energy.

3.1.6.1 ZigBee Networks

ZigBee [30] is a technology that consist in a group of specifications for wireless networked sensors and controllers. ZigBee is built on top of the physical (PHY) layer and medium access control (MAC) layer defined in the IEEE standard 802.15.4 [30]. The Figure 34 represents the ZigBee architecture. Compared to other wireless communication networks, ZigBee is designed specifically for providing wireless networking capability for battery-powered, low-cost, low capability sensor and controller nodes.

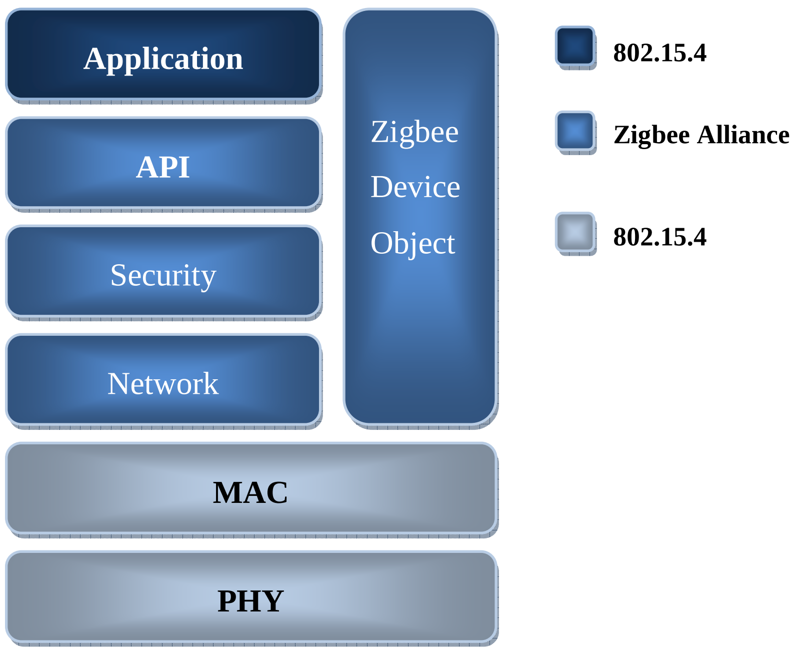


Figure 34- ZigBee Architecture

At its core, ZigBee is mesh network architecture. Its network layer natively supports three types of topologies: both star and cluster tree networks and generic mesh networks. Every network must have one coordinator device, responsible with its creation, the control of its parameters and basic maintenance.

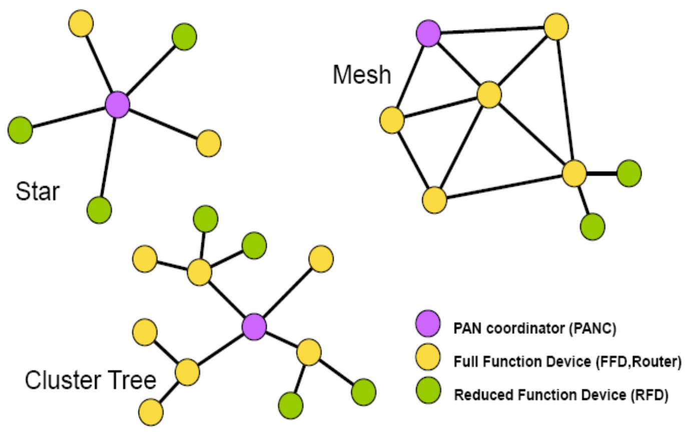

Star
Mesh
Cluster Tree
PAN coordinator (PANC)
Full Function Device (FFD,Router)
Reduced Function Device (RFD)

Figure 35- ZigBee Topologies

Studies [17] have been made to test the performance of VoIP in such networks where the compression header mechanisms (RoHC standard) were used to help reduce the number of bytes of information exchange between the nodes. These studies conclude that ZigBee networks were capable of supporting VoIP communications, but with some limitations. They are limited to a small number of voice connections within a small number of hops. The increase of any of these factors will cause transmission delay and packet loss ratio also to increase.

3.1.7 IEEE 802.16e

IEEE 802.16e [34] also known commercially as Mobile WiMAX is design to provide mobile wireless broadband services for data communication. Although the Mobile WiMAX as option a header compression mechanism called *Payload Header Suppression* (PHS), it has very limited compression capabilities. The PHS can only compress the static fields between consecutive packets belonging to the same packet stream. In addition, if the size of the static fields or the size of the sum of static fields which are located successively, it cannot be compressed. The PHS, therefore, has low compression ratio and does not work well over links with high bits error rates, since it cannot detect the bit error at the radio link. Therefore, an alternative to PHS is required. Performance studies [16] of RoHC over mobile WiMAX have been made. RoHC was proposed due to his high efficiency in wireless links. Studies [16] showed that RoHC is able to provide an efficient use of radio resource compared to PHS. The PHS provides a fixed compression efficiency saving 18 bytes per header while RoHC in every mode of operation, can double the amount of bytes saved [16].

3.2 RoHC Performance Problems and Short lived flows

In some case scenarios, the RoHC protocol may not perform well in scenarios that consist of short-lived flows. RoHC requires that the first packets send contain the full header in order to build its *Context*. For short-lived flow, this poses a problem since the firsts packets sent can be the only packets ever sent. In this case, the compression ratios obtained are close to zero. To reduce the overhead generated by the establishment of the *Context* and to make this step faster, for this flows. The *Compressor* performs the *Context* reutilization by reusing an existing *Context*. The *Compressor* sends a packet to *Decompressor* with the information of the selected *Context*, along with some fields that need to be updated in the new replicated *Context*. The *Decompressor* replicates the *Context* according with the information received for the *Compressor* [28]. The use of this technique raises security considerations that have to be taken in account especially if the *Compressor* uses a corrupted *Context* as a base for replication. This could lead to a failed attempt to initialize a new *Context* in the *Decompressor* or it can originate the *Decompressor* to reconstitute packets that do not corresponded to the original packets.

3.3 Conclusions

This chapter gives an overview on the use HC schemes over heterogeneous networks.. The key motivation for the use HC schemes over the different networks that support IP-based services is the large overhead created by the packets headers.

The chapter starts by describing the architecture and the use of HC schemes in networks such as GPRS and UMTS. In these networks, the HC schemes are supported in the protocol stack by the protocol SNDCP in the GPRS network, and PDCP in the UMTS network. The HC schemes supported by the GPRS networks are the cTCP and the IPHC schemes and, in the UMTS networks, the IPHC scheme. Studies [6] [7] in this area conclude that the high robustness and efficiency of RoHC improve the communication over theses networks. These studies also propose values for IR_TIMEOUT, FO_TIMEOUT that are not quantified in the RoHC standard [1]. The IR_TIMEOUT values are between 200 and 400 packets if a value of FO_TIMEOUT is around 90; if the value of FO_TIMEOUT is around 30 packets, the value for IR_TIMEOUT must be higher in order to maintain robustness. Also analyzed in this chapter is the use of HC schemes through the following networks: the IEEE 802.11 networks, the sensors networks, and IEEE 802.16e networks. The use of RoHC over wireless LANs such as the IEEE 802.11 can provide better transmission efficiency to several applications such as VoIP. The study [35] showed that packet compression ratios go up to 71.5%. In addition, this study indicates that RoHC does not significantly deteriorate the delay jitter in the voice signal. RoHC studies on the impact of RoHC in wireless video transmissions revealed that RoHC has small impact on the quality of video received. RoHC reduces the bandwidth requirement by about 10% for intermediate quality video. For lower quality video, the bandwidth reductions are significantly larger, up to 40% [12]. In sensors networks the use of HC schemes are beneficial, so they are enable to save energy. Performance studies [16] of RoHC over IEEE 802.16e show that RoHC provides an efficient use of radio resource compared to the header suppression mechanism PHS. The PHS provides fixed compression gains, saving 18 bytes per header while RoHC, in every mode of operation, can double the amount of bytes saved.

This chapter also presented a metric used to measure the performance of RoHC over IEEE 802.11 networks called RoHC Gain [35]. This metric is characterized as the relative amount of extra payload (data) bytes that becomes available for other flows after applying RoHC.

4 RoHC U-mode Implementation

This chapter describes the requirements, the architecture and tools used in development of the RoHC U-mode implementation. The RoHC U-mode implementation was limited to the Profile 0x0001, for sending compressed IP/UDP/RTP headers. The implementation aimed to develop a user space prototype running in a Linux environment that is transparent to the application generating RTP traffic.

Initially, to ease the development process, a tester application was developed that read a set of RTP packets previously captured with *libpcap* [18] and sends them to another host. Later on, a new version of the implementation replaced this tester application. This version consisted on intercept the uncompressed RTP packets in the sender with the use of a TAP interface, an Ethernet virtual interface. The packets compressed by the RoHC U-mode implementation and sent through the network towards the RoHC *Decompressor* node, with a RoHC ethertype. When the compressed packet arrives at the *Decompressor* is decompressed and delivered to application. To simulate RTP packets it was used the packet generator *packETH* [13].

In this chapter is also described the data structures used to developed the RoHC U-mode implementation. Is also described the major features of this implementation. These features include the description of the operation procedures of the *Compressor* and *Decompressor*. The RoHC U-mode implementation was developed to support two network scenarios: 1) nodes connected to an Ethernet switch; 2) nodes connected to an IEEE 802.11 Access Point.

4.1 Requirements

The RoHC implementation was limited to the U-mode and to the Profile 0x0001, for sending compressed IP/UDP/RTP headers. This implementation does not implement the negotiation phase. All the variables that are negotiated in this phase are statically initialized. The RoHC U-mode implementation was developed to work in a network connected by an AP or switch and complete transparent to the applications that generate the RTP traffic. The channel between *Compressor* and *Decompressor* do not have duplicate packets. The

Decompressor must receive packets in the same order as the *Compressor* sent them. The RoHC U-mode implementation is able to support multiple packet streams. For that uses a distinct CID to indentify each stream. The RoHC scheme has been designed to deal with residual errors in the headers delivered to the *Decompressor*. The CRCs and the sanity checks are used to prevent or reduce damage propagation.

This RoHC U-mode implementation uses a TAP device to send and receive the Ethernet frames generated by the local RTP applications. A TAP device is a Linux mechanism that enables the use of virtual interface. A virtual network can be viewed as a simple point-to-point or Ethernet device, which instead of receiving packets from a physical interface, receives them from a user space program. The packets are sent over a virtual interface that is associated to a physical interface. In the receiving machine, the received packets are delivered to a user space program [9].

Packet sockets can be used by user space applications to receive and send raw packets to network device drivers. The protocols family chosen was the PF_PACKET family. This family allows an application to send and receive packets dealing directly with the network card driver. The PF_PACKET family supports two different socket types: SOCK_DGRAM and SOCK_RAW. The SOCK_DGRAM leaves to the kernel the burden of adding and removing Ethernet headers. The SOCK_RAW gives the application complete control over the Ethernet header [10]. The RoHC U-mode implementation uses SOCK_RAW in the Packet Sockets maintained by the *Compressor* and *Decompressor* threads.

The RoHC packets are transported in Ethernet frame with the ethertype 0x8901. This ethertype was chosen so it would not to come into conflict with the existing erhertypes in the Linux Kernel. In the Figure 36 represents of a RoHC packet.

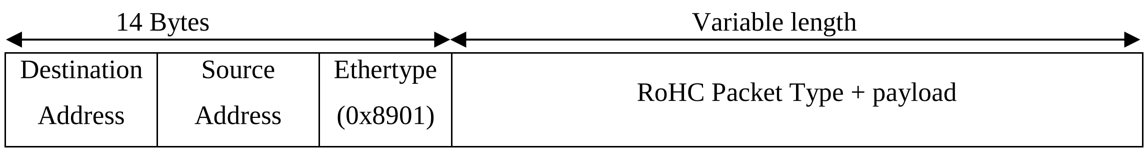


Figure 36- RoHC packet

4.2 Architecture

The architecture of the RoHC U-mode implementation is illustrated in Figure37, and it is organized as follows: each node runs the RoHC application that performs the role of RoHC *Compressor* and RoHC *Decompressor*. Two separate threads, called the *Compressor* and the *Decompressor*, perform the two roles. A TAP interface is created by the RoHC application when the application is launched. The TAP is used to intercept the RTP traffic generated by an application. The *Compressor* intercepts the traffic, coming from an application to the network, with a virtual Ethernet network device (rohc0). The *Compressor* thread also creates a Packet Socket to a local physical interface for the purpose of injecting compressed packets. The local interface is defined by command line parameter when launching the applications. The *Decompressor* thread also creates a Packet Socket for receiving RoHC packets.

For each flow of RTP packets, a *Context* is created and it is kept both in the *Compressor* and in the *Decompressor*. A CID is generated for each flow, which identifies each *Context* created. The first packets sent from the *Compressor* to the *Decompressor* are always sent with full headers, which include all the fields in uncompressed state plus the generated CID. This allows the *Decompressor* to create the *Context* for that flow. For the following packets, the *Compressor* will compress them by sending the dynamic fields and variations on the packets fields, omitting the static fields and other inferred fields. After the packet is compressed, it is injected into the RoHC channel using, the Packet Socket. In the RoHC channel, the RoHC packets re sent with a RoHC specific ethertype to the destination node. When the compress packet reaches the destination node, it is received in the *Decompressor* Packet Socket. The *Decompressor* thread checks if a *Context* for that CID already exists, creates it if not, decompresses the packet by rebuilding the original RTP packet. To accomplish that, the *Decompressor* thread uses the *Context* created for that flow to complete the packet and infer/calculate the missing fields. The decompressed packets are injected in the localhost interface that delivers them to the application. All the steps in this process are transparent to the application that generates the RTP traffic.

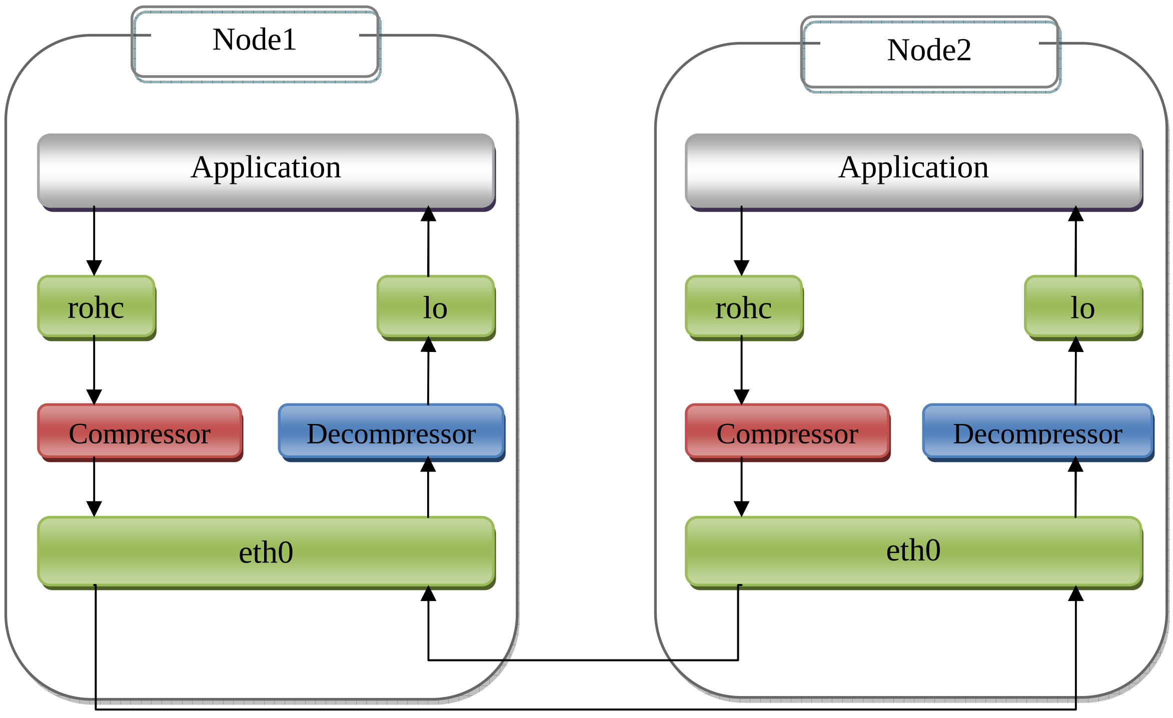


Figure 37- RoHC Architecture

4.3 Data structures

This section describes all of the data structures used in the RoHC U-mode implementation. The *struct node* represented in Table IV, is the base structure of the RoHC U-mode implementation it contains the variables negotiated in the negotiation phase. The RoHC application first initializes this structure by setting default values for the RoHC negotiation phase, since this phase is not supported by this implementation. This structure contains the IP address of the node (ipaddr), the highest CID number that may be used by the *Compressor* (max_cid). The max_cid value represents the capacity that both the *Compressor* and the *Decompressor* can support in term of memory resources. The host machine can support max_cid + 1 contexts. This context must be kept until it gets reuse or the RoHC channel is broken or renegotiated. The large_cids value indicates whether not short CIDs or large CIDs are used. The use of large CIDs implies the use of an additional 1 to 2 bytes in the RoHC packet to represent the CID. The profile value represents the profile supported by the *Compressor* and the *Decompressor*. The *Compressor* cannot compress a packet with a profile not supported by the *Decompressor*. The mrru value is largest reconstructed unit in octets that the *Decompressor* is can reassemble from segments. The enabled value indicates if the node is active and it is ready to send and received packets.

Field	**Type**	**Description**
Ipaddr	*struct in_addr*	The IP address of the node.
max_cid	int	Highest CID number that may be used by the *Compressor*.
large_cids	int	If false, the short CID is used, if true, the large CID.
profiles	int	Indicates a profile supported by the *Compressor* and *Decompressor*
Mrru	int	Maximum reconstructed reception unit.
enabled	int	Indicates if the node is active.

Table IV – struct node

The structure *struct contextCompressor* represented in Table V. This table contains the context for the *Compressor*. All the information necessary for the successful compression of a packet is kept in *Context*. The CID identifies the *Compressor* and *Decompressor Contexts*. The profile identifies the profile that is going to be used by the *Compressor* to compress the packets. For the RoHC U-mode implementation, the profile used was Profile 0x0001, for sending compressed IP/UDP/RTP headers. The c_mode identifies the mode of operation of RoHC used. In the this case the operation mode is the U-mode. Although this implementation is only limited to the U-mode, it is prepared for the change to other modes of operation. The c_mode value may take the following values: For the U-MODE the value one, the O-MODE the value two, and for the R-MODE the value three. The state identifies the state in which the *Compressor* is. This value may take the following values: For the IR state the value one, the FO state the value two, and for the SO state the value three. The DateCreated and DateModified are respectively the date of creation of the context and the date on which the context has changed. The rnd flag indicates whether the IP-ID behaves randomly. The rbo flag indicates if the IP-ID is in network byte order. The sport and the dport are the source and destination ports of the packet to be compressed. The ip_src and the ip_dst are the source and destination IP address of the packet to be compressed. The *Compressor* also keeps in its context the IP header and UDP header of the packet previous compressed (old_ip and old_udp). This helps the *Compressor* to detect changes in packets flow. The *Compressor Context* uses the *UmodeCompressor* structure to access the values of the U-Mode parameters: IR_TIMEOUT, FO_TIMEOUT and L-VAR. These values are initialized with default values. The rtp field corresponds to the RTP structure of the packet to be compressed. The stats structure contains the statistics for the *Compressor* context. The wlsb_sn, wlsb_ts, wlsb_ip_id structures are LSB structure for the SN, TS, and IP-ID fields of the packet to be compressed. These structures assist in the calculation of the LSB value send for the referred fields. The tmp_var contains temporary variables structure for the *Compressor Context.* These variables help the *Compressor* to make decisions. These decisions include which type of packet to send, if required which extension, and the change of state of the *Compressor*.

Field	**Type**	**Description**
cid	int	Context identifier.
Profile	int	Number of existing profile.
c_mode	mode_t	Operation modes.
State	c_state	States of the Compressor.
DateCreated	int	Date of creation.
DateModified	int	Date of modification.
Status	int	Status of the Context.
Rrnd	int	Flag indicating whether the IP-ID behaves randomly.
Rbo	int	Flag indicates if the IP-ID is in Network Byte Order.
Sport	u_short	Source port of the packet to be compressed.
Dport	u_short	Destination port of the packet to be compressed.
ip_src	*struct in_addr*	Source IP address of the packet to be compressed.
ip_dst	*struct in_addr*	Destination IP address of the packet to be compressed.
*old_ip	*struct iphdr2*	The IP header of the packet previous sent.
*old_udp	*struct udphdr2*	The UDP header of the packet previous sent.
*umode	struct UmodeCompressor	U-MODE structure for the Compressor context.
*rtp	*struct RTPCompressor*	RTP structure of the packet to be compressed.

*stats	*struct StatsCompressor*	Statistic structure for the Compressor context.
*wlsb_sn	*struct c_wlsb_sn*	LSB structure for the SN field of the packet to be compressed.
*wlsb_ts	*struct c_wlsb_ts*	LSB structure for the TS field of the packet to be compressed.
*wlsb_ip_id	*struct c_wlsb_ip_id*	LSB structure for the IP-ID field of the packet to be compressed.
*tmp_var	*struct RoHC_tmp_variables*	Temporary variables structure for the Compressor.
*next	*struct contextCompressor*	Next Context.

Table V – struct contextCompressor

The structure *struct contextDecompressor* represented in Table VI corresponds to the *Decompressor Context* and is very similar to the *Compressor Context*. The following text will describe the different fields. The d_mode indicates the mode of operation of the *Decompressor* as the *Compressor* is limited to the U-mode, and it can take the same values as the *Compressor*. The state identifies the state for *Decompressor* and it cans that the following values: for the NO state the value one, for the SC state the value two and for the FC the value three. The ip_id field of the packet to be decompressed is kept in the context in order to detect errors or in packet flow.

Field	**Type**	**Description**
cid	int	Context identifier
Profile	int	Number of existing profiles
d_mode	mode_t	Modes of operation
State	d_state	States of the Decompressor
DateCreated	int	Date of creation of the Decompressor context
DateModified	int	Date of modification of the Decompressor context
Status	int	Status of the Decompressor context
Sport	u_short	Source port of the packet to be decompressed
Dport	u_short	Destination port of the packet to be decompressed
Rnd	int	Flag indicating whether the IP-ID behaves randomly
Rbo	int	Flag indicates if the IP-ID is in Network Byte Order.
ip_id	int	IP_ID of the packet to be decompressed
ip_src	*struct in_addr*	Source IP address of the packet to be decompressed
ip_dst	*struct in_addr*	Destination IP address of the packet to be decompressed
*ip	*struct iphdr2*	The IP header of the packet of the packet to be decompressed
*udp	*struct udphdr2*	The UDP header of the packet of the packet to be decompressed
*rtp	*struct RTPDecompressor*	RTP structure of the packet to be decompressed

*wlsb_sn	*struct d_wlsb_sn*	LSB structure for the SN field of the packet to be compressed
*wlsb_ts	*struct d_wlsb_ts*	LSB structure for the TS field of the packet to be compressed
*wlsb_ip_id	*struct d_wlsb_ip_id*	LSB structure for the IP-ID field of the packet to be compressed
*next	*struct contextDecompressor*	Next Context

Table VI – struct contextDecompressor

The following structures are used to store the headers that form the headers of an RTP/UDP/IP packet encapsulated in an Ethernet packet. The first header is the Ethernet header, which is composed by the source host address (MAC address) field, destination host address (MAC address) field and the Ethernet type (ethertype). This header contains 14 bytes of information.

Field	**Type**	**Description**
ether_dhost[ETHER_ADDR_LEN]	u_char	Destination host address
ether_shost[ETHER_ADDR_LEN]	u_char	Source host address
ether_type	u_short	Ethernet type

Table VII – struct ethhdr2

The IPv4 packet header uses 20 bytes of information. The following structure holds the fields of the IPv4 header. The ip_vhl field represents the version and the *Internet Header Length* (IHL). The first four bits is version field, this has a value of four because it was used IPv4 packets. The IHL correspond to the next four bits. The ip_tos indicates the type of service this specifies how the packet should be handled as he is transported through a network. The ip_len indicates the total length of the packet, including header and data. The ip_id field is an identification field is used for identifying fragments of an original IP packet. The ip_off is the fragment-offset field, is 13 bits long and specifies the offset of a particular fragment relative to the beginning of the original IP packet. The IP_RF, IP_DF, IP_MF and IP_OFFMASK are flags used to control or identify fragments. The ip_ttl field limits the packet lifetime and is specified in seconds. The ip_p identifies the protocols used in the packet. The ip_sum is the checksum field used for error checking of the header. The ip_src field and the ip_dst field correspond respectively to the source and destination address of the packet.

Field	**Type**	**Description**
ip_vhl	u_char	version << 4 | header length >> 2
ip_tos	u_char	Type of service
ip_len	u_short	Total length

ip_id	u_short	IP identification
ip_off	u_short	Fragment offset field
IP_RF	int	Reserved fragment flag
IP_DF	int	Don’t fragment flag
IP_MF	int	More fragments flag
IP_OFFMASK	int	Mask for fragmenting bits
ip_ttl	u_char	Time to live
ip_p	u_char	Protocol
ip_sum	u_short	Checksum
struct in_addr	ip_src	Source address
struct in_addr	ip_dst	Destination address

Table VIII – struct iphdr2

The following structure represents the UDP header fields use in RoHC U-mode implementation. This header contains 8 bytes of information. This structure contains the source and dest fields that correspond to the source and destination ports of the packet. The len field is the length of the UDP header and data. The check field is checksum field.

Field	**Type**	**Description**
source	u_short	Source port
dest	u_short	Destination Port
len	u_short	Length
check	u_short	Checksum

Table IX – struct udphdr2

The following structure represents the RTP header fields used by the *Compressor Context*. The first field is the RTP version number. Follow by the padding, that if it has a value, the packet will contain additional padding bytes at the end, which are not part of the payload. The extension field it has a value the fixed header is followed an extension. The crsccount field identifiers the follow the fixed header. The marker field identifies the format of the RTP payload and determines its interpretation by the application. The payloadtype identifies the format of the RTP payload and determines its interpretation by the application. The seqnum is the RTP sequence number increments by one for each RTP data packet sent. The timestamp gives the sampling instant of the first byte in the packet and is used to remove jitter introduced by the network. The clock frequency depends on applications and it has a random initial value. The ssrc identifies the synchronization source. The value is chosen randomly. Within the same RTP session, will have different SSRC. The crsc is an array of 0 to 15 CSRC elements identifying the contributing sources for the payload contained in this packet. In addition to the header fields of the RTP header this structure also contains and the ts_tstride and ts_scale variables. These variables were added to this structure in order to facilitate the calculation of the value of RTP TS. The ts_stride field predicts increment/decrement of the RTP TS field when it changes. The ts_scaled is the integral multiple that communicated between *Compressor* and *Decompressor* by WLSB encoding.

Field	**Type**	**Description**
rtpversion	int	RTP Version
Padding	int	If set, this packet contains one or more additional padding bytes at the end which are not part of the payload
extension	int	If set the fixed header is followed by an extension.
crsccount	int	The number of CSRC identifiers that follow the fixed header.
Marker	int	Identifies the format of the RTP payload and determines its interpretation by the application.
payloadtype	int	Identifies the format of the RTP payload and determines its interpretation by the application.
Seqnum	int	The sequence number increments by one for each RTP data packet sent
timestamp	double	The timestamp reflects the sampling instant of the first octet in the RTP data packet
Ssrc	double	Identifies the synchronization source
Crsc	double	An array of 0 to 15 CSRC elements identifying the contributing sources for the payload contained in this packet.
ts_stride	int	Predicts increment/decrement of the RTP Timestamp field when it changes
ts_scaled	int	Is the integral multiple that communicated between *Compressor* and *Decompressor* by WLSB encoding

Table X – struct RTPCompressor

The following structure represents the RTP header fields used by the *Decompressor Context*. This structure is very similar to the one used by the *Compressor*. This structure contains the header fields of the RTP header and the variables use to calculate the value of the RTP TS. In addition to the variables used by the *Compressor* (ts_stride, ts_scaled), the *Decompressor* uses the ts_offset. The ts_offset is the linear offset. The ts_offset is implicitly communicated by sending several uncompressed TS values.

Field	**Type**	**Description**
rtpversion	int	RTP Version
Padding	int	If set, this packet contains one or more additional padding bytes at the end which are not part of the payload
extension	int	If contains a value different from zero, the fixed header is followed by exactly one header extension.
crsccount	int	The number of CSRC identifiers that follow the fixed header.

Marker	int	Identifies the format of the RTP payload and determines its interpretation by the application.
payloadtype	int	Identifies the format of the RTP payload and determines its interpretation by the application.
Seqnum	int	The sequence number increments by one for each RTP data packet sent
timestamp	double	The timestamp reflects the sampling instant of the first octet in the RTP data packet
Ssrc	double	Identifies the synchronization source
Crsc	double	An array of 0 to 15 CSRC elements identifying the contributing sources for the payload contained in this packet.
ts_stride	int	Predicts increment/decrement of the RTP Timestamp field when it changes
ts_scaled	int	Is the integral multiple that communicated between *Compressor* and *Decompressor* by WLSB encoding
ts_offset	int	TS OFFSET is the linear offset. TS OFFSET is implicitly communicated by sending several uncompressed TS values.

Table XI – struct RTPDecompressor

The *UmodeCompressor* structure contains the values of the maximum numbers of packets that can be sent in the IR state and in the FO state before changing to a higher compression state. It also contains the value of the timeouts that cause the *Compressor* to revert to lesser compression state. These timeouts are defined as a number of packets sent. The values of these parameters are relevant to the RoHC performance in terms of compression efficiency in the U-Mode.

Field	**Type**	**Description**
max_ir_count	int	Maximum number of packet send in the IR state
max_fo_count	int	Maximum number of packet send in the FO state
IRtimeout	int	IR Timeout
FOtimeout	int	FO Timeout

Table XII – struct UmodeCompressor

The structure *StatsCompressor* represented in Table XIII. This structure contains counters that hold the number of packets sent in the different U-Mode states by the *Compressor*. Also, include the values for the timeouts in the different states.

Field	**Type**	**Description**
num_send_packets	int	Number of packets sent
num_send_ir_packets	int	Number of packets sent in the IR state
num_send_fo_packets	int	Number of packets sent in the FO state
num_send_so_packets	int	Number of packets sent in the SO state

num_send_ir_dyn_packets	int	Number of IR-DYN packets sent
timeout_ir	int	IR timeout
timeout_fo	int	FO timeout

Table XIII –struct StatsCompressor

The structure *StatsDecompressor* represented in Table XIV. This structure contains the counters for the *Decompressor*. Theses counters hold the number of packets arrived in *Decompressor*.

Field	**Type**	**Description**
num_recv_packets	int	Number of packets arrived in *Decompressor*
num_recv_ir_packets	int	Number of IR packets arrived in *Decompressor*
num_recv_ir_dyn_packets	int	Number of IR-DYN packets arrived in *Decompressor*

Table XIV –StatsDecompressor

The *RoHC_tmp_variables* structure is used by the *Compressor* to make decisions based on the conditions of the RoHC flow. These include the decision of what type packet to send, whether or not to send an extension, and what type of extension to send. Every time a packet arrives to the *Compressor* these variables are updated.

The *RoHC_tmp_variables* structure is holds the following fields: the num_of_ip_hdrs field contains the number of IP headers that the packet has. The changed_fields field indicates if the packet has any changes in it fields. The send_static field indicates if the *Compressor* has to send the static part of the *Context*. The send_dynamic field indicates if the *Compressor* has to send the dynamic part of the *Context*. The fields: nr_ip_id_bits, nr_sn_bits and nr_ts_bits contains the number of bits of the IP-ID fields, SN, and TS respectively. The packet_type field indicates the RoHC packet type send to the *Compressor* is sends to the *Decompressor*. The s, RTS, and i fields indicates if the SN, TS and IP_ID fields are encoded in the extension. The tsc field indicates that TS is scaled. The reaming fields of field of this structure are flags that also indicate the presence of certain fields in the inner IP header fields.

Field	**Type**	**Description**
num_of_ip_hdrs	int	Number of IP Headers
changed_fields	int	If there was any change in the fields
send_static	int	Indicates if the *Compressor* has to send the static part of the *Context*
send_dynamic	int	Indicates if the *Compressor* has to send the dynamic part of the *Context*
nr_ip_id_bits	int	Number of bit of the IP-ID
nr_sn_bits	int	Number of bit of the SN
packet_type	int	Packet Type

nr_ts_bits	int	Number of bit of the TS
S	int	Indicates if the SN is encoded in the extension
RTS	int	Indicates if the TS is encoded in the extension
Tsc	int	Indicates that TS is scaled
I	int	Indicates the presence of IP-ID in the extension
Ip	int	Indicates the presence of Inner IP header fields in the extension
Rtp	int	Indicates the presence of RTP header flags and fields in the extension
Tos	int	Indicates presence of field Type of Service/Traffic Class in inner IP header fields
Ttl	int	Indicates presence of field Time to Live/Hop Limit in inner IP header fields
Df	int	Don't Fragment bit of IP header.
Pr	int	Indicates presence of field Protocol/Next Header in inner IP header fields
Ipx	int	Indicates presence of field IP extension headers in inner IP header fields
Nbo	int	Indicates whether the octets of header (IP identifier) of this IP header are swapped before compression and after decompression. NBO = 1 indicates that the octets need not be swapped. NBO = 0 indicates that the octets are to be swapped.
Rnd	int	Indicates whether header (IP identifier) is not to be compressed but instead sent as-is in compressed headers.

Table XV – RoHC_tmp_variables

In order to encode the LSB value of SN, TS, and IP-ID fields the following structures represented in the Table XVI and Table XVII. Both the *Compressor* and the *Decompressor* threads use them. The c_wlsb_** holds the windowWidth field that has the size of the sliding window. The bits field that holds number of maximum bits for representing a value. The c_window_** holds the entries for the sliding window. These entries have the original value for the fields SN, IP-ID and TS. The calculated LSB values for which original value.

Field	**Type**	**Description**
windowWidth	int	Size of the window
*window	struct c_window_sn	windowWidth number of c_window_**
Bits	int	Number of bits
Next	int	keeps track of the current position in the window

** - Replace by SN, TS, and IP-ID

*Table XVI – c_wlsb_***

Field	**Type**	**Description**
**	Int	Value of SN, TS or IP-ID
Value	Int	LSB value
Used	Int	If is used

** - Replace by SN, TS, and IP-ID

*Table XVII – c_window_***

4.4 Implementation Features

4.4.1 Compressor

The negotiation phase of this RoHC U-mode implementation consists on serial of predetermined values and initialization functions. The actually negotiation between the *Compressor* and *Decompressor* was not implemented. After this initialization phase, the *Compressor* and the *Decompressor* are ready to send and receive packets. When the *Compressor* receives, a packet it checks if an entree exists in the *Compressor* context for the packet. If there is no entree in the context for that packet, it will be created. If exists an entree in the *Compressor* context, the packet is checked for changes in the static and dynamic parts of the packet headers. The process that checks for changes in packet headers also updates the temporary variables. The temporary variables combined with the state of the *Compressor* determined, which type of packet and extensions to send to the *Decompressor*.

When the *Compressor* starts, he always starts in the IR state. In this state, all packets sent to the *Decompressor*, are IR packets. The IR packets are used to initializes or to refresh the static and dynamic parts of the context in the *Decompressor*. The *Compressor* stays in this state until it sends a predetermined number of packets. This parameter is call the L-variable (L_VAR), and although the RoHC standard [1] does not give a define value. For this implementation, one was estimated. When the *Compressor* reaches this value, the state is changed to the FO state.

In the FO state, the *Compressor* can send two types of packets the IR-DYN packets or UOR-2 packets depending on the temporary variables, at that particular moment. The packets send in this state serve to communicate irregularities in the packet stream therefore the big majority of information send corresponds to dynamic part of the context and only a few static fields are updated. The upward transition for this state is similar to the IR state. When the *Compressor* send a predetermined number of packets it changes state to the SO state.

The SO state is the higher level of the compression, in this state the *Compressor* can send tree types of different packets UO-0, UO-1, UO-2. In this state as in the previous states, the decision between the packets depends on the temporary variables.

If the packet types send, are UO-1-IP-ID, UO-2-IP-ID or UO-2-TS, these packets types can take extensions. These extensions take the extra bits from SN, TS and IP-ID and provide other type of information. The decision of witch extensions is send depends in what field or fields as the extra bits and what type of information it is needed to be sent. When the decision of what packet type and extension to send. The only thing left to do is to code the packet accordantly and send it to the *Decompressor*. Then the statistics for the *Compressor* is updated. (See A2 for the diagram of this feature)

4.4.2 Type of packet to send in FO

The decision of packet type to send in the FO state relays in two items:

- Which part (dynamic or static) of the context does the *Compressor* needs to send to the *Decompressor*;
- The number of packets it has to send in the FO state.

Depending of the values of these two items the *Compressor* decides which packet to send an IR-DYN or UOR-2-ID type of packet. The *Compressor* starts by checking if all packets in this state have been send. If not, it checks if it has to send the static part of the *Context* if true the *Compressor* has to send a UOR-2-ID packet type and the *Compressor* as to change state to the IR state. If the *Compressor* does have to send the static part of the context and it has to send the dynamic part, the *Compressor* will send an IR-DYN packet type. If the *Compressor* does not have to send neither the static nor the dynamic parts, it will decide to send a UO2 packet type (See A3 for the diagram of this feature).

4.4.3 Type of packet to send SO

Decision type of packet to send in SO state, relies mainly in:

- The number of bits of SN, IP-ID and TS;
- The value attributed to the flag RND;
- The number of packets it has to send in SO.

Depending of the values of these items the *Compressor* decides which packet to send: UO-2-TS, UO-2-IP-ID, or UO-0.

The *Compressor* starts by checking if flag RND equals one and if the number of bits of SN is less or equal to four or if the number of bits of SN less or equal to four and the number of bits of IP-ID and TS is zero, the *Compressor* send the UO-0 packet type. If these conditions are not true and the number of bits of SN less or equal to five and number of bits of IP-ID is zero and if the number of bits of TS is zero the UO-2-IP-ID packet type is send, if the number of bits of TS is not zero the UO-2-TS packet type is send. If the number of bits of SN is less or equal to five and the number of bits of IP-ID is less or equal to six and if the number of bits of TS is less or equal to six the UO-1-IP-ID packet type is send. If the number of bits of TS is higher than six the UO-2-TS packet type is send. If the number of bits of SN is less or equal to five and the number of bits of TS is less or equals to six the UO-2-TS packet type is send. If these conditions are not true, the UO-1-IP-ID packet type is send. (See A4 for the diagram of this feature)

4.4.4 Decide extensions

The extensions help to carry extra bits of certain fields such as SN, IP-ID and TS, and other bits of important information to update the flow context. The decision on what extension the packet type will take depends on the fowling items:

- Which part (dynamic or static) of the context does the *Compressor* needs to send;
- The number of bits of the SN, IP-ID;
- The value attributed to the flag RND.

If the *Compressor* as to send the static and the dynamic parts of the *Context* or if the number of bits of the SN is higher than eight and the number of bits of IP-ID it is higher than eleven, it has to send an extension 3. If number of bits of SN is less than five and the number of bits of the IP-ID equals zero or RND flag equals one no extension is required to send. If number of bits of SN is less or equal to eight and the number of bits of the IP-ID is less or equal to three an extension 0 is sent. If number of bits of SN is less or equal to eight and the number of bits of the IP-ID is less or equal to eleven an extension 1 is chosen.

4.4.5 Change of state

The logic transition of states is based in three principles: the optimistic approach principle, timeouts and the need for updates [1]. They are two types of transition of state in the *Compressor*. The upward transition is caused by each state sending a predetermine number of packets. The downward transition is caused by a series of timeouts or small variations in the packet stream that causes the *Compressor* go back to previous state so it can keep the *Context* synchronized.

The *Compressor* always starts in the IR state it stays in this state until it send a predetermine number of packets. After this, it will change to the FO state. In the FO state, the *Compressor* checks if the IR timeout equals zero, this will cause the *Compressor* to change back to the IR state. If not, it checks if it has sent the maximum number of packets for this state. If not it, checks if it has to send the static part of the *Context*, if it has this will cause the *Compressor* to stay in the FO state and reset all its variables. When the *Compressor* reaches maximum number of packets for this state, it will change to the SO state. In the SO state, it will check if any of the timeouts have been reached (the IR timeouts and FO timeouts) if true the *Compressor* will change state. Then it checks if it has to send the dynamic or static part of *Context*, if true this will caused the *Compressor* to change state to the FO state, if not it will remain in the SO state until one of the timeouts causes a change of state.

4.4.6 Decompressor

The *Decompressor* receives the RoHC packets sent and recreates the original packets and delivered them to the application level.

When the *Decompressor* receives a packet, it checks if an entree exists in the *Decompressor Context* for that flow. If there is no entree in the *Context* for that flow, it will be created. Next, the *Decompressor* determines what type of packet it was send by the *Compressor*. The type of packet combined with the state of the *Decompressor* will determine how the packet will be decompressed or discarded due to unsynchronized *Context*.

The *Decompressor* has three states of operation: the NO state, the SC state and the FC state. The *Decompressor* always starts in the NO state, in this state only IR packets can be received. Any other types of packets are discarded. The first time that the *Decompressor* decompresses a packet correctly, it enters in the FC state, the *Decompressor* remains in this state it will occur repeated failures. When this happens it will go back to SC to try to recover the *Context* if that does not happen, the *Decompressor* enters in the NO state.

When the *Decompressor* decompresses a packet, it has to recreate the original packet headers (Ethernet headers, IP header, UDP headers, and RTP headers) for that is uses the *Context* of the *Decompressor* and calculates the missing fields such as the IP and UDP checksums. With the original packet reconstructed the *Decompressor* it delivers the packet to the upper layers.

4.5 Conclusions

This chapter approaches the details of the RoHC implementation; it includes requirements, architecture and implementation aspects. This implementation of the RoHC standard [1] is focused on the Profile 0x0001, for sending compressed IP/UDP/RTP packets. The RoHC implementation was developed in user space. This implementation is transparent to the user that would permit the compression of all RTP traffic in order to improve bandwidth performance over low bandwidth links.

The architecture of the RoHC U-mode implementation is organized as follows: each node runs the RoHC application that performs the two roles of RoHC Compressor and RoHC *Decompressor*. The role of the *Compressor* is to intercept the traffic, coming from an application to the network, and to compress the packet and inject it in the RoHC channel. For each flow of RTP packets, a *Context* is created and it is kept both in the *Compressor* and in the *Decompressor*. A CID identifies each *Context*. The RoHC packets are sent with a RoHC specific ethertype to the destination node. When the compressed packet reaches the destination node, it is received in the *Decompressor*. The *Decompressor* check if a *Context* for that CID already exists, creates it if not, and decompresses the packet by rebuilding the original RTP. The decompressed packets are injected in the localhost interface that delivers them to the application. All the steps in this process are transparent to the application.

This chapter also describes the data structures used in this RoHC U-mode implementation, and the features of this RoHC U-mode implementation. These features include the description of the *Compressor* and of the *Decompressor* functionality.

5 RoHC Implementation Tests

This chapter presents the functional and performance tests carried out with the RoHC U-mode implementation and discuss the results obtained.

In order to test the RoHC U-mode implementation two test scenarios were set up. The first, named fixed scenario test consists of two terminals connected by an IEEE 802.3 switch; the second, named wireless scenario test, consists of two terminals connected to an AP an IEEE 802.11 network using an AP. In both tests, each terminal runs a *Compressor* and a *Decompressor* instance.

The purpose of the functional tests is to validate the functional requirements. The first set of tests consists in establishing a VoIP call between two nodes, with and without RoHC, in the test scenarios. The second set of tests consist in randomly introducing a packet drop, to verify if in U-Mode the *Decompressor* is able to recover from the loss propagation.

The performance tests measure the performance of RoHC using a metric called *RoHC Gain* [35]. The performance tests were made using only the wireless scenario, and consist in establishing VoIP calls and a TCP flow as background load traffic. The TCP flow is configured to always have data to send. Therefore, the TCP congestion control mechanism tries to adjust the TCP flow bit-rate to the bandwidth available in the wireless medium.

5.1 VoIP call parameters

To establish the VoIP calls it was used the packet generator *packETH* [13] was. It was configured to send packets using the voice codec G.729 [39]. The G.729 was preferred because it generates constant bit-rate (CBR) traffic of 8 Kbit/s and offers a voice quality equivalent to the *Public Switch Telephone Network (*PSTN). In addition, this codec has a small payload size (20 bytes) and low CPU usage.

Using IPv4 at the network layer and using UDP/RT P headers, the total uncompressed size of the VoIP packets is 74 bytes. 14 for the Ethernet header, plus 20 bytes for the IPv4 header, 8 bytes for the UDP header, 12 bytes for the RTP header and 20 bytes of VoIP payload. The call time used in all tests was approximately 5 minutes.

5.2 RoHC parameters

The RoHC U-Mode parameters are the IR_TIMEOUT, the FO_TIMEOUT and the L-VAR. The RoHC standard does not define values for these parameters. Therefore, the values used for these parameters are based on those used in [6] [8]. These values were chosen as reference because the most nearly of our test conditions. The values used for the timeout variables were defined in number of packets. It was defined in number of packets instead of using time for these variables to simplify the development of the RoHC U-mode implementation. For the IR_TIMEOUT it was used a value of 250 packets, which in these tests correspond approximately to 5 seconds of VoIP traffic; for the FO_TIMEOUT it was used a value of 150 packets, which in these tests correspond approximately to 3 seconds of VoIP traffic. For the L_VAR a value of two packets was used.

5.3 Delay and Jitter

The transmission delay of a packet is defined as the time since the packet is queued in the sender node until it reaches the destination node. The average delay of a stream is calculated by dividing the sum of the delay of each packet of the stream by the number of packets. Jitter is defined in [14] as the absolute value of the difference between the delay of two consecutive packets of a same stream. The jitter is calculated according to [22].

Thus, the *Delay* (D) is calculated using the following equation:

$$D_{(i,i+1)} = T_{i+1} - T_i$$

The Jitter can be calculated using the following equation:

$$J = D_{(i+1)} - D_i.$$

The *Average Delay* (AD) and the *Average Jitter* (AJ) can be calculated using the following equations:

$$AD_{(stream)} = \frac{\sum_{i=1}^{n} D_{(i)}}{n}$$

$$AJ_{(stream)} = \frac{\sum_{i=1}^{n} J_{(i)}}{n}.$$

In the specific case of the RTP traffic, the Delay calculation can use the RTP timestamp as reception time. Thus, the Delay can be calculated using the equation:

$$\mathrm{D}_{(i,j)} = (\mathrm{R}_j - \mathrm{R}_i) - (\mathrm{S}_j - \mathrm{S}_i) = (\mathrm{R}_j - \mathrm{S}_j) - (\mathrm{R}_i - \mathrm{S}_i)$$

S -RTP timestamp
i- Packet number,
j- Number of consecutive packet

R - The time of arrival in RTP timestamp
D – Difference

The ITU recommendation G.114 [41] addresses acceptable delays for voice applications. This recommendation defines three delay ranges as shown in Table XVIII.

Range in Milliseconds	Description
0-150	Acceptable for most user applications.
150-400	Acceptable provided some considerations
Above 400	Unacceptable for most user applications.

Table XVIII – Delay

5.4 Functional Tests

To test the RoHC U-mode implementation two test scenarios were set up. The first, called fixed scenario test consists of two terminals connected by an IEEE 802.3 switch, represented in Figure 38. The second scenario, called wireless scenario test consists of two terminals connected to an AP an IEEE 802.11 network represented in Figure 39. In both scenarios, each terminal runs a *Compressor* and a *Decompressor* instances.

The functional tests validate the functional requirements. Two set of tests were performed. The first set using the fixed scenario and the wireless scenario. The second set of tests is equivalent to the first set except packet loss is intentionally introduced to verify if the U-mode recovery mechanisms are working as specified.

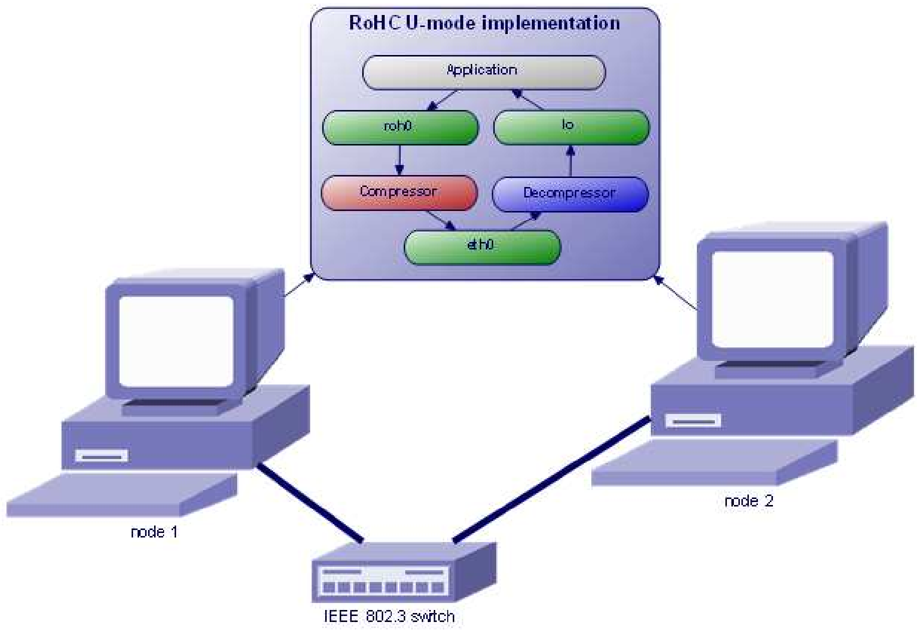


Figure 38- Fixed Scenario

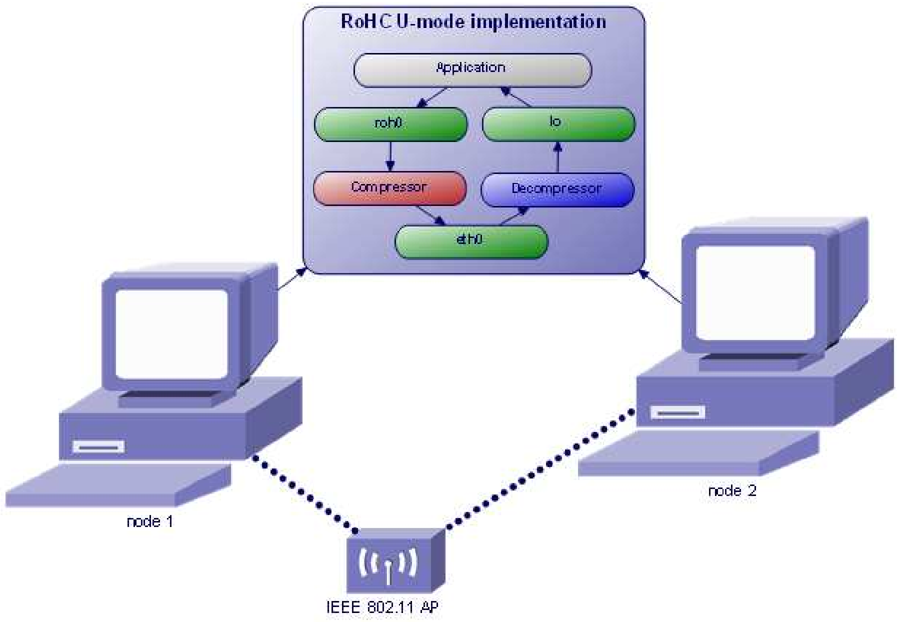


Figure 39- Wireless Scenario

5.4.1 Fixed Scenario tests

This test consists in establishing a single VoIP call between node1 and node2 using the fixed scenario, testing the use of the RoHC U-mode implementation and compares it with the results obtained when RoHC is not used. The VoIP traffic was collected using the packet sniffer *Wireshark* [40]. The Table XVIII resumes the information gathered from the packet capture. The total number of packets exchanged between the two nodes was 20000. The packets had a size of 74 bytes, exchange during a period of 5 minutes. This corresponds to an average of 33 packets/second, or an average of 2467 bytes/second.

	Node 1		**Node2**	
	Sent	**Received**	**Sent**	**Received**
Number of packets send	10000	10000	10000	10000
Time between the first and last packet (seconds)	300	300	300	300
Average packets/second	33,34	33,33	33,33	33,33
Average packet size	74	74	74	74
Total bytes	740000	740000	740000	740000
Average bytes/second	2466,83	2466,73	2466,74	2466,85

Table XIX – Fixed scenario test without using RoHC

The test was repeated but using RoHC to compress the headers of the VoIP flow. The same traffic measurements were performed, and are presented in the tables XX and XXI. As previously, 20000 packets were exchanged between the nodes, with an uncompressed size of

74 bytes, during a period of approximately of 5 minutes. An average packet rate of 32 packets/second was observed. The VoIP packets produced by the *packETH* were header compressed using RoHC, resulting in an average packet size near 37 bytes. Before applying RoHC, the packets had 54 bytes of headers after the using RoHC the packets had on average for 17 bytes of headers. When using RoHC, it was transmitted 370392 bytes less (740000-369608). In both nodes, there is a discrepancy between the packets sent and received this is due to padding added to the smaller packets before sending. The Ethernet card that cannot provide an efficient service for very small packets, so in order to correct this it adds the padding to the packets. The RoHC U-mode implementation does not support padding, therefore the total bytes of the decompressed packets it is larger than the ones send.

	Node 1			
	Packets sent by the application	**Packets Compressed sent (RoHC)**	**Packets Compressed received (RoHC)**	**Packets Decompressed**
Number of packets send	10000	10000	10000	10000
Time between the first and last packet (seconds)	312,47	319	317,77	319,18
Average packets/ second	32	31,3	31,47	31,33
Average packet size (bytes)	74	36,96	60,19	97,27
Total bytes (bytes)	740000	369608	601920	972730
Average bytes/ second	2368,21	1158,36	1894,18	3047,56

Table XX – Fixed scenario Test using RoHC - Node 1

	Node2			
	Packets sent by the application	**Packets Compressed sent (RoHC)**	**Packets Compressed received (RoHC)**	**Packets Decompressed**
Number of packets send	10000	10000	10000	10000
Time between the first and last packet (seconds)	311,14	317,77	319,1	319,09
Average packets/ second	32,14	31,46	31,34	31,33
Average packet size (bytes)	74	36,96	60,19	97,27
Total bytes (bytes)	740000	369608	601920	972730
Average bytes/ second	2378,12	1163,12	1886,42	3048,43

Table XXI – Fixed scenario Test using RoHC - Node 2

In Figure 40, was observed the U-mode state transitions over a period of 15 seconds and its relation with the packet length. The packets lengths vary from 80 bytes (60 bytes of headers) in the IR state, 62 bytes (42 bytes of headers) in the FO state and 36 bytes (16 bytes of headers) in the SO state. Because some packets are using Ethernet padding, the size of some packets is increased to 60 bytes.

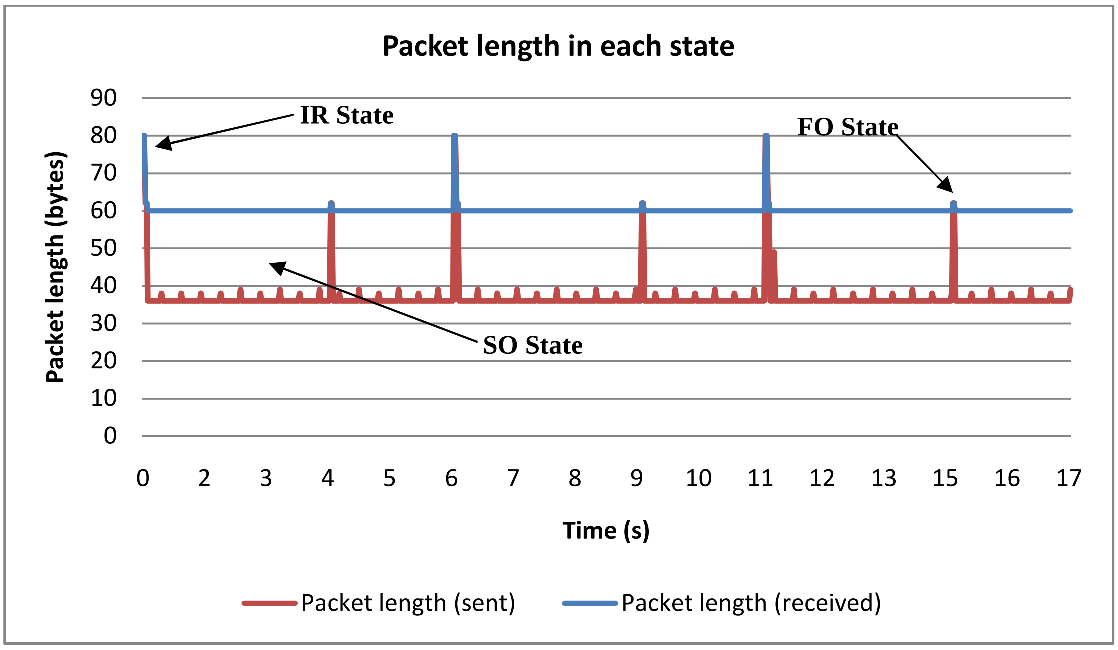


Figure 40-Packet length – Fixed scenario with RoHC

The percentages of packets sent in each U-mode state are represented in Figure 41. Only 1% of the packets are sent with full headers corresponding to the IR state, 51% of the packets send the dynamic part of the context corresponding to the FO state and 48% of the packets send fully compressed headers which correspond to the SO state.

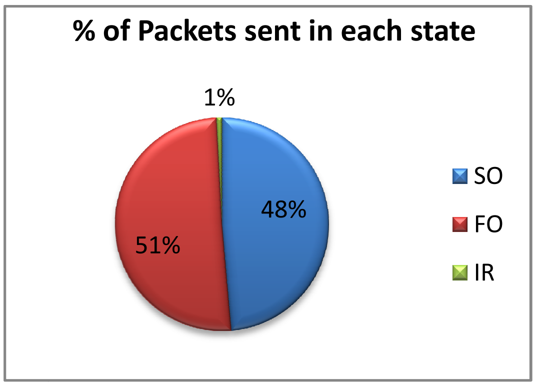


Figure 41- Packets sent in each state

Figure 42 presents the bit-rate of the VoIP stream with and without RoHC in the fixed scenario. To calculate the bit-rate it was only considered L3 headers, payloads and padding. The bit-rate of the VoIP stream not using RoHC is in average 8 kbps. When applying RoHC to the VoIP headers, the average bit-rate drops to 4.6 kbps. This amounts to 43% reduction.

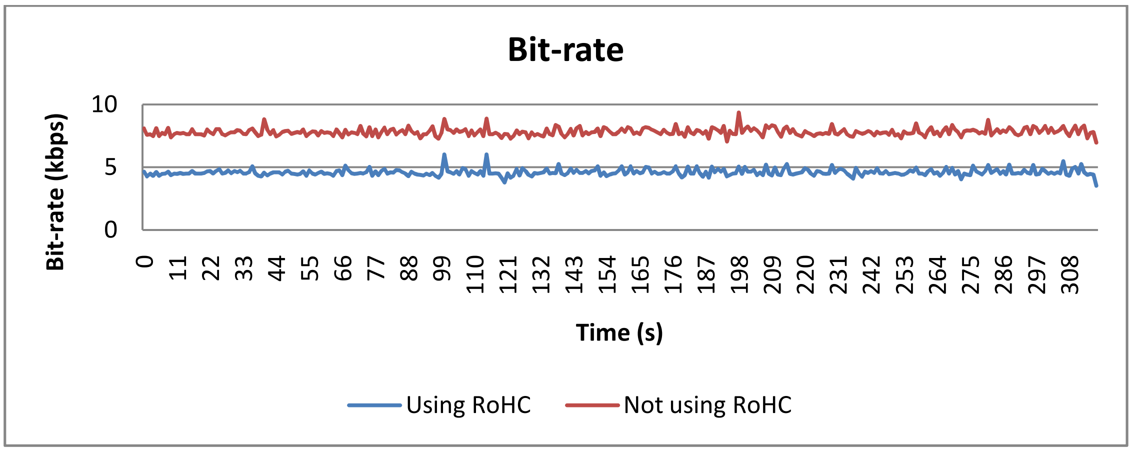


Figure 42- VoIP stream bit-rate in the fixed scenario

The delay introduced by the use of RoHC is presented in Figure 43. The delay without the application of RoHC remains constant at 30 ms during the duration of the transmission. The delay introduced by the application of RoHC has several fluctuations around 30 ms line. These fluctuations can be explained by the delay introduced by the application of RoHC to packets. For example, the smaller RoHC packets are sent in the SO state and easier to decompress, this introduce the smallest delay lowering the delay values up to 25 ms, in average. While the larger RoHC packets, in particularly those that carry extensions introduce the highest delay that can reach the 144 ms. Nevertheless, all packet delay measures were under the 150 ms, the upper bound value on the acceptable delay as recommended by [41].

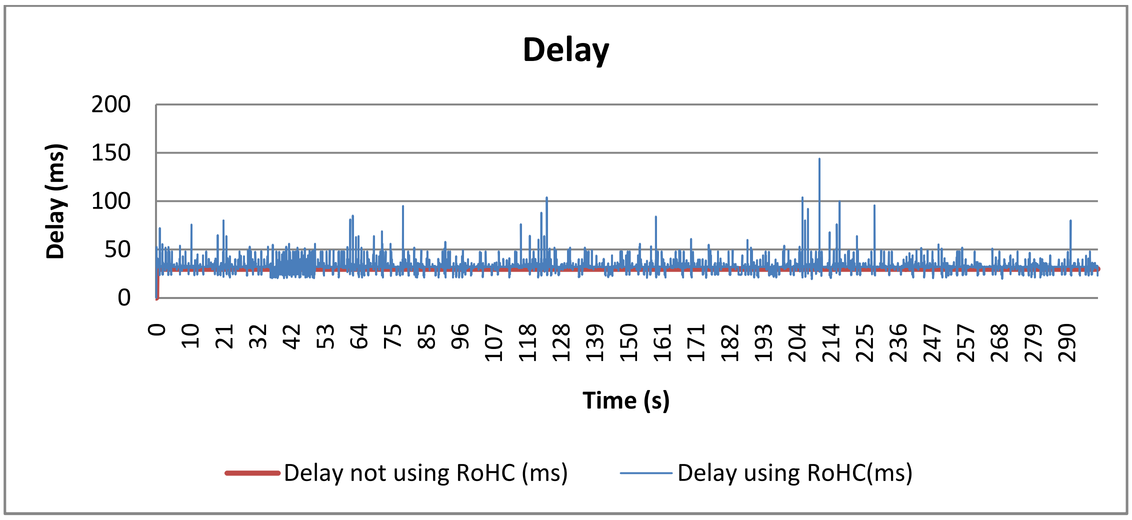


Figure 43- Delay in the fixed scenario

Figure 44 presents the jitter measured in the fixed scenario. The jitter measured when RoHC is not used is constant at 20 ms. The application of RoHC does not worsen significantly the jitter and in some cases, there is a little improvement reaching values of 15ms.

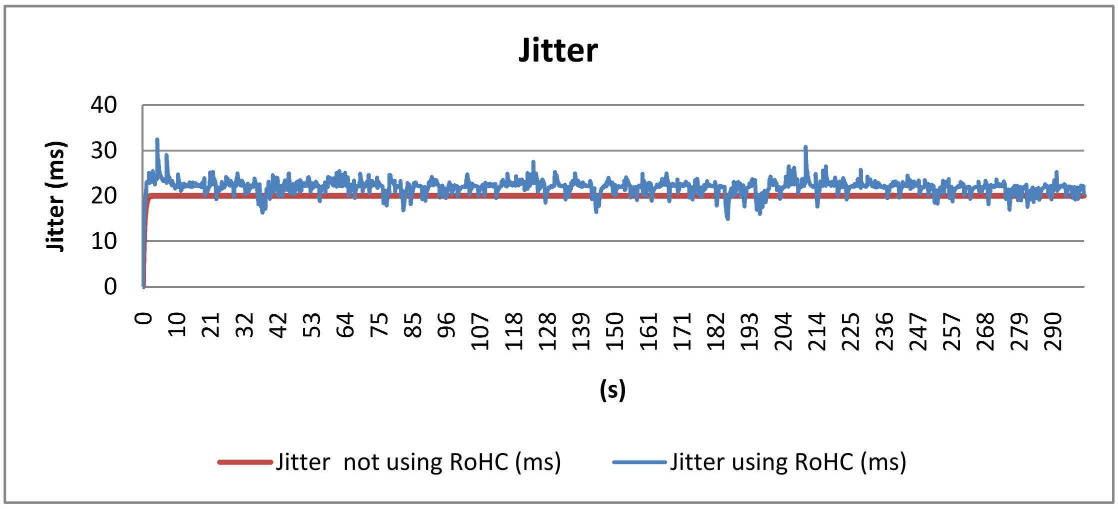


Figure 44- Jitter in the fixed scenario

5.4.2 Wireless scenario tests

The same tests were conducted using the wireless scenario. Again, a single VoIP stream was established between node 1 and node 2, and the tests were repeated using and not using RoHC. Table XXII resumes the traffic statistics gathered from the packet capture without using RoHC. The total number of packets exchanged between the two nodes during a period of approximately 5 minutes was 19988 packets. The packets had an average packet size of 74 bytes (54 bytes of headers). This corresponds to an average of 33 packets/second, or an average of 2466 bytes/second.

	Node 1		**Node2**	
	Packets sent	**Packets received**	**Packets sent**	**Packets received**
Number of packets send	9994	9994	9994	9994
Time between the first and last packet	300	300	300	300
Average packets/second	33,34	33,32	33,34	33,32
Average packet size	74	74	74	74
Total bytes	739556	739556	739556	739556
Average bytes/second	2466,77	2465,30	2466,78	2465,79

Table XXII – Wireless scenario test without using RoHC

In the tables XXIII and XXIV, it is presented the traffic statistics gathered from the packet capture of the wireless scenario test when using RoHC. The total number of packets exchanged between the two nodes was 19988. The packets had a size of 74 bytes, exchange during a period of 5 minutes. This corresponds to an average of 31 packets/second.

	Node 1			
	Packets sent by the application	**Packets Compressed sent (RoHC)**	**Packets Compressed received (RoHC)**	**Packets Decompressed**
Number of packets send	9994	9994	9994	9994
Time between the first and last packet (seconds)	314	320	320	320
Average packets/ second	31,78	31,18	31,21	31,12
Average packet size (bytes)	74	36,96	36,96	74
Total bytes (bytes)	739556	369608	369608	739556
Average bytes/ second	2351,56	1152,39	1153,65	2304,35

Table XXIII – Test statistics with RoHC wireless scenario node 1

	Node2			
	Packets sent by the application	**Packets Compressed sent (RoHC)**	**Packets Compressed received (RoHC)**	**Packets Decompressed**
Number of packets send	9994	9994	9994	9994
Time between the first and last packet (seconds)	311	320	320	320
Average packets/ second	32,06	31,22	31,17	31,16
Average packet size (bytes)	74	36,96	36,96	74
Total bytes (bytes)	739556	369608	369608	739556
Average bytes/ second	2372,42	1154	1152,06	2307,15

Table XXIV – Test statistics with RoHC wireless scenario node

Figure 45 presents the bit-rate of the VoIP stream with and without using RoHC in the wireless scenario. To calculate the bit-rate it was only considered L3 headers and payloads. The bit-rate of the VoIP stream without using RoHC is an average of 8 kbps as in the fixed scenario. With the use of RoHC, the bit-rate drops to an average of 2.2 kbps. This amounts to a 72 % of compression. In this scenario, the used bit-rate for the VoIP stream with RoHC is less than in the fixed scenario. This is explained by the fact that in the IEEE 802.11 there is no minimum frame size, and therefore, no padding is added to the frame as in the case of the IEEE 802.3.

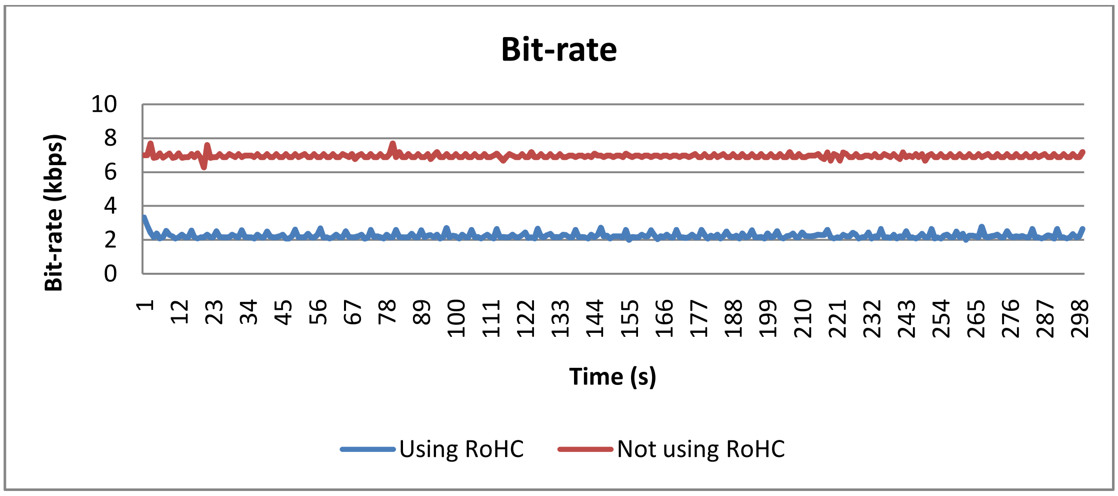


Figure 45- VoIP stream bandwidth in the wireless scenario

The delay introduced by the use of RoHC in the wireless scenario is lower than in the fixed scenario. This can explain by the no addition of padding to the packets in the wireless scenario. This makes the packets smaller and easier to decompress by the *Decompressor* thus introducing less delay. In this case, all delay values are below 70 ms. In some cases, using RoHC even lowers the delay values up to 18 ms well below the 30 ms registered delay when RoHC is not used.

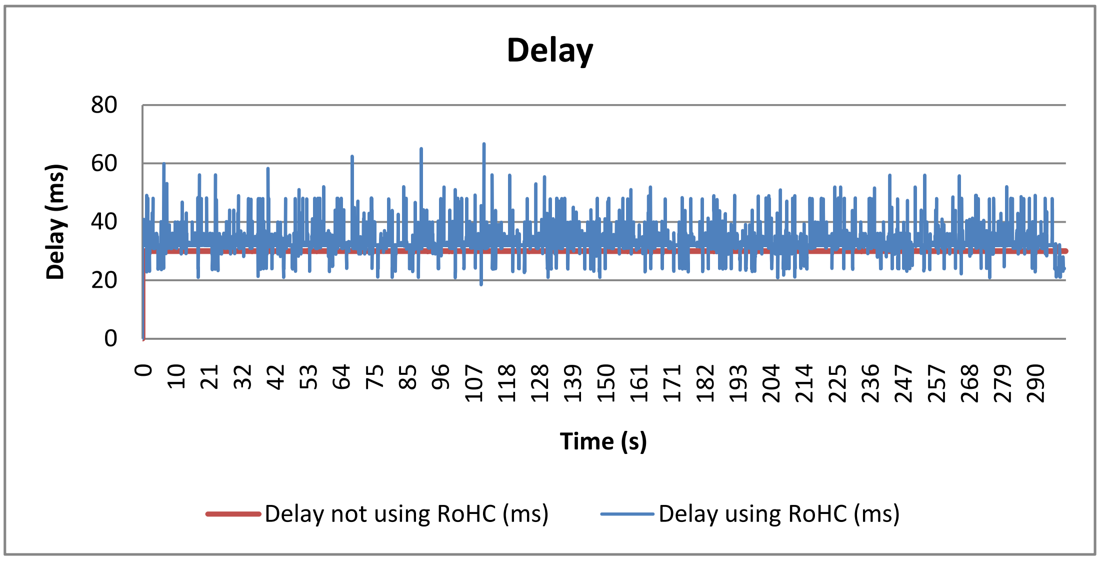


Figure 46- Delay in the wireless scenario

As the tests performed in the fixed scenario, the values of jitter are not significantly worse by the application of RoHC. All values of jitter measured applying RoHC, are below the 32ms.

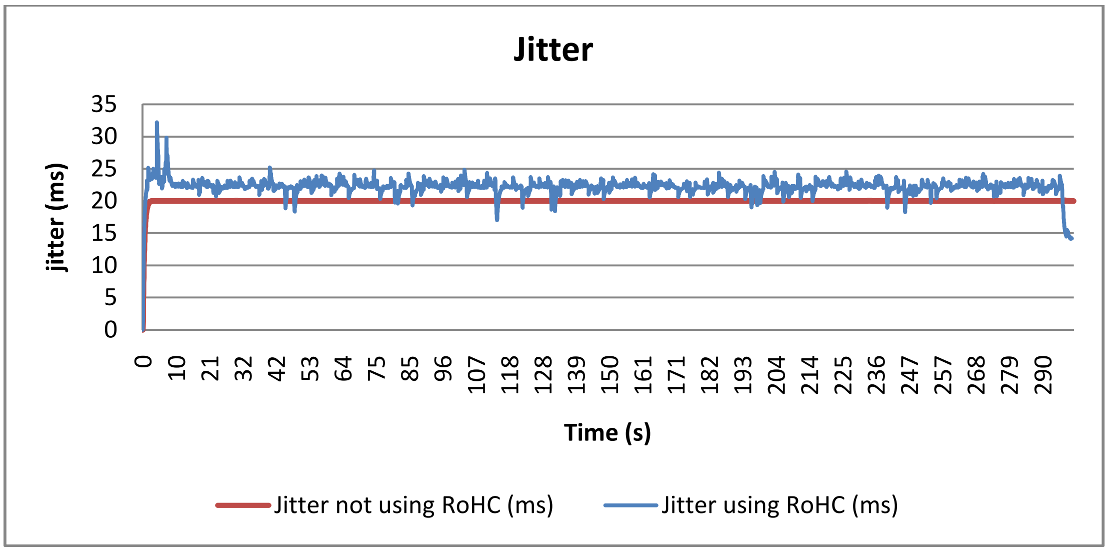


Figure 47- Jitter in the wireless scenario

5.4.3 Recover after failure test

In a RoHC stream, if a packet is lost or contains errors, the *Decompressor* will not be able to update the RoHC context correctly. This can cause the following RoHC packets to be incorrectly decompressed or dropped by the *Decompressor*, thus resulting in a loss propagation event. In these cases, the *Decompressor* has to be able to recover from an invalid context after receiving RoHC packets sent in the IR state or in some cases in the FO state. The goal of this test is to inject a flaw and observe if after a time interval the *Decompressor* is able to recover from an invalid context. In this test is simulated the loss of packets by using a new routine added to the *Compressor* code, that would discard packets in key points of RoHC state machine. Four test cases were tested and are presented in Table XXVII. First test case consisted in the loss of the initial IR packets. The result was the *Decompressor* discarded all packets until it received IR packets in order to create the context for that flow. The test resulted in 3% packets loss. The second test case consisted in the loss of the initial IR-DYN packets. This caused the *Decompressor* to fail to decompress the following packets, since it lacked the dynamic part of the context. This forced it to enter in the NC state and discard all types of packets until it received an IR packet. This test resulted in 2% packets loss. The third test case consisted in the loss of the IR-DYN packets. In this case, the *Decompressor* already knows the dynamic part of the context, so the *Decompressor* does not discard the following packets, but those packets are incorrect decompressed due to the outdated context. The fourth test case consisted in the loss of the UO-0 packets, since the UO-0 packets do not update the context. Only the UO-0 packets were lost. In this test, it was assumed that all UO-0 packets were lost, which led to 5% packet lost.

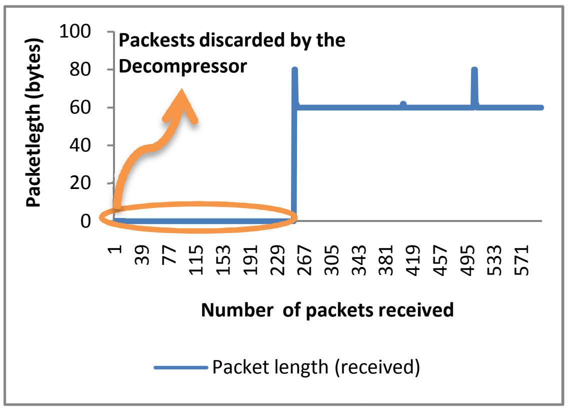
Test Case	**Compressor state**	**Decompressor state**	**Decompressor behaviour**
The loss of the initial IR packets	IR state	NC state	All packets are discarded until the *Compressor* returns to the IR state. This is because the *Decompressor* cannot start decompressing the packets without first creating the context. Therefore, if the *Decompressor* is in NC state, it will discard all packets until receiving IR packets.  *Figure 48- Discarded packets by Decompressor*
	FO state		
	SO state		
The loss of the initial IR-DYN packets	FO state	SC state	The loss of an IR-DYN RoHC packet can cause the *Decompressor* to fail decompressing the following packets, since it lacks the dynamic part of the context. This will force it to enter in the NC state and discard all types of packets until it receives an IR packet.
The loss of the IR-DYN packets	FO state	FC state	The loss of an IR-DYN RoHC packet can cause the context to contain incorrect data, which can lead to incorrect decompression of the following RoHC packets. The context can be repaired when the *Decompressor* receives an IR packet with a dynamic chain or an IR-DYN packet
The loss of the UO-0 packets.	SO state	FC state	Since the UO-0 packets do not update the context. Only the UO-0 packets will be lost.

Table XXV – Decompressor behaviour in the presence of loss of RoHC packets

5.5 Performance Tests

In order to measure, the performance of RoHC it was used a metric called the *RoHC Gain* [35]. By definition, the *RoHC Gain* is the amount of bandwidth that becomes available for other flows after applying RoHC to some flows. To test this it was used the wireless scenario. Series of experiments were conducted. Each experiment contained a number of VoIP calls established between the two nodes. The first experiment had a single VoIP call, the second two VoIP calls, and so on. Each experiment was repeated twice. One was using RoHC to compressed VoIP traffic and the other was using uncompressed VoIP traffic. A TCP flow was also established between the nodes as background traffic. The TCP flow was configured always to have data to send. The TCP congestion control mechanism adjusts the TCP flow bit-rate to all bandwidth available. The difference between the TCP bit-rate in the experiments where RoHC is not applied to the VoIP traffic and the TCP bit-rate in the experiments where RoHC is applied is the RoHC gain.

Ideally, this experiment should test a considerable number of VoIP calls to measure the RoHC Gain for a higher network loads. This was not possible due to limitations of the packet generator. These limitations consisted in, to each voice call, it was necessary to open a new session of the packet generator. With an increasing number of sessions opened, the packet generator did not perform well.

From the results gathered, was observed that the *RoHC Gain* increases with each VoIP call added. Figure 49 presents, for each VoIP call added the amount of bandwidth that became available for other flows after applying RoHC.

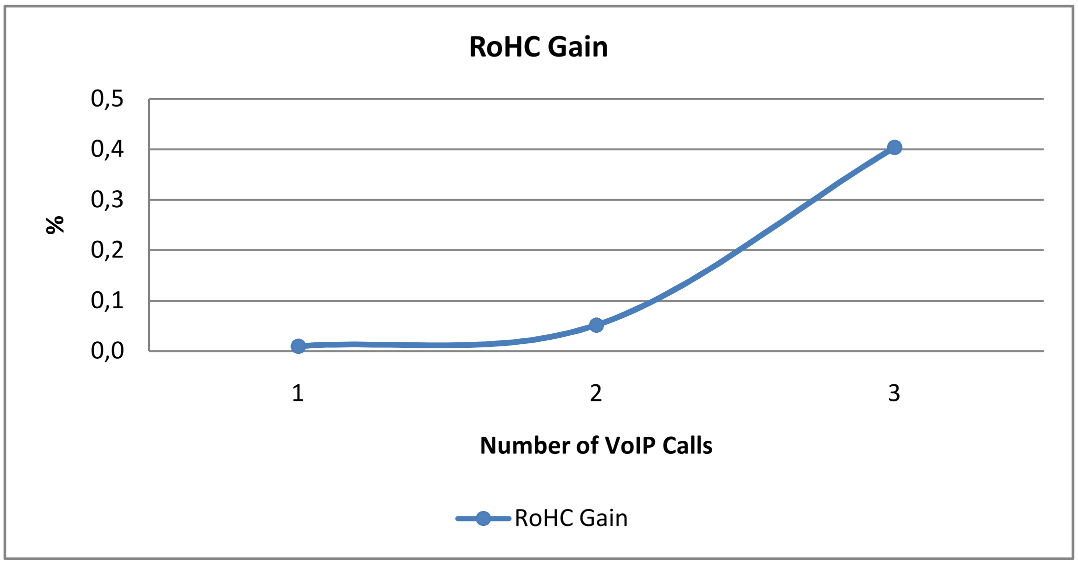


Figure 49- RoHC Gain

5.6 Conclusions

This chapter presented the functional and performance tests carried out. Two test scenarios were used. The first scenario, named the fixed scenario, consisted of two terminals connected by an IEEE 802.3 switch. The second scenario, named the wireless scenario, consisted of two terminals connected by an AP to an IEEE 802.11 network.

The functional tests goal was to validate the functional requirements of RoHC. These tests were divided in two set of tests. The first set of tests consisted in establishing a VoIP call between two nodes, with and without the application of RoHC, using the test two scenarios. In these tests, the application of RoHC reduced the bandwidth usage from an average of 8 kbps to an average of 4.6 kbps, in the fixed scenario, that equals to a 43% reduction. In the wireless scenario, it reduced the occupied bandwidth from an average of 8 kbps to an average of 2.2 kbps that equals to a 72% reduction. The addition of padding to the smaller packets in fixed scenario increased the number of bytes sent and lowed the reduction. In the wireless scenario there was no addition of padding, therefore the percentages of reductions were higher. The values presented for the wireless scenario are in line with values obtained in previous studies [8] [35] done in the area. Also measured in these tests were the delay and the jitter with and without the application of RoHC. The RoHC application, and especially when RoHC was in the higher state of compression, could improve the values of delay and jitter. In the second set of tests consisted in randomly drop packets, in order to verify if in U-Mode the *Decompressor* could recover from the loss propagation. Depending of the state of RoHC *Compressor* at the time of the flaw, the *Decompressor* could recover quickly with minimum packet loss or all packets would be discarded until the *Compressor* returned to the IR state and sent a packet with full headers, so the context could be updated or created. This allowed the *Decompressor* to decompress correctly the following packets.

The performance tests aimed to measure the performance of the RoHC U-mode implementation by using a metric call *RoHC Gain* [35]. The performance tests used the wireless scenario between the two nodes. It was sent simultaneously a greedy TCP flow and VoIP calls. The idea was to use the TCP congestion control mechanism of the TCP flow that would use the bandwidth left available after the application of RoHC. For each test performed, the number of VoIP increased. In each simulation run, an increasing number of VoIP flows were established between two nodes. This test was performed with three VoIP calls and, for each VoIP call added the value of the RoHC gain also increased. Due to problems with the traffic generator used, it was not possible to test with more than three VoIP calls.

6 Conclusion

This dissertation started by presenting the state of the art of header compression schemes that contribute to the development of RoHC, including cTCP, IPHC and CRTP. These schemes do not perform well over low bandwidth links or links having long round trip times. As a result of lost or out of order packets, the probability of an incorrect decompression increases. In order to address these limitations ITEF has developed RoHC.

The study of the RoHC protocol in this dissertation began by analyzing the RoHC standard. This specification includes the mains modes of operation, compression and decompression states, packets types and extensions sent in each state. Only one of RoHC the profiles was addressed: profile 0x0001, for sending compressed IP/UDP/RTP packets. The use of encoding methods by RoHC, that save additional bits, and its importance in packet compression and decompression, were also defined. These concepts were later applied to the development of the RoHC U-Mode implementation.

The study of the RoHC protocol continued with the study of RoHC over different types of networks. Solutions were presented to minimize the problem of large overhead over low bandwidth links, and the importance of RoHC in the future of these networks. We started by describing the architecture and the use of HC schemes, specially the use of RoHC on networks, such as GPRS, UMTS, IEEE 802.11 networks, sensors networks, and IEEE 802.16e networks.

The development of the RoHC U-mode implementation began by defining requirements, architecture and the choice of tools to use for development. The RoHC U-mode implementation was focused on the Profile 0x0001, for sending compressed IP/UDP/RTP packets. The implementation was made for a Linux environment, as an user space program, transparent to the application that generates the RTP traffic.

With the RoHC U-mode implementation, the functional and performance tests were carried out. The functional tests consisted in sending a VoIP call between the nodes of the fixed and wireless scenarios. The application of RoHC reduced the bit-rate from 8 kbps to 4.6 kbps in the fixed scenario that is a reduction of about 43%. A reduction of 72% of the IP headers was achieved in the wireless scenario, from 8 kbps to 2.2 kbps. The header compression ratio is higher in the wireless scenario because in the fixed scenario Ethernet padding is added to the IP payload. The tests also showed that using RoHC does not significantly worsen the delay and jitter and, in some cases, even improves them. In order to validate the behaviour of the *Decompressor* in the presence of packet loss, the same tests were executed again with packet drops being introduced intentionally. The results obtained showed that the context recovery mechanism of the U-mode function was implemented correctly. The *Decompressor* was able to recover the context in the different tests performed. The performance tests aimed to measure the performance of the RoHC U-mode implementation using a new metric called *RoHC Gain* [35]. The performance tests were also performed in the wireless scenario. This scenario was chosen because there is no addition of Ethernet padding to the payload. Between two nodes, a TCP flow was established and along with VoIP flows. The TCP congestion control mechanism of the TCP flow was used to measure the bandwidth left available after applying RoHC to the VoIP. In each simulation run, an increasing number of VoIP flows were established between two nodes. This test was performed with three VoIP calls and, for each VoIP call added, the value of the RoHC gain also increased. Due to problems with the traffic generator used, it was not possible to test with more than three VoIP calls.

6.1 Results

The main result of this dissertation is RoHC U-mode implementation, focused on the Profile 0x0001, used for sending compressed IP/UDP/RTP packets. This implementation was developed to run in a Linux environment.

6.2 Future Work

Several functionalities can be improved or added to the current RoHC U-mode implementation, namely:

- IP Encapsulation – Implement features able to compress and decompress encapsulated packets;
- Padding – Implement Ethernet padding removal, avoiding situations where this padding is propagated to non-Ethernet bridges (e.g. IEEE 802.11);
- Implement the O-mode and the R-Mode;
- Implement the re-utilization of the CID for short-lived flows.

Another possible improvement is to implement RoHC in a kernel module in order to provide transparency to the user applications.

References

[1] C. Borman et al, “Robust Header Compression (ROHC): Framework and four profiles: RTP, UDP, ESP, and uncompressed”, RFC 3095, IETF, July 2001.

[2] David J. Farber, Gary S. Delp, Thomas M. Conte, ”A Thinwire Protocol for connecting personal computers to the INTERNET“, September 1984.

[3] V. Jacobson,” Compressing TCP/IP Headers for Low-Speed Serial Links”, February 1990.

[4] L-E. Jonsson, K. Sandlund, G. Pelletier,P. Kremer,” RObust Header Compression (ROHC): Corrections and Clarifications to RFC 3095”, RFC 4815, IETF, February 2007.

[5] Pedro Fortuna, Manuel Ricardo, “Robust header Compression Over IEEE 802.11”

[6] A. Minaburo, L. Nuaymi, K.D. Singh, L. Toutain, “Configuration and Analysis of Robust Header Compression in UMTS”, in Proc. 14th IEEE International Symposium of PIMRC, Beijing, China, September 2003, Vol.3, pp.2402-2406.

[7] A. Minaburo, K. Singh, L. Toutain, L. Nuaymi, “Proposed Behaviour for Robust Header Compression over a radio link”, in IEEE ICC, June 2004, Paris, France.

[8] António Pedro Freitas Fortuna dos Santos,”Robust Header Compression in 4G Networks”, Faculdade de Engenharia Universidade do Porto, May 2007.

[9] Universal TUN/TAP Driver, http://vtun.sourceforge.net/tun/, 2008.

[10] Inside the Linux Packet Filter, http://www.linuxjournal.com/article/4852, 2008

[11] Jawad Ibrahim, “4G Features”, December 2002.

[12] F.H.P. Fitzek, S. Rein, P. Seeling, and M. Reisslein, “Robust Header Compression (RoHC) Performance for Multimedia Transmission over 3G/4G Wireless Networks”.

[13] packETH - ethernet packet generator, *http://packeth.sourceforge.net/, 2008*

[14] S. Poretsky, J. Perser, Veriwave, S. Erramilli, Telcordia ,S. Khurana, Motorola, “Terminology for Benchmarking Network-layer Traffic Control Mechanisms”, RFC 4689, October 2006.

[15] A. T. Connie, P. Nasiopoulos, V. C. M. Leung and Y. P. Fallah, “Video Packetization Techniques for Enhancing H.264 Video Transmission over 3G Networks”, 2008.

[16] Hyunje Woo, Jooyoung Kim, Meejeong Lee, JeongMin Kwon, “Performance analysis of Robust Header Compression over mobile WiMAX”, 2008.

[17] Chonggang Wang, Kazem Sohraby, Rittwik Jana, Lusheng Ji, and Mahmoud Daneshmand, “Voice Communications ZigBee Networks”, IEEE Communications Magazine , January 2008.

[18] Programming with pcap, http://netmirror.org/mirror/tcpdump.org/pcap.htm, 2008

[19] S. Casner, V. Jacobson, “Compressing IP/UDP/RTP Headers for Low-Speed Serial Links”, RFC 2508, IETF, February 1999.

[20] M. Degermark, B. Nordgren, S. Pink, “IP Header Compression “, RFC 2507, IETF, February 1999.

[21] M. Degermark, H. Hannu, L.E. Jonsson, K. Svanbro, “Evaluation of CRTP Performance over Cellular Radio Networks”, August 2000.

[22] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, “RTP: A Transport Protocol for Real-Time Applications”, RFC 3550, IETF, July 2003.

[23] G. Pelletier, K. Sandlund, "RObust Header Compression Version 2 (ROHCv2): Profiles for RTP, UDP, IP, ESP and UDP Lite", RFC 5225, April 2008.

[24] L-E. Jonsson, G. Pelletier, K. Sandlund, "The RObust Header Compression (ROHC) Framework", RFC 4995, July 2007.

[25] L-E. Jonsson and G. Pelletier, "RObust Header Compression (ROHC): A Compression Profile for IP", RFC 3843, June 2004.

[26] L-E. Jonsson, G. Pelletier and K. Sandlund, "RObust Header Compression (ROHC): A Link-Layer Assisted Profile for IP/UDP/RTP", RFC 4362, January 2006.

[27] Z. Liu and K. Le, "0-byte Support for R-mode in Extended Link-Layer Assisted ROHC Profile", RFC 3408, November 2002.

[28] G. Pelletier, L. Jonsson, K. Sandlund and M. West, "RObust Header Compression (ROHC): A Profile for TCP/IP (ROHC-TCP)", RFC 4996, July 2007.

[29] G. Pelletier, "RObust Header Compression (ROHC): Profiles for UDP-Lite", RFC 4019, April 2005.

[30] G. Montenegro, N. Kushalnagar, J. Hui , D. Culler, “Transmission of IPv6 Packets over IEEE 802.15.4 Networks”, RFC 4944, September 2007.

[31] 3GPP2, “Ultra Mobile Broadband (UMB) Air Interface Specification,” C.S0084 v. 2.0, September 2007.

[32] Sean McBeath, Jack Smith, Linhong Chen, and Anthony C. K. Soong, Huawei Technologies,Hao Bi, Motorola Inc, “VoIP Support Using Group Resource Allocation Based on the UMB System”, IEEE Communications Magazine, January 2008.

[33] 3GPP, General Packet Radio Service (GPRS); Service Description; Stage 2 (Release 7); TS 23.060 v7.0.0, March 2006.

[34] IEEE 802.16e-2005, "IEEE Standard for Local and metropolitan area networks - Part 16: Air Interface for Fixed and Mobile Broadband Wireless Access Systems", February 2006.

[35] Pedro Fortuna, Manuel Ricardo,” Header Compressed VoIP in IEEE 802.11”, Wireless Communications, IEEE, July 2009.

[36] IEEE 802.11,”IEEE 802.11 Wireless Local Area Networks”.

[37] 3GPP, General Universal Mobile Telecommunications System (UMTS) architecture (Release 6); TS 23.101 v6.0.0, December 2004.

[38] 3GPP, Packet Data Convergence Protocol (PDCP) specification (Release 7); TS 25.323 v7.0.0, March 2006.

[39] ITU-T G.729a codec, http://www.itu.int/rec/T-REC-G.729/en, 2008.

[40] Wireshark, http://www.wireshark.org/, 2008.

[41] ITU-T Recommendation G.114, Series G: Transmission Systems and Media, Digital Systems and Networks, May 2003.

Annex

A1. General Format of the Compressor

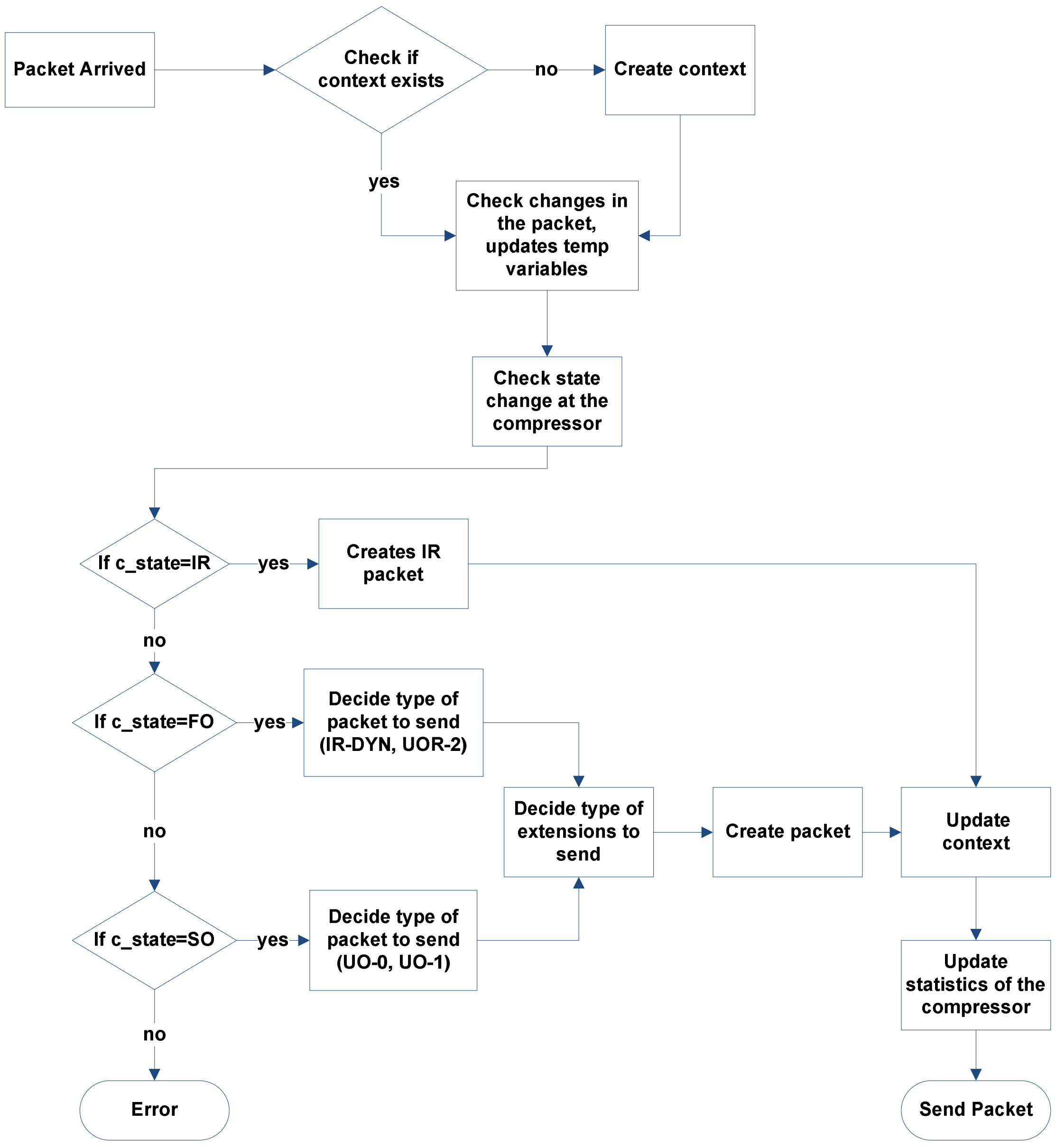

Packet Arrived
Check if context exists
no
Create context
yes
Check changes in the packet, updates temp variables
Check state change at the compressor
If c_state=IR
yes
Creates IR packet
no
If c_state=FO
yes
Decide type of packet to send (IR-DYN, UOR-2)
no
Decide type of extensions to send
Create packet
Update context
If c_state=SO
yes
Decide type of packet to send (UO-0, UO-1)
Update statistics of the compressor
no
Error
Send Packet

Figure 50- General Format of the compressor

A2. The FO state

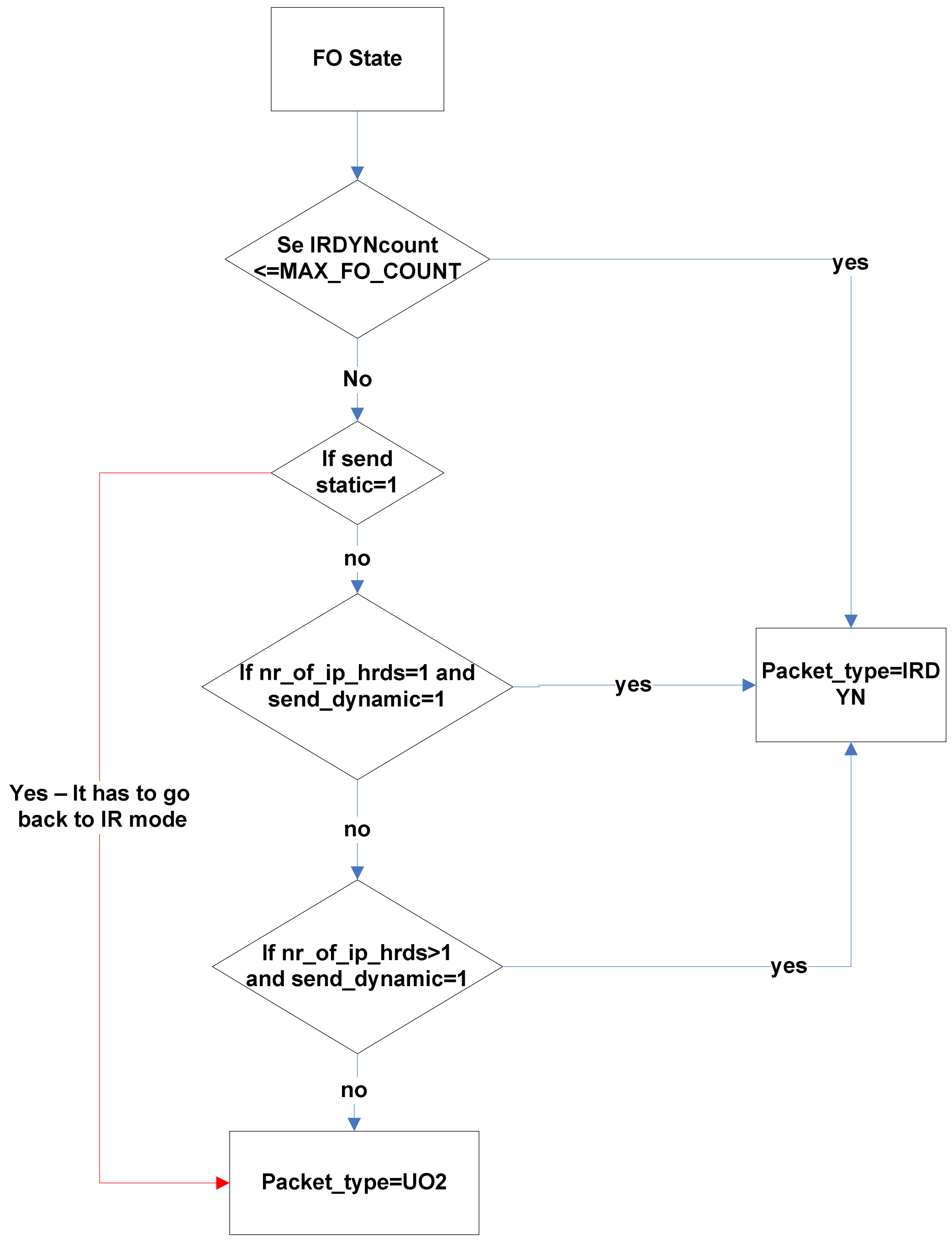

FO State
Se IRDYNcount <=MAX_FO_COUNT
yes
No
If send static=1
no
If nr_of_ip_hrds=1 and send_dynamic=1
yes
Packet_type=IRDYN
Yes – It has to go back to IR mode
no
If nr_of_ip_hrds>1 and send_dynamic=1
yes
no
Packet_type=UO2

Figure 51- the FO state

A3. The SO state

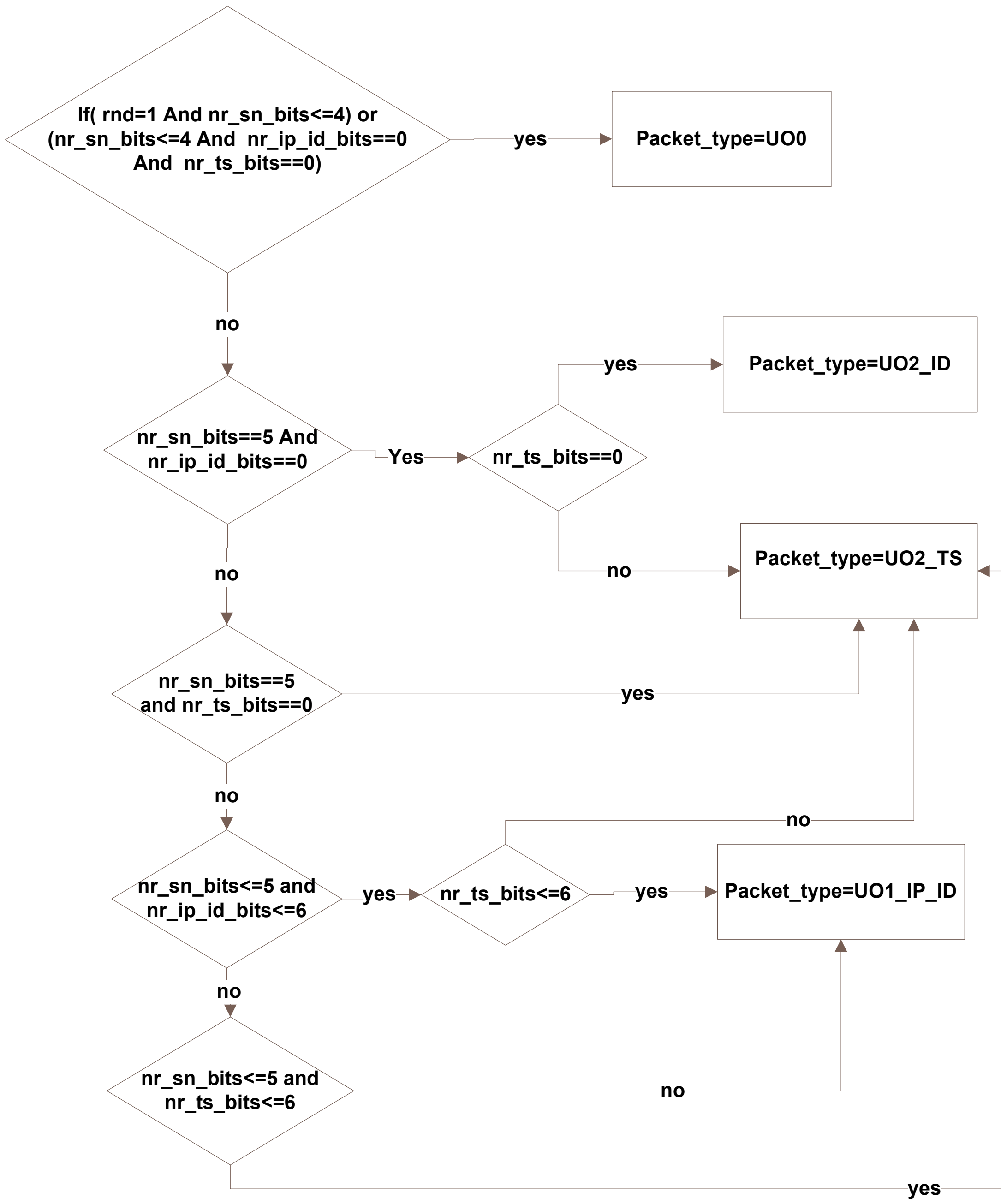

If(rnd=1 And nr_sn_bits<=4) or (nr_sn_bits<=4 And nr_ip_id_bits==0 And nr_ts_bits==0)
yes
Packet_type=UO0
no
yes
Packet_type=UO2_ID
nr_sn_bits==5 And nr_ip_id_bits==0
Yes
nr_ts_bits==0
no
no
Packet_type=UO2_TS
nr_sn_bits==5 and nr_ts_bits==0
yes
no
no
nr_sn_bits<=5 and nr_ip_id_bits<=6
yes
nr_ts_bits<=6
yes
Packet_type=UO1_IP_ID
no
nr_sn_bits<=5 and nr_ts_bits<=6
no
yes

Figure 52- the SO state

A4. Extensions

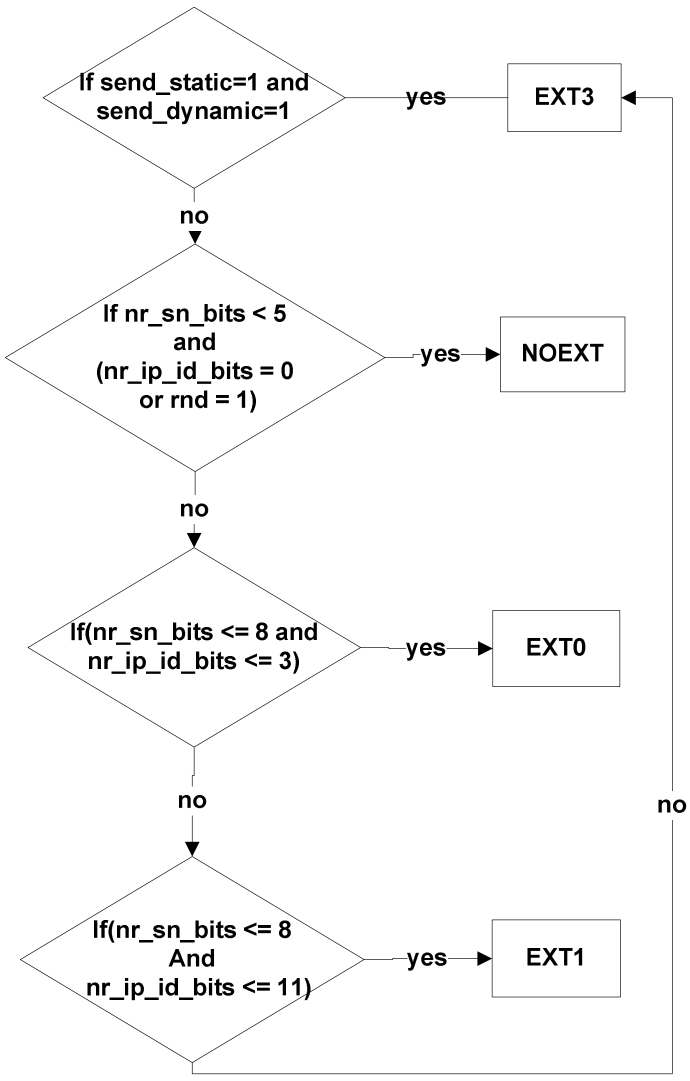

If send_static=1 and send_dynamic=1
yes
EXT3
no
If nr_sn_bits < 5 and (nr_ip_id_bits = 0 or rnd = 1)
yes
NOEXT
no
If(nr_sn_bits <= 8 and nr_ip_id_bits <= 3)
yes
EXT0
no
no
If(nr_sn_bits <= 8 And nr_ip_id_bits <= 11)
yes
EXT1

Figure 53- Extensions

A5. State transitions

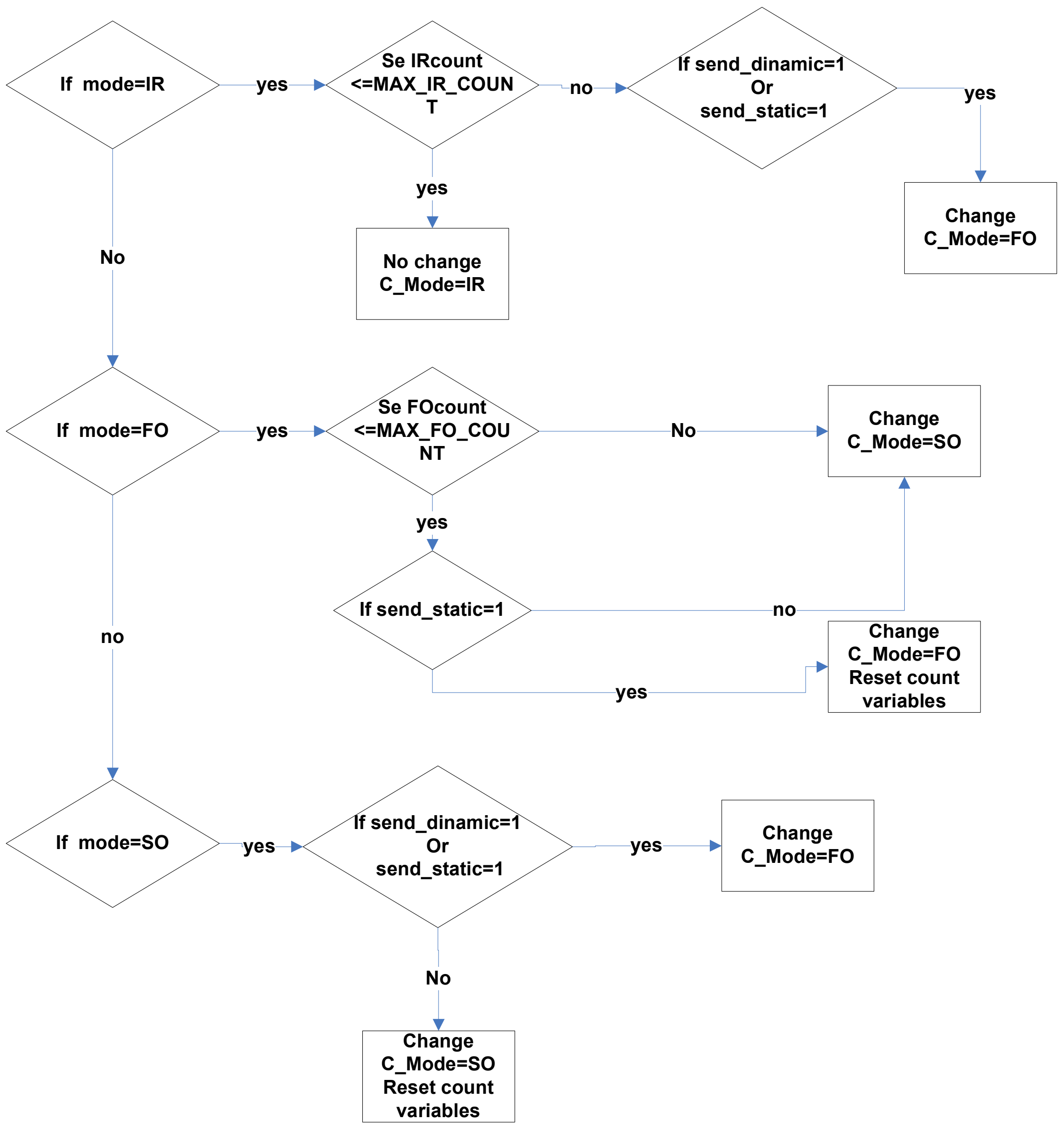

If mode=IR
yes
Se IRcount <=MAX_IR_COUNT
no
If send_dinamic=1 Or send_static=1
yes
Change C_Mode=FO
yes
No change C_Mode=IR
No
If mode=FO
yes
Se FOcount <=MAX_FO_COUNT
No
Change C_Mode=SO
yes
If send_static=1
no
Change C_Mode=FO Reset count variables
yes
no
If mode=SO
yes
If send_dinamic=1 Or send_static=1
yes
Change C_Mode=FO
No
Change C_Mode=SO Reset count variables

Figure 54- State transitions

A6. Decompressor

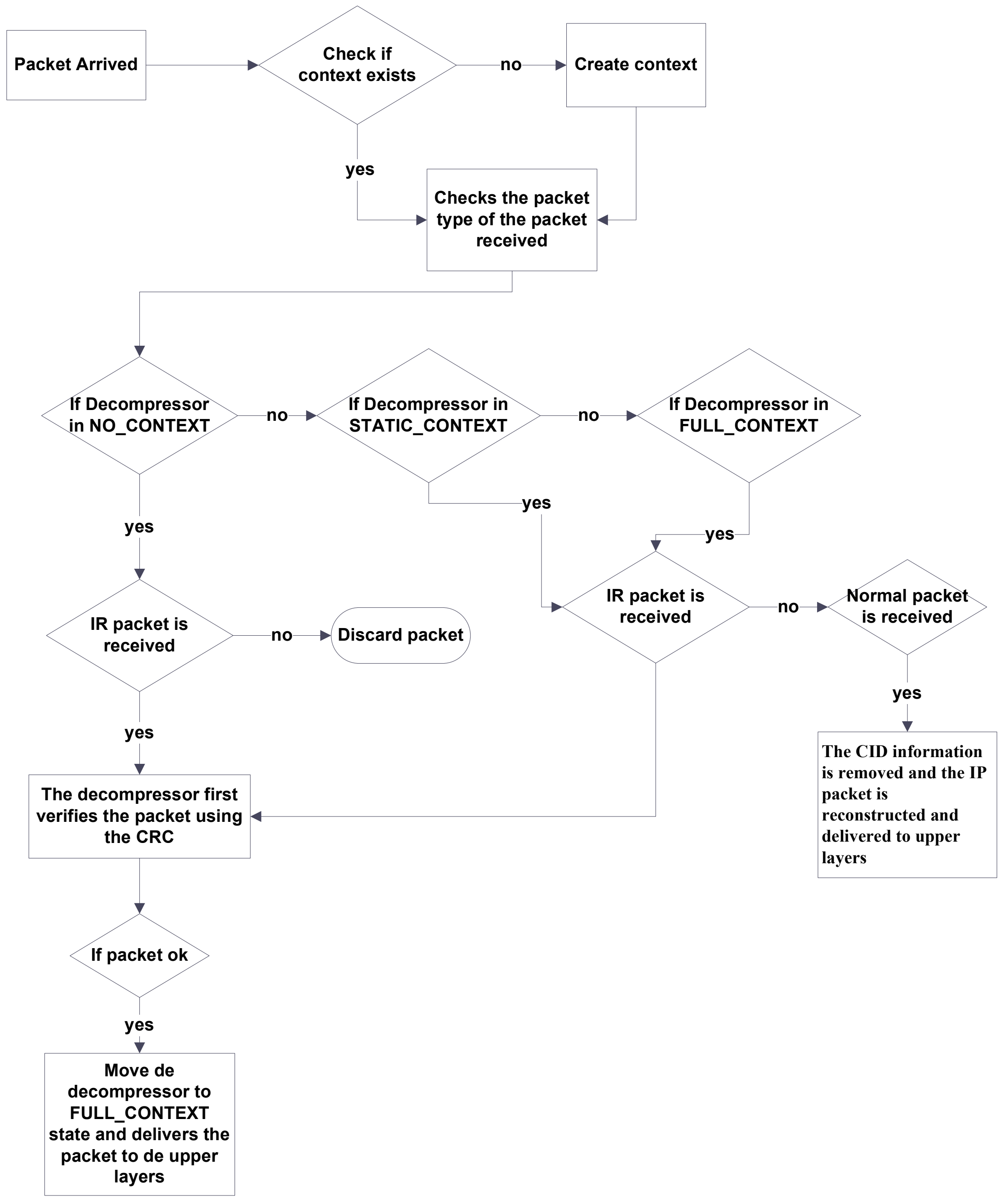

Packet Arrived
Check if context exists
no
Create context
yes
Checks the packet type of the packet received
If Decompressor in NO_CONTEXT
no
If Decompressor in STATIC_CONTEXT
no
If Decompressor in FULL_CONTEXT
yes
yes
yes
IR packet is received
no
Normal packet is received
IR packet is received
no
Discard packet
yes
yes
The decompressor first verifies the packet using the CRC
The CID information is removed and the IP packet is reconstructed and delivered to upper layers
If packet ok
yes
Move de decompressor to FULL_CONTEXT state and delivers the packet to de upper layers

Figure 55- Decompressor

A7. Constants

In order to facilitate the development of the RoHC U-mode implementation several constants were defined and are listed next.

Constant	Value	Description
UMODE	1	These constants represent the RoHC modes of operation. Although this implementation is limited to the U-mode, the other constants were added to accommodate future development.
OMODE	2	
RMODE	3	
IR	1	These constants represent the states of the RoHC *Compressor*: IR, FO, SO.
FO	2	
SO	3	
NO_CONTEXT	1	These constants represent the states of the RoHC *Decompressor*: NO, SC, FC.
STATIC_CONTEXT	2	
FULL_CONTEXT	3	
ETH_P_ROHCS	0x8901	This constant identifies the ethertype. In this case, a compressed RoHC packet. This number does not collide with the ethertypes defined in the include file netinet/in.h.
SIZE_ETHERNET	14	The sizes of the headers that compose the RTP packet: SIZE_ETHERNET, SIZE_IP, SIZE_UDP e SIZE_RTP.
SIZE_IP	20	
SIZE_UDP	8	
SIZE_RTP	12	
TYPE_IR	0xfc	These constants represent the packet types identifiers of the compressed packets.
TYPE_IR_	0xfd	
TYPE_IRDYN	0xf8	
TYPE_UO0	0x0	
TYPE_UO1_TS	0x5	
TYPE_UO1_ID	0x4	
TYPE_UO2_TS	0x6	
TYPE_UO2_ID	0x61	
EXT0	0	These constants represent the extensions identifiers of the compressed packet types.
EXT1	1	
EXT2	3	
NOEXT	-1	
CRC_TYPE_3	1	These constants indicate the type of CRC used: 3, 7 or 8 bits.
CRC_TYPE_7	2	
CRC_TYPE_8	3	
C_WINDOW_WIDTH	16	The size of the sliding window used.
TIMEOUT_IR	5	The values used in the U-mode timeouts.
TIMEOUT_FO	3	

Table XXVI – Constants

Index